UNIVERSITY OF CALIFORNIA, SAN DIEGO

Domain Partitioning Methods for Elliptic Partial Differential Equations

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Mathematics w/ Specialization Computational Science

by

Christopher George Deotte

Committee in charge:

Professor Randolph E. Bank, Chair
Professor Scott B. Baden
Professor David J. Benson
Professor Michael Holst
Professor Melvin Leok

2014

The dissertation of Christopher George Deotte is approved, and it is acceptable in quality and form for publication on microfilm:

_____

_____

_____

_____

_____
Chair

University of California, San Diego

2014

DEDICATION

To God, family, and friends.

EPIGRAPH

With God all things
are possible.
—Jesus

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Graduate research and writing are difficult work. Among the joyful moments of discovery and success are discouraging setbacks and obstacles. Indispensable is the support from God, family, friends, and teachers. I especially thank God for his love and blessings.

I am thankful for so many caring and helpful people in my life. I thank my mother, father, and brother for their love and support. My mother is an incredible source of strength and encouragement. Without her, I doubt I would have finished my PhD. Thank you mom.

Thank you Janina, my girlfriend, for your love and support. The final months were difficult but you made them easier.

Randy Bank is a brilliant mathematician and great mentor. He is always available whenever I need help or want to share results. No matter what obstacles we face, he has insights and suggestions which keep us moving forward. Thank you Randy. Also I'm grateful for his software package PLTMG 11.0 which enables us to conduct exciting large scale numerical experiments.

Thank you Mike Holst for your friendship and optimism. If I entered your office discouraged, I would leave hopeful and energized. Your encouragement got me to the finish line. Thank you David Benson for introducing me to Finite Elements. Thank you Scott Baden for introducing me to large scale parallel computing. Thank you Melvin Leok for helping on my committee. And thanks James Bunch for teaching me numerical linear algebra.

Wilson Cheung can solve any computer problem; thanks Wilson. Thank you Holly Proudfoot, Scott Rollins, Lois, Terry, and Janice for all your administration help.

It is a blessing to be a part of CSME and CCoM. These groups generously provide their members with many opportunities and resources including the supercomputer BOOM.

The UCSD Math Department is a wonderful community. We enjoy weekly ultimate frisbee games, board game meet ups, frequent pot luck dinners, and other fun get togethers. Thanks math friends. Thanks Andy and Helen, Janine

and Mark, Caleb and Leslie, Jimmy and Katie, Mike and Courtney, Jake and Suz, Andy Parrish, Franklin, Brian, Susan, Alex Brik, and others.

My roommates Tim Banham, Pete Overholser, Mike Kelley, and Gordon Honerkamp-Smith are great people and great friends. I am grateful to have them in my life. The five of us struggled and celebrated graduate school together. We shared lots of adventures together including a 100 mile hike along the Sierra High Trail in northern California.

Thank you Jahir Orozco-Holguin and Taylor Oliver for your encouragement. Your friendship is a blessing and I appreciate the example you provide of balancing work and play. You are role models in your respective research fields and fun people who enjoy dancing and playing volleyball.

The toughest class I took at UCSD was Real Analysis with Lei Ni. It was also a qualifying exam class for my PhD. My friend Jesus Oliver helped me get through this course. We would meet regularly to discuss the material and help each other solve problems. This was one of the best math collaboration experiences of my life. Thanks Jesus.

Preparing, applying, and interviewing for jobs is a big endeavor. Thanks Pete Spragins for sharing the job hunting experience with me and thanks for being a good friend. Winning the Intuit hackathon with you was loads of fun. I always enjoy our programming discussions. Have fun at Google.

I thank my athletic friends Sean Mowen, Mark Olinger, Lauren Smith, and Patrick Hauf who share playing volleyball and windsurfing with me. Without regularly playing sports, I wouldn't be able to focus on my work. And thanks to Lyla Fadali, David Zimmermann, and Hooman Sherkat for letting me spend so much time at your apartments. Hanging out is fun.

Thank you to the community at http://play-agricola.com for providing me with creative outlets, entertainment, and stimulating discussions. And finally, thank you to San Diego's Punjabi Tandoor restaurant for supplying me with hundreds of fantastic Indian dinners.

VITA

1990-1995	Party Entertainer for my business, Ultimate Bass
1998-2000	Rock Climbing Instructor for Furman University, South Carolina
1997-2007	Designed Restaurant and Entertainment Guides for my business, City Guides
1995-2000	**BA - Mathematics** Cornell University and Furman University - Major GPA: 4.0
2002-Present	Buy, Sell, Lease, and Renovate Houses for my business, BTGB
2005-2007	Robotics Coach and High School Mathematics Teacher for Greenville High, South Carolina
2007-Present	Mathematics Instructor and Mathematics TA for UCSD, California
2007-2009	**MA - Applied Mathematics** UCSD - GPA: 3.9
2007-Present	Scientific Software Developer as UCSD PhD student
2012-2013	Data Analyst and Software Developer for Attorney General's Economic Crimes Division, Florida
2008-Present	Data Analyst and Software Developer for Lookout Games, Germany
2009-2014	**PhD - Mathematics w/ Specialization Computational Science** UCSD - GPA: 4.0

ABSTRACT OF THE DISSERTATION

Domain Partitioning Methods for Elliptic Partial Differential Equations

by

Christopher George Deotte
Doctor of Philosophy in Mathematics w/ Specialization Computational Science
University of California San Diego, 2014
Professor Randolph E. Bank, Chair

Numerically solving elliptic partial differential equations for a large number of degrees of freedom requires the parallel use of many computer processors. This in turn requires algorithms to partition domains into subdomains in order to distribute the work.

In this dissertation, we present five novel algorithms for partitioning domains that utilize information from the underlying PDE. When a PDE has strong convection or anisotropic diffusion, directional dependence exists and a partition that favors this direction is desirable. Our schemes fall into two classes; one class creates rectangular shaped subdomains aligned in this direction and one class creates subdomains that increase in size as you move in this direction.

These schemes are mathematically described and analyzed in detail. Then they are tested on a variety of experiments which include solving the convection-diffusion equation for $\frac{1}{4}$ billion unknowns on 512 processors using over 1 teraflop of computing power.

Theory and experiments demonstrate that these schemes improve the domain decomposition convergence rate when the underlying PDE has directional dependence. In our hundreds of experiments, the number of DD iterations required for convergence reduces by a factor between 0.25 and 0.75. And these methods maintain or improve the final finite element solution's accuracy also.

Chapter 1

Introduction

Frequently, in order to model reality, scientists wish to solve second order partial differential equations. After discretizing their domains of interest, they need to solve for billions of unknown values. Even with modern computing power, this is a daunting task which may require days, weeks, or months to complete. Consequently, many computer processors are used simultaneously. Algorithms are needed to divide the many unknowns among the many processors. Different algorithms affect both the time needed to find the unknowns and the accuracy of the computed values. In this thesis, we present new algorithms to distribute these unknowns which allow the unknowns to be found faster and more accurately than existing methods.

1.1 Problem Definition

The problem we are interested in solving is the second order elliptic boundary value problem

$$-\nabla \cdot (a(x,y)\nabla u) + b(x,y) \cdot \nabla u + c(x,y)u - f(x,y) = 0 \qquad \text{in } \Omega \tag{1.1}$$

with boundary conditions

$$a(x,y)\nabla u \cdot n = g_N(x,y) \qquad \text{on } \partial\Omega_N \tag{1.2}$$

$$u = g_D(x,y) \qquad \text{on } \partial\Omega_D \tag{1.3}$$

Here $\Omega \in \mathbb{R}^d$ is a bounded domain, n is the unit normal vector, a is a $d \times d$ spd matrix, b is a vector of length d, and $[a]_{i,j}$, $[b]_i$, c, f, g_N, and g_D are scalar functions.

1.2 Overview

Chapters 2-4 walk the reader through a typical procedure for solving (1.1)-(1.3). In summary, you convert the above Strong Form of your PDE into a Weak Variational Form and then apply a Galerkin Method. This reduces a problem of infinite unknowns to one of finite unknowns. A Galerkin Approximation commonly uses Finite Elements to define its finite dimensional space. The chosen elements determine the quantity and location of the degrees of freedom (unknowns). Next, in the process of Domain Decomposition, you utilize a Partitioning Algorithm and assign each element and its associated degrees of freedom to one or more of your multiple processors. Afterward, the processors begin solving for the unknowns. This is generally an iterative process that requires repeated local computation and communication between processors. When this procedure finishes, you combine the computed values from all the processors together and have a finite approximation to the solution of (1.1)-(1.3).

This is a popular method because computers do it well and it has been shown that this approximate solution converges to the true solution as you increase the number of unknowns and decrease h, the element size.

$$\|u - u_h\|_{\alpha,\Omega} \leqq h^{2-\alpha}\|u\|_{2,\Omega} \tag{1.4}$$

This was shown by Babuska and Aziz in [1].

1.3 Our Contributions

Standard Partitioning Algorithms don't use information from the original equations (1.1)-(1.3) nor the intermediate solutions during iteration. But, these equations and solutions contain valuable information about the dependencies between the unknowns you desire. We found that by using this information, you can

group dependent degrees of freedom together on the same processor and minimize the dependence between processors. Intuitively, this is a better division of labor (partition) since each processor requires less knowledge about the unknowns on other processors and can spend more time focused on solving for its own unknowns. Our theory and experiments show that distributing unknowns among processors in this fashion speeds up the convergence of Domain Decomposition Methods which saves computing time and therefore saves money. These new divisions of labor also maintain or improve the accuracy of the final solution.

In Chapters 5 and 7, we reveal five ways to incorporate information from the original equations and intermediate solutions into partitioning algorithms.

In Chapter 8, we provide mathematical analysis that proves the benefit of these methods in simple cases and in Chapter 9, we provide experimental numerical results demonstrating the improvement in a variety of cases.

Chapter 10 outlines ideas for future research. All the work in Chapters 5,7,8 and 9 is original.

Chapter 2

Finite Elements

A typical procedure for solving the second order elliptic PDE (1.1)-(1.3) with a computer is to convert it into its Weak Form and use a Galerkin Approximation and Finite Elements [37] [21] [31].

2.1 Overview

2.1.1 Weak Form

Let $H^1(\Omega)$ denote the usual Sobolev space. Let

$$S = \{\phi \in H^1(\Omega) \,|\, \phi = g_D \text{ on } \partial\Omega_D\} \tag{2.1}$$

$$V = \{\phi \in H^1(\Omega) \,|\, \phi = 0 \text{ on } \partial\Omega_D\} \tag{2.2}$$

Then the weak form of (1.1)-(1.3) is: find $u \in S$ such that

$$a(u, v) = b(v) \quad \forall v \in V, \tag{2.3}$$

where

$$a(u, v) = \int_\Omega a(x, y)\nabla u \cdot \nabla v + (b(x, y) \cdot \nabla u)v + c(x, y)uv \, dx \, dy \tag{2.4}$$

$$b(v) = \int_\Omega f(x, y)v \, dx \, dy \; + \int_{\partial\Omega_N} g_N(x, y)v \, ds \tag{2.5}$$

2.1.2 Galerkin Approximation

The solution of (2.3) has an infinite number of degrees of freedom. A computer prefers to solve for a finite number of unknowns. A Galerkin Method solves (2.3) by constructing finite dimensional approximations of S and V and finding a finite dimensional u which approximates the true solution.

The new function spaces are made by partitioning Ω into Finite Elements. Let T_h denote a triangulation of Ω and let M_h be the space of C^0 piecewise linear functions associated with T_h.

2.1.3 Lagrange Basis Functions

Form a basis of linear functions from the standard linear Lagrange nodal functions with degrees of freedom placed at the triangles' vertices.

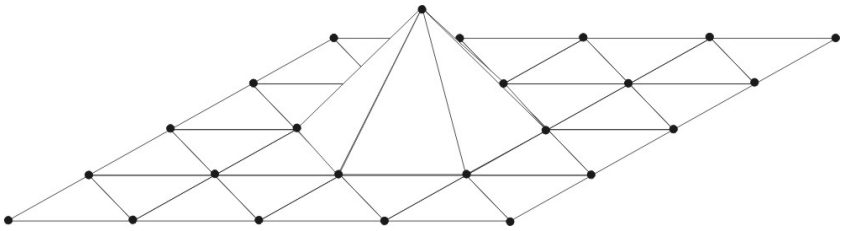


Figure 2.1: A linear Lagrange basis function v_k on a uniform triangle mesh.

2.1.4 Finite Element Form

We can now formulate a discrete analog of (2.3). Let $I : H^1(\Omega) \to M_h$ denote a continuous piecewise linear function interpolation operator that interpolates at the degrees of freedom of T_h. Let

$$S_h = \{\psi \in M_h \,|\, \psi = I(g_D) \text{ on } \partial\Omega_D\} \tag{2.6}$$

$$V_h = \{\psi \in M_h \,|\, \psi = 0 \text{ on } \partial\Omega_D\} \tag{2.7}$$

Then our discrete Finite Element form of (2.3) is: find $u_h \in S_h$ such that

$$a(u_h, v_h) = b(v_h) \quad \forall v_h \in V_h \tag{2.8}$$

where a and b are defined above in (2.4) and (2.5). If the dimension of S_h is n, then (2.8) is a system of n equations.

2.1.5 Matrix Form

If the underlying partial differential equations (1.1)-(1.3) are linear, then these equations are n linear equations which can be represented with matrix notation. Equation (2.8) can be written as

$$AU = F \tag{2.9}$$

where A is a matrix commonly referred to as the *stiffness matrix*, $A \in \mathbb{R}^{n\times n}$ and $[A]_{j,k} = a(\psi_k, \psi_j)$ where $\{\psi_k \,|\, k = 1, 2, 3..., n\}$ are the basis functions of V_h. The vector $U \in \mathbb{R}^n$ are the degrees of freedom of u_h and are the coefficients of the basis functions when representing u_h as

$$u_h = u_D + \sum_{k=1}^{n} U_k \psi_k \tag{2.10}$$

for some $u_D \in S_h$. The vector $F \in \mathbb{R}^n$ is $[F]_i = b(\psi_i) - a(u_D, \psi_i)$ and is referred to as the *load vector*.

2.2 Upwinding

When the underlying partial differential equation is not self-adjoint, the discretization in (2.8) needs some upwinding terms added to improve stability. Therefore instead of solving (2.8), we would solve

$$a_h(u_h, v_h) = b(v_h) \quad \forall v_h \in V_h \tag{2.11}$$

A simple form of upwinding is artificial diffusion

$$a_h(u_h, v_h) = a(u_h, v_h) + \frac{\alpha||b||h}{2} \int_\Omega w \nabla u_h \cdot \nabla v_h \, dxdy \tag{2.12}$$

where b is the coefficient of ∇u in the original PDE, α is a damping constant, and $w \in \mathbb{R}^{d\times d}$ which depends on b and the mesh.

There are various ways of adding diffusion. Two popular methods are Scharfetter Gummel Upwinding and Streamline Diffusion [3] [4]. Scharfetter Gummel Upwinding behaves as (2.12) and Figure 2.2 shows how much diffusion Scharfetter

Gummel adds to the convection diffusion equation $-\triangle u + b \cdot \nabla u - 1 = 0$ being solved on a uniform equilateral triangle mesh based on convection strength. The y axis is α from (2.12). In each case, $w \approx \text{diag}(b/||b||)$.

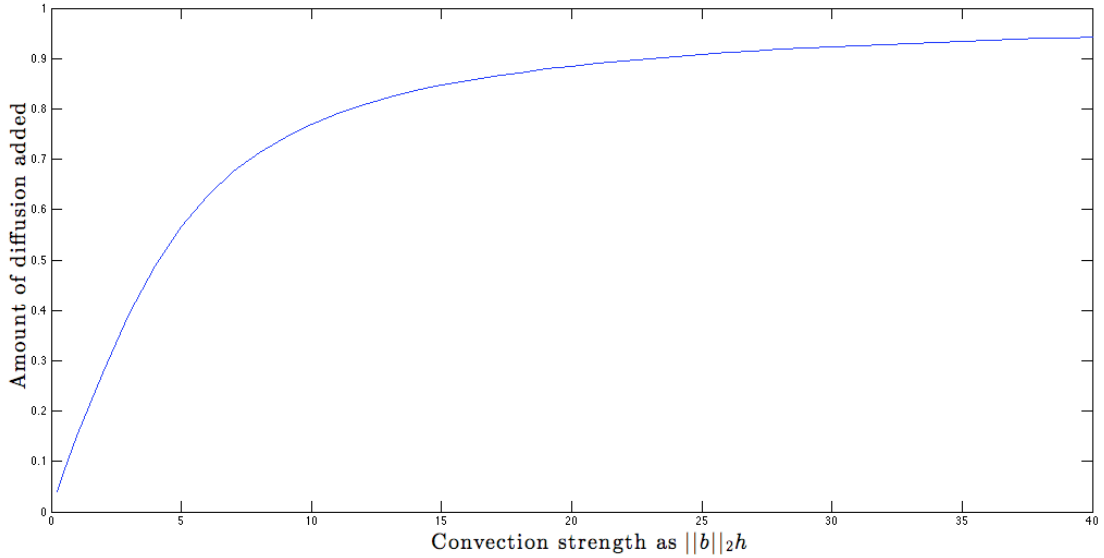


Figure 2.2: Scharfetter Gummel upwinding

2.3 Nonlinear PDE

When the underlying partial differential equation is nonlinear, Newton's Method can be used to solve (2.8). Let the vector U correspond to the degrees of freedom of the Finite Element solution u_h as defined in (2.10), then (2.8) can written as a system of nonlinear equations

$$G(U) = 0 \tag{2.13}$$

where the j^{th} nonlinear equation of $G(U) = 0$ is $a\left((u_D + \sum_{k=1}^{n} U_k \psi_k), \psi_j\right) - b(\psi_j) = 0$. The Jacobian matrix for this system is

$$A(U) = \frac{\partial G(U)}{\partial U} \tag{2.14}$$

Start with an initial guess U_k for $k = 0$. Calculate the residual $R_k = -G(U_k)$. Then solve the following system of linear equations for δU_k

$$A_k(\delta U_k) = R_k. \tag{2.15}$$

Afterward, $U_{k+1} = U_k + s_k \delta U_k$ where s_k is a damping factor. Calculate the new residual, and repeat this procedure until a convergence criteria is met.

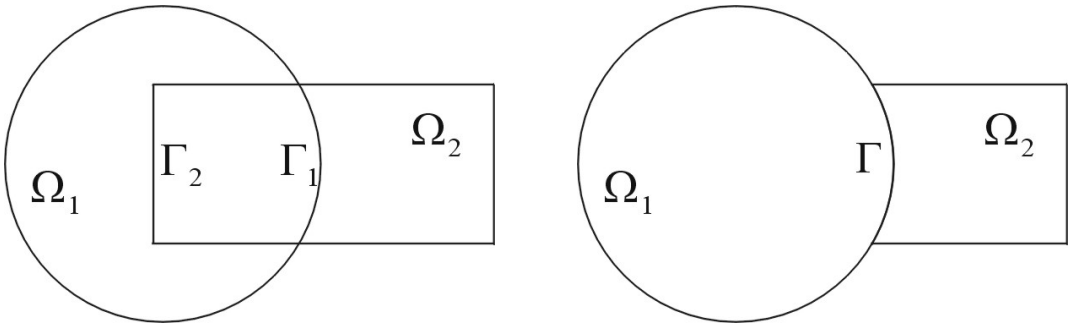
Chapter 3

Domain Decomposition

The Strong Form, (1.1)-(1.3), is difficult to solve on a computer, but computers excel at solving the Matrix Form, (2.9), and Newton Method's Form, (2.15). In practice, these forms are used to tackle the solution of real world problems.

Although these two forms are more tractable, they are still difficult to solve because each can have billions of unknowns to compute. This proves difficult for one computer to deal with. A reasonable approach is to distribute the unknowns over many processors (computers) and let them work together.

Given a domain Ω, you can divide the domain into overlapping subdomains or non-overlapping subdomains and then use a Domain Decomposition method [39] [43] [44].



(a) Overlapping (b) Non-overlapping

Figure 3.1: Subdomain Types

3.1 Overlapping Subdomains

The simplest and oldest Domain Decomposition Methods are the overlapping Schwartz Methods [43] with the two most popular being the Multiplicative Schwarz Method and the Additive Schwarz Method.

3.1.1 Schwarz Framework

The Multiplicative Schwarz Method works as follows. In Figure 3.1a, let $\Omega = \Omega_1 \cup \Omega_2$ and note that the boundary of Ω_1 is $(\partial\Omega_1 \setminus \Gamma_1) \cup \Gamma_1$. One part of $\partial\Omega_1$ is the original boundary of Ω and one part is a new artificial boundary. We wish to solve (1.1)-(1.3) on Ω. In order to simplify the language in this section, define L as the differential operator from (1.1)

$$Lu \equiv -\nabla \cdot (a(x,y)\nabla u) + b(x,y) \cdot \nabla u + c(x,y)u \tag{3.1}$$

and define B as the boundary operator from (1.2)-(1.3)

$$Bu \equiv \begin{cases} a(x,y)\nabla u \cdot n \text{ on } \Omega_N \\ u \text{ on } \Omega_D \end{cases} \tag{3.2}$$

The Multiplicative Schwarz Method, also referred to as the Alternating Schwarz Method, proceeds by alternating solving $Lu_1 = f_1$ on Ω_1 and $Lu_2 = f_2$ on Ω_2 with appropriate boundary conditions.

Algorithm 1 Multiplicative Schwarz Method

Initialize $u_2^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

2: Find $u_1^k \in H^2(\Omega_1)$ such that

$$Lu_1^k = f_1 \text{ on } \Omega_1$$
$$Bu_1^k = g_1 \text{ on } \partial\Omega_1 \setminus \Gamma_1$$
$$u_1^k = u_2^{k-1} \text{ on } \Gamma_1$$

3: Find $u_2^k \in H^2(\Omega_2)$ such that

$$Lu_2^k = f_2 \text{ on } \Omega_2$$
$$Bu_2^k = g_2 \text{ on } \partial\Omega_2 \setminus \Gamma_2$$
$$u_2^k = u_1^k \text{ on } \Gamma_2$$

4: **end for**

In Chapter 8, we show that the discrete version of this method, resulting from using a Galerkin approximation, is equivalent to a block Gauss-Seidel Method applied to that same system of equations.

Multiplicative Schwarz, Algorithm 1, is a serial algorithm. Steps 2 and 3 cannot be performed simultaneously. The parallel version of Algorithm 1 is the Additive Schwarz Method.

Algorithm 2 Additive Schwarz Method

Initialize $u_1^{(0)}$ and $u_2^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

Perform steps 2 and 3 simultaneously on different processors.

2: Find $u_1^k \in H^2(\Omega_1)$ such that

$$Lu_1^k = f_1 \text{ on } \Omega_1$$
$$Bu_1^k = g_1 \text{ on } \partial\Omega_1 \setminus \Gamma_1$$
$$u_1^k = u_2^{k-1} \text{ on } \Gamma_1$$

3: Find $u_2^k \in H^2(\Omega_2)$ such that

$$Lu_2^k = f_2 \text{ on } \Omega_2$$
$$Bu_2^k = g_2 \text{ on } \partial\Omega_2 \setminus \Gamma_2$$
$$u_2^k = u_1^{k-1} \text{ on } \Gamma_2$$

4: Communicate u_i from Γ_j to neighbor processors.

5: **end for**

In Chapter 8, we show that the discrete version of this method resulting from using a Galerkin approximation is equivalent to a block Jacobi Method applied to that same system of equations.

3.2 Non-overlapping Subdomains

When the subdomains are disjoint such as Figure 3.1b, requiring that $u_1 = u_2$ on the interface Γ and then solving for each u_k on its respective domain is not enough to determine a solution to (1.1)-(1.3). We most impose both $u_1 = u_2$ and $a\nabla u_1 \cdot n_\Gamma = a\nabla u_2 \cdot n_\Gamma$ on Γ. These two conditions are referred to as the *transmission conditions*.

There are various ways to impose the transmission conditions. Two common methods are by using the Steklov-Poincare operator and by using Lagrange Multipliers. The Steklov-Poincare operator leads to methods such as the Dirichlet-Neumann or Neumann-Neumann Method and Lagrange Multipliers leads to methods such as FETI, mortar element methods, and the Bank-Holst Paradigm DD solver.

3.2.1 Steklov-Poincare Framework

Algorithm 3 Dirichlet - Neumann Method (serial)

Initialize $v^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

2: Find $u_1^k \in H^2(\Omega_1)$ such that

$$Lu_1^k = f_1 \text{ on } \Omega_1$$
$$Bu_1^k = g_1 \text{ on } \partial\Omega_1 \setminus \Gamma$$
$$u_1^k = v_2^{k-1} \text{ on } \Gamma$$

3: Find $u_2^k \in H^2(\Omega_2)$ such that

$$Lu_2^k = f_2 \text{ on } \Omega_2$$
$$Bu_2^k = g_2 \text{ on } \partial\Omega_2 \setminus \Gamma_2$$
$$n_2 \cdot a\nabla u_2^k = n_2 \cdot a\nabla u_1^k \text{ on } \Gamma$$

4: Update: $v_2^k = \theta u_2^k + (1-\theta)v_2^{k-1}$ on Γ.

5: **end for**

The above algorithm is serial. Below is the parallel version.

Algorithm 4 Dirichlet - Neumann Method (parallel)

Initialize $u_1^{(1)}$ and $u_2^{(1)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

2: Update $\mu^{k+\frac{1}{2}} = \theta n_1 \cdot a\nabla u_1^k + (1-\theta) n_1 \cdot a\nabla u_2^k$

$$g^{k+\frac{1}{2}} = \delta u_1^k + (1-\delta) u_2^k$$

Perform steps 3 and 4 simultaneously on different processors.

3: Find $u_1^{k+\frac{1}{2}} \in H^2(\Omega_1)$ such that

$$\begin{aligned} Lu_1^{k+\frac{1}{2}} &= f_1 \text{ on } \Omega_1 \\ Bu_1^{k+\frac{1}{2}} &= g_1 \text{ on } \partial\Omega_1 \setminus \Gamma \\ n_1 \cdot a\nabla u_1^{k+\frac{1}{2}} &= \mu^{k+\frac{1}{2}} \text{ on } \Gamma \end{aligned}$$

4: Find $u_2^{k+\frac{1}{2}} \in H^2(\Omega_2)$ such that

$$\begin{aligned} Lu_2^{k+\frac{1}{2}} &= f_2 \text{ on } \Omega_2 \\ Bu_2^{k+\frac{1}{2}} &= g_2 \text{ on } \partial\Omega_2 \setminus \Gamma_2 \\ u_2^{k+\frac{1}{2}} &= g^{k+\frac{1}{2}} \text{ on } \Gamma \end{aligned}$$

5: Communicate u_i from Γ to neighbor processors.

6: Update $\mu^{k+1} = \beta n_1 \cdot a\nabla u_1^{k+\frac{1}{2}} + (1-\beta) n_1 \cdot a\nabla u_2^{k+\frac{1}{2}}$

$$g^{k+1} = \alpha u_1^{k+\frac{1}{2}} + (1-\alpha) u_2^{k+\frac{1}{2}}$$

Perform steps 7 and 8 simultaneously on different processors.

7: Find $u_1^{k+1} \in H^2(\Omega_1)$ such that

$$\begin{aligned} Lu_1^{k+1} &= f_1 \text{ on } \Omega_1 \\ Bu_1^{k+1} &= g_1 \text{ on } \partial\Omega_1 \setminus \Gamma \\ u_1^{k+1} &= g^{k+1} \text{ on } \Gamma \end{aligned}$$

8: Find $u_2^{k+1} \in H^2(\Omega_2)$ such that

$$\begin{aligned} Lu_2^{k+1} &= f_2 \text{ on } \Omega_2 \\ Bu_2^{k+1} &= g_2 \text{ on } \partial\Omega_2 \setminus \Gamma_2 \\ n_2 \cdot a\nabla u_2^{k+1} &= \mu^{k+1} \text{ on } \Gamma \end{aligned}$$

9: Communicate u_i from Γ to neighbor processors.

10: **end for**

3.2.2 Lagrange Multiplier Framework

When there is an optimization principle associated with our elliptic partial differential equation, we can impose the transmission conditions by formulating a constrained optimization problem.

Assume that our PDE is minimizing some *energy functional*, $J(\cdot)$, then finding a solution to $Lu = 0$ is the same as minimizing $J(u)$ over all u. If we partition our original domain into two disjoint subdomains, then solving $Lu = f$ becomes, find u_1 and u_2 such that

$$J^*(u_1, u_1) = J_1(u_1) + J_2(u_2) = \min_{w_1 \in X_1, w_2 \in X_2} J_1(w_1) + J_2(w_2) \tag{3.3}$$

$$G(u_1, u_2) = u_1|_\Gamma - u_2|_\Gamma = 0 \tag{3.4}$$

The Lagrangian function is

$$\mathcal{L}(u_1, u_2, \lambda) \equiv J_1(u_1) + J_2(u_2) + \lambda G(u_1, u_2) \tag{3.5}$$

We solve this constrained optimization problem by solving the associated saddle point problem

$$Lu_1 - f_1 + \lambda \frac{\partial G}{\partial u_1}(u_1, u_2) = 0 \tag{3.6}$$

$$Lu_2 - f_2 + \lambda \frac{\partial G}{\partial u_2}(u_1, u_2) = 0 \tag{3.7}$$

$$G(u_1, u_2) = 0 \tag{3.8}$$

Applying a Galerkin Approximation and Finite Elements similar to section 2.1.2, our Lagrange Multiplier Method Matrix Form becomes

$$\begin{bmatrix} A_1^{II} & A_1^{IB} & 0 & 0 & 0 \\ A_1^{BI} & A_1^{BB} & 0 & 0 & I \\ 0 & 0 & A_2^{II} & A_2^{IB} & 0 \\ 0 & 0 & A_2^{BI} & A_2^{BB} & -I \\ 0 & I & 0 & -I & 0 \end{bmatrix} \begin{bmatrix} U_1^I \\ U_1^B \\ U_2^I \\ U_2^B \\ \lambda \end{bmatrix} = \begin{bmatrix} F_1^I \\ F_1^B \\ F_2^I \\ F_2^B \\ 0 \end{bmatrix} \tag{3.9}$$

A, U, and F are defined as they were in Section 2.1.5. The additional superscripts

of I and B refer to degrees of freedom on the "interior" and "boundary" respectively. In this two subdomain problem, the "interior" of A_k refers to $\Omega_k \cup (\partial\Omega_k \setminus \Gamma)$ while the "boundary" of A_k refers to Γ.

Two common ways to solve this saddle point problem in parallel are Uzawa's Method and FETI Method. Both are similar. They differ only in how they update the Lagrange Multipliers, λ. Each repeatedly solves local subdomain problems $Lu_k = f_k$ interpreting the Lagrange Multipliers as Neumann boundary data on the interface. After each solve, the Lagrange Multipliers get updated.

Algorithm 5 Uzawa's Method

Initialize $\lambda^{(0)}$ to an initial guess

1: **for** $k = 0, 1, 2, ...$ until convergence **do**

2: Solve simultaneously i=1,2

Find $u_i^k \in H^2(\Omega_i)$ such that

$Lu_i^k = f_k$ on Ω_i

$Bu_i^k = g_i$ on $\partial\Omega_i \setminus \Gamma$

$a\nabla u_i^k \cdot n_\Gamma = \lambda^k$ on Γ

3: Communicate u_i from Γ to neighbor processors.

4: Update Lagrange Multipliers

$\lambda^{k+1} = \lambda^k - \theta(u_1^k - u_2^k)$ on Γ

5: **end for**

Algorithm 6 FETI Method

Initialize $\lambda^{(0)}$ to an initial guess

1: **for** $k = 0, 1, 2, ...$ until convergence **do**

2: Solve simultaneously i=1,2

Find $u_i^k \in H^2(\Omega_i)$ such that

$Lu_i^k = f_k$ on Ω_i

$Bu_i^k = g_i$ on $\partial\Omega_i \setminus \Gamma$

$a\nabla u_i^k \cdot n_\Gamma = \lambda^k$ on Γ

3: Communicate u_i from Γ to neighbor processors.

4: Solve simultaneously i=1,2

Find $w_i^k \in H^2(\Omega_i)$ such that

$Lw_i^k = 0$ on Ω_i

$w_i^k = 0$ on $\partial\Omega_i \setminus \Gamma$

$w_i^k = u_1^k - u_2^k$ on Γ

Communicate ∇w_i from Γ to neighbor processors.

Update Lagrange Multipliers

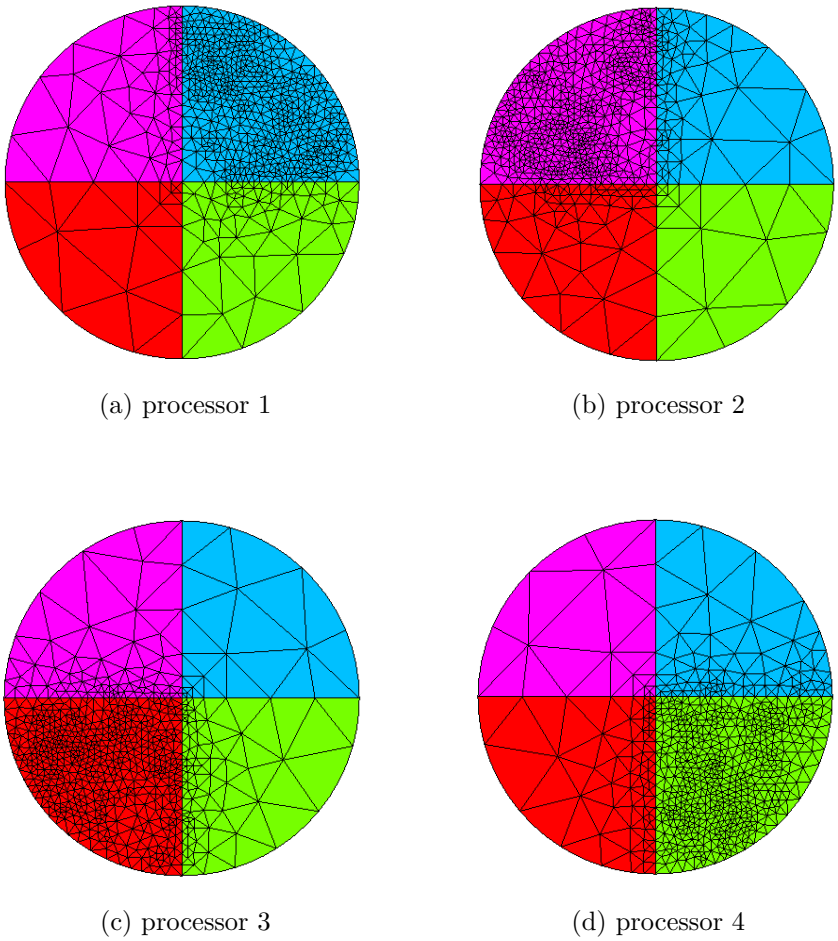
$\lambda^{k+1} = \lambda^k - \theta(a\nabla w_1^k \cdot n_\Gamma + a\nabla w_2^k \cdot n_\Gamma)$

5: **end for**

3.3 Bank-Holst Paradigm DD Solver

The Bank-Holst paradigm DD solver is a unique Domain Decomposition solver that has properties of both overlapping and non-overlapping methods. Every processor maintains a mesh of the entire domain giving it the benefits of overlapping schemes. And, similar to non-overlapping methods, each processor owns a unique disjoint portion of the domain known as its subdomain. When a processor refines its mesh, it places the majority of the new degrees of freedom within the subdomain it has responsibility for. Therefore on a specific processor, the domain has a fine mesh within its subdomain and a coarse mesh outside its subdomain. Figure 3.2 illustrates this for 4 processors. The entire domain is the unit circle. Processor 1's subdomain is quadrant 1 and processor k's subdomain is quadrant

k.



(a) processor 1 (b) processor 2

(c) processor 3 (d) processor 4

Figure 3.2: Local meshes of four processors.

The Bank-Holst paradigm DD solver enforces the transmission conditions between the disjoint subdomains with Lagrange Multipliers as in Section 3.2.2. But because each processor has information about the entire domain, the Bank-Holst paradigm DD solver method doesn't need to explicitly solve for the multipliers like Uzawa's Method (Algorithm 5) or the FETI Method (Algorithm 6).

Each processor creates their own saddle point problem like (3.9) and then

solves only for the unknowns they need.

$$\begin{bmatrix} A_1^{II} & A_1^{IB} & 0 & 0 & 0 \\ A_1^{BI} & A_1^{BB} & 0 & 0 & I \\ 0 & 0 & \bar{A}_2^{II} & \bar{A}_2^{IB} & 0 \\ 0 & 0 & \bar{A}_2^{BI} & \bar{A}_2^{BB} & -I \\ 0 & I & 0 & -I & 0 \end{bmatrix} \begin{bmatrix} \delta U_1^I \\ \delta U_1^B \\ \delta \bar{U}_2^I \\ \delta \bar{U}_2^B \\ \lambda \end{bmatrix} = \begin{bmatrix} R_1^I \\ R_1^B \\ R_2^I \\ R_2^B \\ U_2^B - U_1^B \end{bmatrix} \tag{3.10}$$

Equation (3.10) is the saddle point problem that processor 1 forms. In place of $A_2^{II}, A_2^{IB}, A_2^{BI}$, and A_2^{BB} it substitutes its own stiffness matrices created from the coarse mesh that is has located in processor 2's subdomain and denoted above as $\bar{A}_2^{II}, \bar{A}_2^{IB}, \bar{A}_2^{BI}$, and $\bar{A}_2^{BB}$. Processor 1 receives R_2^B and U_2^B from processor 2 and it sets $R_2^I = 0$. Processor 2 creates a similar saddle point problem substituting its coarse stiffness matrices for A_1's and receives R_1^B and U_1^B from processor 1.

Since processor 1 doesn't need the quantities $\delta \bar{U}_2^B$ and λ, it reorders (3.10) and eliminates these sets of equations by block elimination

$$\left[\begin{array}{cc|ccc} 0 & -I & 0 & I & 0 \\ -I & \bar{A}_2^{BB} & 0 & 0 & \bar{A}_2^{BI} \\ \hline 0 & 0 & A_1^{II} & A_1^{IB} & 0 \\ I & 0 & A_1^{BI} & A_1^{BB} & 0 \\ 0 & \bar{A}_2^{IB} & 0 & 0 & \bar{A}_2^{II} \end{array}\right] \left[\begin{array}{c} \lambda \\ \delta \bar{U}_2^B \\ \hline \delta U_1^I \\ \delta U_1^B \\ \delta \bar{U}_2^I \end{array}\right] = \left[\begin{array}{c} U_2^B - U_1^B \\ R_2^B \\ \hline R_1^I \\ R_1^B \\ R_2^I \end{array}\right] \tag{3.11}$$

The 3×3 Schur complement system is

$$\begin{bmatrix} A_1^{II} & A_1^{IB} & 0 \\ A_1^{BI} & A_1^{BB} + \bar{A}_2^{BB} & \bar{A}_2^{BI} \\ 0 & \bar{A}_2^{IB} & \bar{A}_2^{II} \end{bmatrix} \begin{bmatrix} \delta U_1^I \\ \delta U_1^B \\ \delta \bar{U}_2^I \end{bmatrix} = \begin{bmatrix} R_1^I \\ R_1^B + R_2^B + \bar{A}_2^{BB}(U_2^B - U_1^B) \\ \bar{A}_2^{IB}(U_2^B - U_1^B) \end{bmatrix} \tag{3.12}$$

Algorithm 7 Bank-Holst paradigm DD solver

Initialize U_1 and U_2 to an initial guess

1: **for** $k = 0, 1, 2, ...$ until convergence **do**

2: Compute residuals R_1 and R_2

3: Communicate R_i^B and U_i^B from Γ to neighbor processors

4: Simultaneously for i=1,2

Solve (3.12) $A_i^* \delta U_i = R_i^*$ for δU_i.

5: **end for**

This algorithm is described in more detail in [10] [2] [38] [15]. It is motivated by and similar to domain decomposition algorithms described in [8] [7]. The Bank-Holst paradigm is described in [5] [6] with an additional contribution described in [13].

3.4 Multiple Subdomains

For simplicity of discussion, all the methods and expositions in this Chapter have dealt with only two subdomains. Of course in practice, scientists use many subdomains. All of these schemes extend naturally to many subdomains.

Each algorithm follows roughly the same pattern. First they solve local problems in parallel, then communicate boundary information, and last, some methods update variables. When you use these methods on partitions with more than two subdomains, you follow the same pattern. Each processor solves their local problem, then they communicate boundary information to the appropriate neighbor processor, and lastly they optionally update variables.

Chapter 4

Partitioning Algorithms

In order to employ a Domain Decomposition Method, the original domain must be partitioned into subdomains, non-overlapping or overlapping. In both cases, the domain can first be partitioned into non-overlapping subdomains. Then in the latter case, the non-overlapping subdomains can be extended to overlap their neighbors.

The problem of decomposing a domain can be formulated as a Graph Partitioning problem after discretizing the original domain into Finite Elements as in Section 2.1.2. Associate each element (triangle) with a graph's vertex and each side shared by elements as a graph's edge. Then you can apply standard Graph Partitioning algorithms with additional constraints you choose. [20] [39]

4.1 Definitions

Definition 4.1.1. A *graph* $G = (V, E)$ consists of a collection V of n vertices, $V = \{v_1, ..., v_n\}$ together with a collection E of m edges, $E = \{e_1, ..., e_m\}$ where each edge represents adjacency between pairs of vertices. If e_k represents the connectivity of v_i and v_j, we say $e_k = (v_i, v_j) = (v_j, v_i) \in E$.

Associated with each graph are some useful matrices.

Definition 4.1.2. Given a graph G, its connectivity can be represented with a

symmetric $n \times n$ *Adjacency Matrix*, M_G, defined as

$$[M_G]_{i,j} = \begin{cases} 1, \text{ if } (v_i, v_j) \in E \\ 0, \text{ if } (v_i, v_j) \notin E \end{cases} \tag{4.1}$$

Definition 4.1.3. A weighted graph is a graph $G = (V, E)$ with weights $w_{i,j}$ assigned to each edge $(v_i, v_j) \in E$ and weights assigned to each vertex $v_k \in V$ denoted $w(v_k)$. All the edge weights then form a symmetric $n \times n$ *Weight Matrix*, W_G.

Remark 4.1.4. By default, an unweighted graph can be converted into a weighted graph by assigning weight equal 1 to all of its edges and vertices; Let $[W_G]_{i,j} = [M_G]_{i,j}$ $\forall i, j$ and $w(v_k) = 1$ $\forall k$.

Definition 4.1.5. Given a graph G, we can define a *Laplacian Matrix*, L_G as follows:

$$[L_G]_{i,j} = \begin{cases} \sum_{k \neq i} w_{i,k}, & \text{if } j = i \\ -w_{i,j}, & \text{if } (v_i, v_j) \in E \\ 0, & \text{if } (v_i, v_k) \notin E \text{ and } i \neq j \end{cases} \tag{4.2}$$

4.2 Graph Partitioning Algorithms

After representing our domain as a graph, we partition it into parts. Each processor will get assigned its own subdomain and the degrees of freedom therein.

Definition 4.2.1. Given a graph $G = (V, E)$ and a parameter $\epsilon > 0$, define K_ϵ as a partition of G into p parts, $V_1, ..., V_p$ of size $n_1, ..., n_p$ respectively if

$$(1 - \epsilon)\frac{n}{p} \leq n_i \leq \frac{n}{p}(1 + \epsilon) \text{ for } i = 1, ..., p \tag{4.3}$$

Definition 4.2.2. Given a graph $G = (V, E)$ with weight matrix W_G and two disjoint vertex subsets V_i and V_j of V, we define

$$\delta(V_i, V_j) \equiv \sum_{\{v_r \in V_i, v_s \in V_j\}} w_{rs} \tag{4.4}$$

$\delta(V_i, V_j)$ is the sum of the weights of the edges connecting parts V_i and V_j and is referred to as *edge cut*. For three or more vertex subsets, we define

$$\delta(V_1, ..., V_p) \equiv \sum_{i=1}^{p-1} \sum_{j=i+1}^{p} \delta(V_i, v_j) \tag{4.5}$$

From the Algorithms in Chapter 3, we saw that iterative Domain Decomposition Methods must communicate between each local solve and the communication volume is proportional to the combined length of the interfaces between processors. Therefore, when partitioning the degrees of freedom in a discretized PDE, the partition should strive to distribute the unknowns equally among the parts while minimizing the length of the interfaces between parts. This can be formally written as

$$\begin{gathered}\text{Find a partition } K_\epsilon \text{ such that} \\ \delta(V_1, ..., V_p) = \min_{\bar{V}_1,...,\bar{V}_p} \delta(\bar{V}_1, ..., \bar{V}_p)\end{gathered} \tag{4.6}$$

Remark 4.2.3. This is an NP hard discrete problem in which no algorithm of polynomial complexity is known to exist, see [40]. We therefore solve it with *heuristic* algorithms and approximate the solution. In the next three subsections, we show three of these algorithms.

The terminology of Sections 4.1 and 4.2 is illustrated in Example 4.5.1.

4.2.1 Kernighan-Lin Algorithm

The Kernighan-Lin Algorithm as detailed in [36], is a discrete descent method to solve (4.6). Start with an initial partition $V_1, ..., V_p \equiv K_\epsilon$ and repeatedly consider swapping a vertex from one part with a vertex from another part. If the new partition is still in K_ϵ and the switch reduces edge cut $\delta(V_1, ..., V_p)$ then make the swap. To avoid being stuck in a local minimum, the Kernighan-Lin Algorithm allows a fixed number of exchanges that increase edge cut. Additionally, this method should be run on a set of initial partitions and the best final partition would be selected.

Definition 4.2.4. Define the *gain* of exchanging $v_r \in V_i$ with $v_s \in V_j$ as

$$\text{gain}(v_r, v_s) = \begin{cases} d_{V_i}(v_r) - d_{V_j}(v_r) + d_{V_j}(v_s) - d_{V_i}(v_s) & \text{if } (v_r, v_s) \notin E \\ d_{V_i}(v_r) - d_{V_j}(v_r) + d_{V_j}(v_s) - d_{V_i}(v_s) - 2w_{rs} & \text{if } (v_r, v_s) \in E \end{cases}$$

where

$$d_{V_k}(v_r) \equiv \sum_{\{(v_r,v_s)\in E, v_s \in V_k\}} w_{rs}$$

Using the gain function, choose pairs of vertices and swap them if the gain is negative.

4.2.2 Recursive Spectral Bisection Algorithm

Recursive Spectral Bisection Algorithm is a very popular graph partitioning algorithm originally described here [23] [24] and written about many times since. It works by repeatedly bisecting a graph into two subgraphs using eigenvalue theory and produces partitions of high quality as measured by low edge cut.

Definition 4.2.5. A Laplacian Matrix's *Fiedler* vector is an eigenvector of L_G corresponding to the smallest non zero eigenvalue $\lambda_2 > 0$.

To Spectrally Bisect graph $G = (V, E)$ with associated Laplacian Matrix L_G defined in (4.2), compute the Fiedler vector x_2 by solving the eigenvalue problem

$$L_G x_2 = \lambda_2 x_2 \tag{4.7}$$

Then sort the entries of x_2 from least to greatest and calculate the median of these entries denoted m. Define $q_i = 1$ if $[x_2]_i \geq m$ and $q_i = -1$ if $[x_2]_i < m$. Vector q defines the two subgraphs $V_1 = \{v_i | q_i = 1\}$ and $V_2 = \{v_i | q_i = -1\}$. To partition weighted vertices or to partition a graph into $p \neq 2^k$ parts, m can be adjusted to take care of this. Regardless of your choice of m, if G is connected then V_1 and V_2 will be connected [23].

Recursive Spectral Bisection is the repeated application of Spectral Bisection. To partition a graph into $p = 2^k$ parts, repeatedly bisect a graph and the subsequent subgraphs until you have the desired number of parts. To partition G into $p \neq 2^k$, then instead of bisecting a graph into two equal pieces, adjust m and bisect the graph into pieces of ratio 2^k to $p - 2^k$ for a k that makes $2^k \approx p/2$. For example, to partition into 15 parts, we first bisect the graph into two parts with ratio $8 : 7$. The part with 8 area is further broken down into 8 parts. The part with 7 area is bisected into two parts with ratio $4 : 3$ etc.

Why does spectral bisection work? By construction, the edge cut of the partition defined by q is

$$\delta(V_1, V_2) = \frac{1}{4} q^T L_G q \tag{4.8}$$

If G is connected then L_G is s.p.d with n orthonormal eigenvectors $x_1, ..., x_n$ and corresponding eigenvalues $0 = \lambda_1 < \lambda_2 \leq ... \leq \lambda_n$. We know $x_1 = \frac{1}{\sqrt{n}}[1, 1, ..., 1, 1]^T$ and $q^T x_1 = 0$. Thus q can be written as $q = ||q|| \sum_{i=2}^{n} \alpha_i x_i$ with $\sum_{i=2}^{n} \alpha_i^2 = 1$.

$$\begin{aligned} q^T L_G q &= n \left(\sum_{i=2}^{n} \alpha_i x_i \right)^T \sum_{i=2}^{n} \lambda_i \alpha_i x_i \\ &= n \sum_{i=2}^{n} \lambda_i \alpha_i^2 \end{aligned} \tag{4.9}$$

therefore

$$0 < n\lambda_2 \leq q^T L_G q \leq n\lambda_n \tag{4.10}$$

From (4.9) and (4.10) we see that $q^T L_G q$ is minimized when the discrete vector q approximates the direction of the continuous Fiedler vector.

4.2.3 Multilevel Graph Partitioning Algorithm

The Multilevel Graph Partitioning Algorithm is motivated by multigrid methodology [22] [28] and uses graph compaction algorithms. Instead of partitioning graph G, we construct a hierarchy of coarser (smaller) graphs and then partition the coarsest (smallest) graph. Then we project the partition back through the finer (larger) graphs and unto G. [42] [19]

Denote the original largest graph by $G^{(0)} = (V^{(0)}, E^{(0)})$ and the sequence of coarser (smaller) graphs as $G^{(k)} = (V^{(k)}, E^{(k)})$ for $k = 1, ..., n$ with Weight Matrices $W^{(k)}$. Given a graph, this algorithm defines a coarser graph by pairing vertices. Note that a pair of vertices can be referenced by the edge that connects the vertices.

Definition 4.2.6. Given a graph $G = (V, E)$, a *matching* is any subset of E such that no two edges contained therein are incident to the same vertex. (This creates unique pairs of vertices where each vertex belongs to at most one pair.). A *maximal matching* is a matching in which no additional edge can be added without violating the matching condition.

A maximal matching can be created by randomly choosing a vertex and then matching it with an adjacent vertex that is unmatched. If there are no adjacent

unmatched vertices, then match the vertex with itself. Continue randomly choosing vertices until all vertices are matched. Figure 4.1 shows an example. The original graph had all vertex and edge weights equal 1.

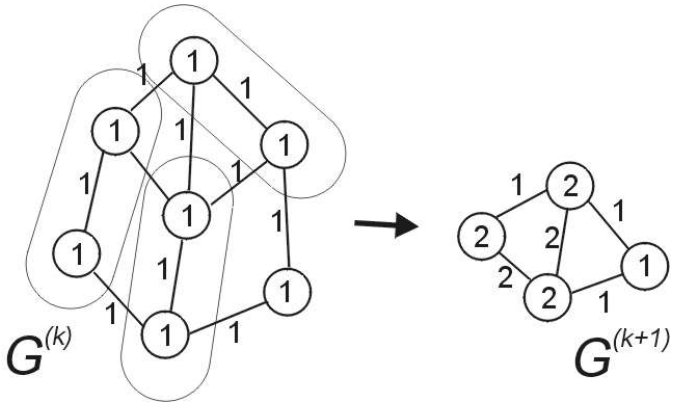


Figure 4.1: A maximal matching being used to coarsen a graph and create new edge and vertex weights.

A matching on $G^{(k)}$ defines the next coarse graph $G^{(k+1)}$. All vertex pairs are merged to single vertices. This creates $V^{(k+1)}$ from $V^{(k)}$. If $v_i^{(k+1)} \in V^{(k+1)}$ belongs to graph $G^{(k+1)}$, denote its parents as $v^{(k)}_{P_1(i,k+1)}$ and $v^{(k)}_{P_2(i,k+1)} \in V^{(k)}$. A vertex $v_i^{(k+1)}$ will be defined as adjacent to $v_j^{(k+1)}$ if any parent vertices of $v_i^{(k+1)}$ are adjacent to any parent vertices of $v_j^{(k+1)}$. This creates $E^{(k+1)}$. The Weight Matrix $W^{(k+1)}$ is defined as follows

$$w^{(k+1)}(v_r^{(k+1)}) = w^{(k)}\left(v^{(k)}_{P_1(r,k+1)}\right) + w^{(k)}\left(v^{(k)}_{P_2(r,k+1)}\right) \tag{4.11}$$

$$w_{i,j}^{(k+1)} = \sum_{r\in\{P_1(i,k+1),P_2(i,k+1)\}} \sum_{s\in\{P_1(j,k+1),P_2(j,k+1)\}} w_{r,s}^{(k)} \tag{4.12}$$

If $v_i^{(k+1)}$ is a singleton and not part of a pair, then both its parents refer to the same vertex and

$$w^{(k+1)}(v_r^{(k+1)}) = w^{(k)}\left(v^{(k)}_{P_1(r,k+1)}\right) \tag{4.13}$$

See Figure 4.1 for an example.

Algorithm 8 Multilevel Graph Partitioning Algorithm

1: **for** $k = 1, ..., n$ **do**

2: Coarsen mesh using maximal matching

$$V^{(k)} \leftarrow V^{(k-1)}$$

$$E^{(k)} \leftarrow E^{(k-1)}$$

3: Compute vertex and edge weights

$$w^{(k+1)}(v_r^{(k+1)}) = w^{(k)}\left(v_{P_1(r,k+1)}^{(k)}\right) + w^{(k)}\left(v_{P_2(r,k+1)}^{(k)}\right)$$

$$w_{i,j}^{(k+1)} = \sum_{r \in \{P_1(i,k+1), P_2(i,k+1)\}} \sum_{s \in \{P_1(j,k+1), P_2(j,k+1)\}} w_{r,s}^{(k)}$$

4: **end for**

5: Partition mesh: $V^{(n)} \rightarrow \{V_1^{(n)}, ..., V_p^{(n)}\}$

6: **for** $k = n, ..., 1$ **do**

7: Uncoarsen mesh: $V_i^{(k-1)} \leftarrow V_i^{(k)}$ for $i = 1, ..., p$.

8: Improve the partition using Kernighan-Lin and $\delta^{(k-1)}$.

9: **end for**

Various software implementations of multilevel partitioners are available. Two popular ones are CHACO [29] and METIS [34]. PLTMG [18] also uses multilevel graph partitioning.

4.3 METIS

METIS is a software package dedicated to graph partitioning that implements Multilevel Graph Partitioning as described in Section 4.2.3. METIS has two main partitioning options. METIS uses either multilevel recursive bisection or multilevel k-way partitioning. The three phases of multilevel graph partitioning are graph coarsening, initial partitioning, and uncoarsening. In multilevel k-way partitioning, each phase is called once. In multilevel recursive bisection, each phase is called approximately $\log_2 p$ times where p is the number of parts desired.

During graph coarsening, METIS offers two choices. The user can use random matching as described in Section 4.2.3 or heavy edge matching. A matching is a subset of edges. When a graph is being coarsened, any edge selected to be in a matching subset cannot get cut during the initial partitioning phase. There-

fore, partitions can be improved by placing edges that will most likely not be cut in a matching subset. Since graph partitioning algorithms attempt to minimize edge cut, they try not to cut edges with large weights. Heavy edge matching improves partitioning by places edges with large weights in a matching subset. When choosing vertex pairs, if v_r is one vertex, choose the partner vertex v_s such that $\max_{(v_r,v_s)\in E} w_{r,s}$ is obtained.

During the initial partitioning phase, METIS offers four choices. The main choices are the Greedy Growing strategy and the Kernighan-Lin strategy (described in section 4.2.1). Older versions of METIS offered the Spectral Bisection strategy also (described in section 4.2.2). Keep in mind that the graph size being partitioning is around 100 vertices when doing a bisection and $100p$ when doing k-way therefore the partitioning algorithm doesn't need to be too elaborate. The final partition is strongly influenced by the coarsening and uncoarsening phases.

A Growing strategy starts with one vertex and adds adjacent vertices until half of the vertex weight is consumed. When a Greedy Growing strategy chooses which neighboring vertex to add, it chooses the one that minimizes edge cut over the other choices. A Growing strategy just takes one at random.

During the uncoarsening phase, METIS uses a modified form of the Kernighan-Lin strategy (described in section 4.2.1). Instead of considering all pairs of vertex exchanges, METIS employs Boundary Kernighan-Lin which only considers exchanging vertices that lie on the interface between parts. This saves time since the gain no longer needs to be calculated n^2 times if n is the number of vertices.

The algorithms used by METIS are explained in detail in [35] [33] [32].

4.4 PLTMG

PLTMG is a software package for solving elliptic partial differential equations [18] that includes Open MPI and can solve problems in parallel. As such it needs to partition domains into subdomains. It accomplishes this with multilevel graph partitioning similar to Algorithm 8. PLTMG deviates from Algorithm 8 by coarsening and uncoarsening in one step each.

During the coarsening phase, instead of using maximal matching repeatedly and coarsening n times until it reaches a small graph, it uses a growing strategy one time to clump groups of vertices together where each clump has a sum of vertex weights approximately equal to $w_{\text{total}}/(100P)$ where $w_{\text{total}} = \sum_{i=1}^{m} w(v_i)$ and P is the number of parts desired.

Next, it partitions this smaller graph with recursive spectral bisection. Lastly, it projects the coarse graph unto the original fine graph and then applies the Kernighan-Lin Algorithm focusing only on the boundaries between parts.

4.5 Finite Element Partitioning Example

Example 4.5.1. Partition the Finite Element mesh in Figure 4.2 below into two subdomains. Try to minimize edge cut and balance the distribution of unknowns as in (4.6).

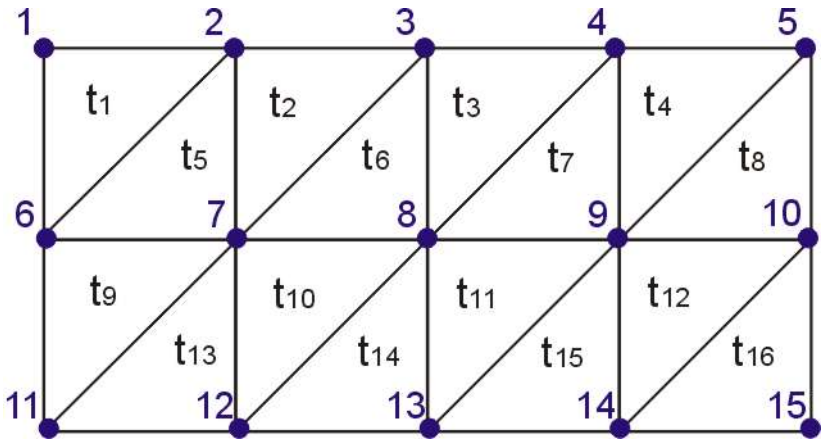


Figure 4.2: Finite Element mesh

The mesh above uses triangle Finite Elements and piecewise linear Lagrange basis functions with degrees of freedom located at the triangle vertices. There are 15 degrees of freedom labeled blue and 16 triangles labeled black.

First we must represent this mesh as a graph. We can either associate the triangles with graph vertices or the degrees of freedom with graph vertices. It seems like we would prefer to associate the degrees of freedom since we wish

to divide them. However, if we do that, we may be forced to break up the Finite Element triangles. Therefore, we will associate triangles with graph vertices. Then whichever subdomain gets assigned a certain triangle will receive responsibility for that triangle and its associated degrees of freedom.

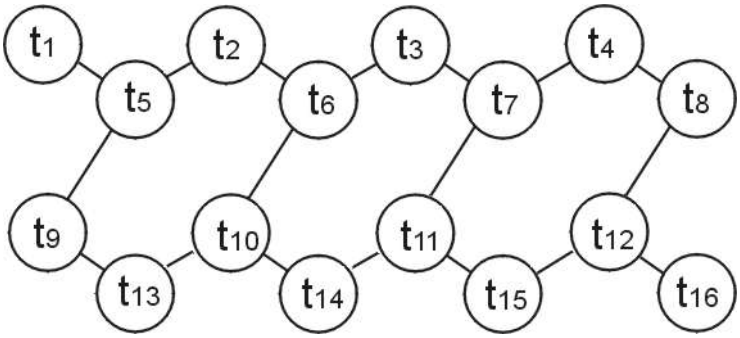


Figure 4.3: Finite Element graph

Figure 4.3 shows the associated graph with vertex $v_k = t_k$ for $k = 1, 2, ..., 16$. Edges are represented by a line connecting vertices. The Adjacency Matrix is

$$M_G = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.14}$$

and the Lapacian Matrix is

$$L_G = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 3 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 3 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 3 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.15}$$

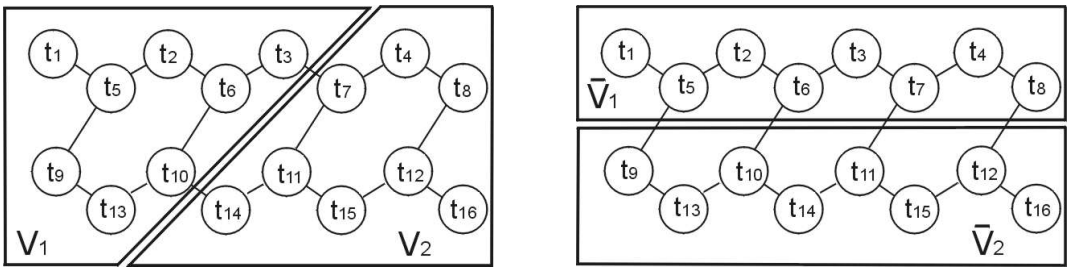
We can use the Spectral Bisection Method to partition this graph into two parts which will heuristically solve (4.6). Solving the eigenvalue problem

$$L_G x = \lambda x \tag{4.16}$$

we find that the least non-zero eigenvalue $\lambda_2 = 0.148$ and the associated eigenvector $x_2 = [0.387, 0.250, 0.003, -0.232, 0.329, 0.133, -0.127, -0.303, 0.303, 0.127, -0.133, -0.329, 0.232, -0.003, -0.250, -0.387]^T$. Therefore the partitions of V are $V_1 = \{t_1, t_2, t_3, t_5, t_6, t_9, t_{10}, t_{13}\}$ and $V_2 = \{t_4, t_7, t_8, t_{11}, t_{12}, t_{14}, t_{15}, t_{16}\}$.

This partition has characteristics $\delta(V_1, V_2) = 2$ and $|V_1| = |V_2| = \frac{n}{p} = 8$. By inspection of the original graph, it can be seen that $\min_{\bar{V_1},...,\bar{V_p} \equiv K_0} \delta(\bar{V_1}, ..., \bar{V_p}) = 2$. Spectral Bisection succeeded in balancing the distribution of unknowns and minimizing edge cut.

Our partition is pictured in Figure 4.4a. Figure 4.4b illustrates a different partition of G that balances the distribution of unknowns but doesn't minimize edge cut.



(a) $\delta(V_1, V_2) = 2$ (b) $\delta(\bar{V_1}, \bar{V_2}) = 4$

Figure 4.4: Different partitions of graph G.

Chapter 5

Edge Weighting Schemes

In the preceding chapter, we worked through a simple example of partitioning a Finite Element mesh using Graph Partitioning algorithms. In the Example 4.5.1, the Weight Matrix W_G associated with our graph G had all its edges' and vertices' weights equal to 1, $W_G = M_G$ and $w(v_k) = 1$, as explained in Remark 4.1.4. In this chapter, we will consider other Weight Matrices.

Solving (4.6) with all the weights equal to 1 both balances the unknowns per processor and minimizes communication between processors. In parallel algorithms, each iteration is a combination of both computation and communication. Recognizing that communication is proportional to the combined interface length (or more accurately discrete edge cut) between the parts of your partition motivates scientists to minimize this length.

Definition 5.0.2. Given a domain Ω with area (volume) A and a parameter $\epsilon > 0$, define $\bar{K}_\epsilon$ as a partition of Ω into P parts, $\Omega_1, ..., \Omega_P$ of areas (volumes) $A_1, ..., A_P$ respectively if

$$(1-\epsilon)\frac{A}{P} \leq A_i \leq \frac{A}{P}(1+\epsilon) \text{ for } i = 1, ..., P \tag{5.1}$$

Definition 5.0.3. Given a domain Ω and partition $\bar{K}_\epsilon$ of P parts labeled $\{\Omega_1, ..., \Omega_p\}$, define the *interface length* as the sum length of all the boundaries shared by 2 or more partitions. Denote this *interface length* as $\bar{\delta}(\Omega_1, ..., \Omega_P)$ or $\bar{\delta}(\bar{K}_\epsilon)$. Formally, $\bar{\delta}(\bar{K}_\epsilon) = \int_\Gamma ds$ where Γ is the interface.

Lemma 5.0.4. *$\bar{\delta}(\Omega_1, ..., \Omega_P) = \frac{1}{2}\sum_{k=1}^{P}\int_{\partial\Omega_k} ds - \frac{1}{2}\int_{\partial\Omega} ds$ where $\partial\Omega_k$ and $\partial\Omega$ are the boundaries of Ω_k and Ω respectively.*

Proof. $\sum_{k=1}^{P}\int_{\partial\Omega_k} ds - \int_{\partial\Omega} ds$ is the combined length of all the parts' boundaries on the interior of Ω counted twice. □

Theorem 5.0.5. *Given a domain $\Omega \in \mathbb{R}^2$ with area A and boundary (perimeter) length S, then all partitions $\bar{K}_0$ of size P have interface length greater than or equal to $\sqrt{\pi AP} - 0.5S$.*

$$\text{For } \Omega \in \mathbb{R}^2 \qquad \min_{\Omega_1,...,\Omega_P \equiv \bar{K}_0} \bar{\delta}(\Omega_1, ..., \Omega_P) \geq \sqrt{\pi AP} - 0.5S \tag{5.2}$$

Proof. Since $\epsilon = 0$, each part Ω_k has area $\frac{A}{P}$. The shape in $\mathbb{R}^2$ which minimizes the ratio of perimeter to area is a circle. Assume every part is shaped optimally as a circle. Then the radius of each part is $r = \sqrt{\frac{A}{\pi P}}$ and has perimeter $2\pi r$.

The total of all the part's perimeters is $t = 2\pi P\sqrt{\frac{A}{\pi P}}$. The portion of these perimeters in the interior of the domain are $t - S$ and the interface length $= \frac{t-S}{2}$. □

Theorem 5.0.6. *Given a domain $\Omega \in \mathbb{R}^3$ with volume A and boundary surface area S, then all partitions $\bar{K}_0$ of size P have interface length (interface surface area) greater than or equal to $\sqrt[3]{4.5\pi A^2 P} - 0.5S$.*

$$\text{For } \Omega \in \mathbb{R}^3 \qquad \min_{\Omega_1,...,\Omega_P \equiv \bar{K}_0} \bar{\delta}(\Omega_1, ..., \Omega_P) \geq \sqrt[3]{4.5\pi A^2 P} - 0.5S \tag{5.3}$$

Proof. Since $\epsilon = 0$, the volume of each part equals $\frac{A}{P}$. Assume every part is shaped optimally as a sphere. Then the radius of each part is $r = \sqrt[3]{\frac{3A}{4\pi P}}$ and has surface area $4\pi r^2$. Apply Lemma 5.0.4. □

Interface length, $\bar{\delta}(\Omega_1, ..., \Omega_P)$, is the continuous analogy of edge cut, $\delta(V_1, ..., V_P)$ which was presented in Definition 4.2.2. The relationship is explained by Lemma 5.0.9.

Definition 5.0.7. Let Ω be a domain with a triangularization T_h of m triangles $t_1, ..., t_m$. Let the graph $G = (V, E)$ be a representation of T_h, formed by associating each vertex $v_k \in V$ with a triangle $t_k \in T_h$. Let the edges of G represent the adjacency of the triangles. If triangle t_i is adjacent to triangle t_j, then $(v_i, v_j) \in E$. Let K_ϵ be a partition of G into P parts $V_1, ..., V_P$ and let $\bar{K}_\epsilon$ be a partition of Ω into P parts $\Omega_1, ..., \Omega_P$. We say that $\bar{K}_\epsilon$ *coincides* with K_ϵ if each $\Omega_k = \cup_{i=1}^{n_k} t_i$ for each $t_i \equiv v_i \in V_k$. We say that $\bar{K}_\epsilon$ *coincides* with T_h if each $\Omega_k = \cup_{i=1}^{n} t_{k_i}$ for some collection of triangles $\{t_{k_1}, ..., t_{k_n}\} \in T_h$.

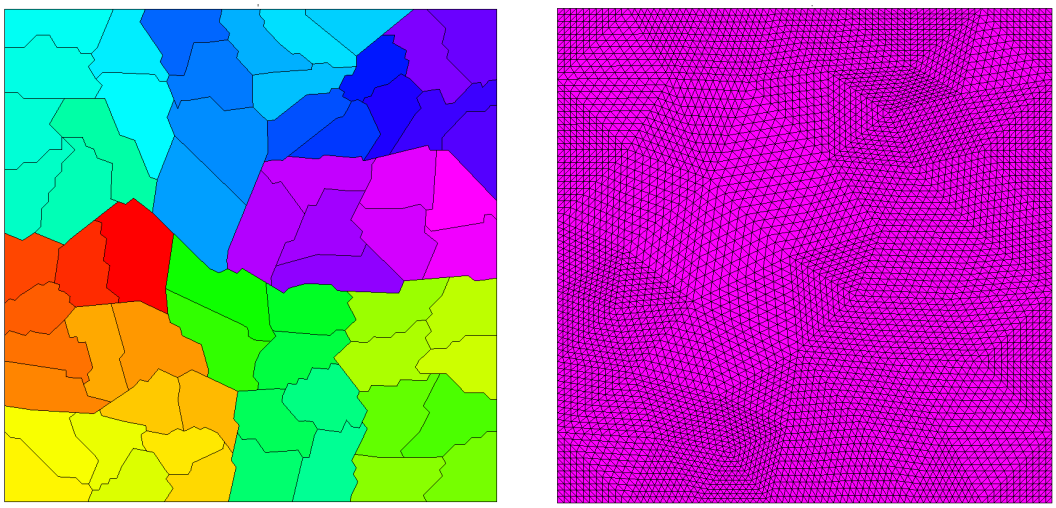
Definition 5.0.8. If T_h is a triangularization of a domain Ω, define a mesh's *element density function* as $D_E(x, y) : \Omega \to \mathbb{R}^+ = 1/h_{x,y}$ where $h_{x,y}$ is the length of the closest edge to $(x, y) \in \Omega$. If two of more edges are equidistant from (x, y), take the average, $D_E(x, y) = m/(\sum_{i=1}^{m} h_{x,y,i})$.

Lemma 5.0.9. *Given a domain Ω with triangularization T_h, mesh element density function $D_E(x, y)$, a representative graph G with edges weights equal 1, and partitions K_ϵ and $\bar{K}_\epsilon$ of size P that coincide, then $\delta(K_\epsilon) = \int_\Gamma D_E ds$ where Γ is the interface of $\bar{K}_\epsilon$. If T_h is a uniform mesh, then $D_E(x, y)$ is a constant function and $\delta(K_\epsilon) = D_E \bar{\delta}(\bar{K}_\epsilon)$.*

When partitioning a non-pathological domain into finite equal area (volume) parts, one can never achieve an interface length of $\sqrt{\pi AP} - 0.5S$ in $\mathbb{R}^2$ nor $\sqrt[3]{4.5\pi A^2 P} - 0.5S$ in $\mathbb{R}^3$ because a domain cannot be divided into disjoint circles or spheres, but it is helpful to have a lower bound on the NP hard continuous problem which is analogous to (4.6), the NP hard discrete problem we wish to solve.

We will continue the discussion in $\mathbb{R}^2$ but all of the results in this chapter can be derived in $\mathbb{R}^3$ also. The minimum edge cut achievable in (4.6) depends on the shape of the domain and the number of parts desired. Although circles can never tile your domain and achieve an edge cut of $D_E(\sqrt{\pi}\sqrt{AP} - 0.5S)$, it is sometimes possible to use regular hexagons, squares, and equilateral triangles to achieve edge cuts of $D_E(\sqrt{2\sqrt{3}}\sqrt{AP} - 0.5S)$, $D_E(2\sqrt{AP} - 0.5S)$, and $D_E(3^{0.75}\sqrt{AP} - 0.5S)$ respectively. (The proofs for these values follow the form of the proof for Theorem 5.0.5.)

Remark 5.0.10. When $\sqrt{AP} \gg 0.5S$, the ratio between the interface lengths from using theoretical circles versus achievable regular hexagons, squares, and equilateral triangles is $\sqrt{\pi} : \sqrt{2\sqrt{3}} : 2 : 3^{0.75} \approx 1.77 : 1.86 : 2.00 : 2.28 = 1.00 : 1.05 : 1.13 : 1.29$. Therefore the lower bound on the minimum interface length is close to achievable.



(a) METIS partition (b) PLTMG mesh

Figure 5.1: Unit square partitioned into 64 parts by METIS from PLTMG's mesh of 40000 triangles with vertex and edge weights equal 1.

Figure 5.1 shows METIS partitioning the unit square in $\mathbb{R}^2$ into 64 parts by solving (4.6) on a graph representing PLTMG's triangularization of 40000 triangles. Notice how the parts are shaped similar to regular hexagons or squares. As mentioned in Remark 4.2.3, algorithms will minimize the interface length heuristically not perfectly. The interface length in this partition equals 16.4 and the edge cut equals 2087. When PLTMG partitions the unit square, it achieves an interface length of 15.3 and edge cut of 1923 and the partition looks similar to Figure 5.1a. The lower bound on the minimum interface length is $\sqrt{\pi}\sqrt{AP} - 0.5S \approx 12.2$. With 40000 triangles, if D_E were constant, then $D_E(x, y) = \sqrt{10^4\sqrt{3}} \approx 131.6$ and therefore the lower bound on the minimum edge cut is 1606. We don't know what the optimal achievable minimum interface length nor edge cut is, but if you partition the unit square uniformly into 64 squares as in Figure 5.2, you would achieve 14.0

and 1842.

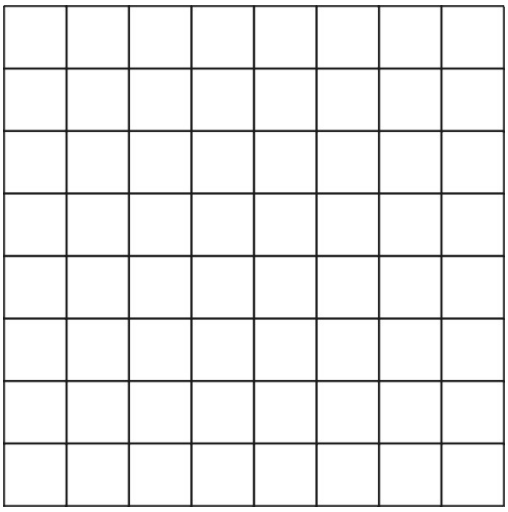


Figure 5.2: Unit square partitioned in 64 uniform square parts.

Since the communication in iterative Domain Decomposition Methods is proportional to the interface length, increasing the interface length will increase the time needed to complete each iteration. The significance of this increase will depend on the ratio between computation time and communication time.

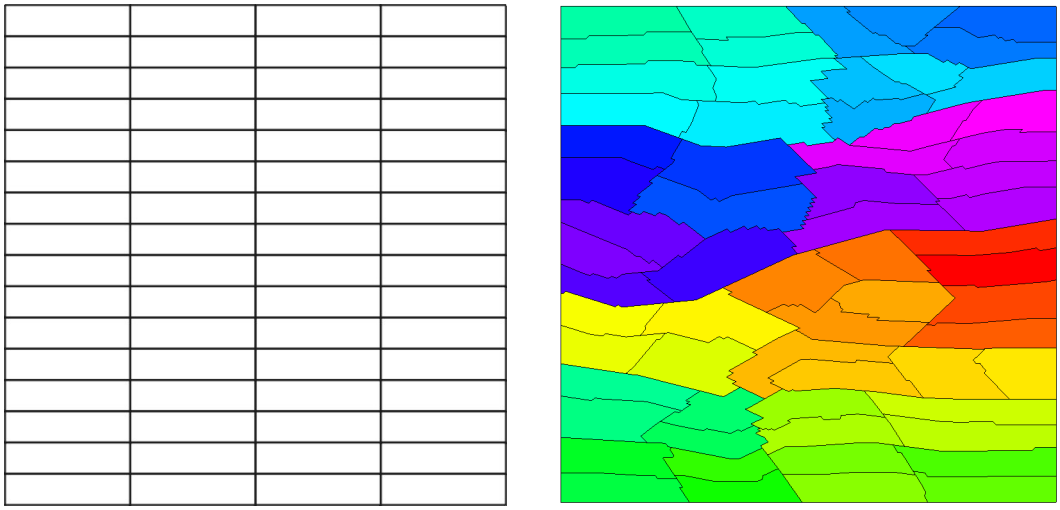
Lemma 5.0.11. *Given that the ratio of computation time, t^{comp}, to communication time, t^{comm}, during one iteration of an iterative parallel method is $\alpha = \frac{t^{comp}}{t^{comm}}$ and assuming that communication time is proportional to the interface length and assuming that computation time is independent of interface length, then increasing the interface length by a factor of $\beta > 1$ will increase the time for one iteration by a factor of $\frac{\alpha+\beta}{\alpha+1}$.*

Lemma 5.0.11 demonstrates that if $\alpha = 10, 100$, or 1000, then doubling the interface length ($\beta = 2$) will increase the time per iteration by a factor of approximately $1.1, 1.01$ and 1.001 respectively. In general if $\alpha = 10^k$, doubling the interface length will increase the time per iteration by 10^{2-k} percent.

Choosing a partition that doesn't minimize the interface length but does reduce the total number of required iterations, can achieve an overall faster execution time even though the time for each iteration is longer.

Lemma 5.0.12. *Assume that the ratio of computation time, t^{comp}, to communication time, t^{comm}, during one iteration of an iterative parallel method is $\alpha = \frac{t^{comp}}{t^{comm}}$. Assume that communication time is proportional to the interface length and that computation time is independent of interface length. Consider two partitions, $\bar{K}_\epsilon^{(1)}$ and $\bar{K}_\epsilon^{(2)}$ where the ratio of their interface lengths is $\beta > 1$, $\bar{\delta}(\bar{K}_\epsilon^{(1)})/\bar{\delta}(\bar{K}_\epsilon^{(2)}) = \beta$. If $\bar{K}_\epsilon^{(1)}$ reduces the number of required iterations compared to $\bar{K}_\epsilon^{(2)}$ by less than a factor of $\frac{\alpha+1}{\alpha+\beta}$, then the total execution time of that iterative method is less when using $\bar{K}_\epsilon^{(1)}$.*

Lemma 5.0.12 encourages us to consider partitions other than the ones that minimize the interface length. Parts shaped like circles or squares minimize interface length. A natural alternative is using rectangles for parts.



(a) uniform rectangles (b) METIS rectangles

Figure 5.3: Unit square partitioned in 64 rectangle parts.

The length of the interface from uniform rectangles with aspect ratio of 4:1 in Figure 5.3a equals 18.0 while the length from METIS rectangles in Figure 5.3b equals 19.6. Compare these to uniform squares (Figure 5.2) that have length 14.0 and METIS squares (Figure 5.1a) that have length 16.4. These rectangles have interfaces that are approximately 25% and 20% longer respectively.

Theorem 5.0.13. *Given a domain $\Omega \in \mathbb{R}^2$ with area A and boundary perimeter*

length S, if $\bar{K}_\epsilon$ partitions this domain into rectangles with aspect ratio θ:1 and area equal $\frac{A}{P}$, then $\bar{\delta}(\bar{K}_\epsilon) = \frac{1+\theta}{\sqrt{\theta}}\sqrt{AP} - 0.5S$.

Proof. Let each part Ω_k be a rectangle with aspect ratio θ:1 and area $\frac{A}{P}$. Then the short side of the rectangle equals $\sqrt{A/(\theta P)}$ and the perimeter equals $(2 + 2\theta)\sqrt{A/(\theta P)}$. Apply Lemma 5.0.4. □

Corollary 5.0.14. *Let $\Omega \in \mathbb{R}^2$ be a domain with area A and boundary length S. Let $\bar{K}_\epsilon^{rect}$ be a partition of Ω into P rectangle parts with aspect ratio $\theta : 1$ and let $\bar{K}_\epsilon^{square}$ be a partition of Ω into P square parts. Assume both partitions have parts of area $\frac{A}{P}$. Assume $2\sqrt{AP} \gg 0.5S$, then $\bar{\delta}(\bar{K}_\epsilon^{rect})/\bar{\delta}(\bar{K}_\epsilon^{square}) = \frac{1+\theta}{2\sqrt{\theta}}$.*

This Corollary agrees with our findings above where a partition of uniform rectangles with aspect ratio 4:1 produced interfaces that were 25% longer than using uniform squares. Table 5.1 summarizes the interface length growth factor, β, for different rectangle aspect ratios, θ.

Table 5.1: Aspect ratio versus growth factor

θ	1	2	3	4	5	6	7	8	9	10
β	1.00	1.06	1.15	1.25	1.34	1.43	1.51	1.59	1.67	1.74

Given a triangularization T_H of a domain Ω and two partitions $K_\epsilon^{(1)}$ and $K_\epsilon^{(2)}$, which one is better? After our Domain Decomposition Method converges, the separate processors combine their solutions to produce a global solution u_k which approximates u_h, the finite element solution to PDE (1.1)-(1.3), to a tolerance of our choosing. Regardless of the partition, each global solution has its degrees of freedom in the same locations and will approach the same u_h. Even if each part refines its mesh further in a uniform fashion to a target number of unknowns during the DD Method, both $K_\epsilon^{(1)}$ and $K_\epsilon^{(2)}$ will still have the same global degrees of freedom and will approach the same u_h. Therefore the better partition is the one that converges the fastest.

When a domain has been uniformly discretized and represented as a graph, all partitioning algorithms will attempt to generate parts shaped as close to circles as possible when the edge weights are all equal to 1. Rectangular parts can be

encouraged by adding weight to the edges in a non uniform fashion. Using rectangular parts increases edge cuts and communication time, but if it achieves a faster convergent rate, it can cause the DD Method to solve for u_h faster. This is the basis of the following three Weighting Schemes; Convection Weighting, Gradient Weighting, and Stiffness Matrix Weighting. In Chapter 7, we discuss the result of weighting the vertices of G.

5.1 Convection Weighting

In equations (1.1)-(1.3), when $b \neq 0$ then convection is present. Convection causes unknowns to become dependent on the unknowns that are up and downwind of them. In order to solve for an unknown, you need to know the value of the neighboring unknown that is upwind against the direction of the convection. Therefore when partitioning a domain, it is reasonable to group degrees of freedom and their neighbors in the direction of convection together. This is Convection Weighting.

Definition 5.1.1. *Convection Weighting* for the solution of (1.1)-(1.3). Let T be a triangularization of domain Ω and $G = (V, E)$ be its representative graph. Edge $e_k = (v_i, v_j) = (v_j, v_i)$ is associated with the side shared by triangles t_i and t_j. Let $n_{i,j}$ denote the unit normal of this triangle side. Define the weight matrix W_G as

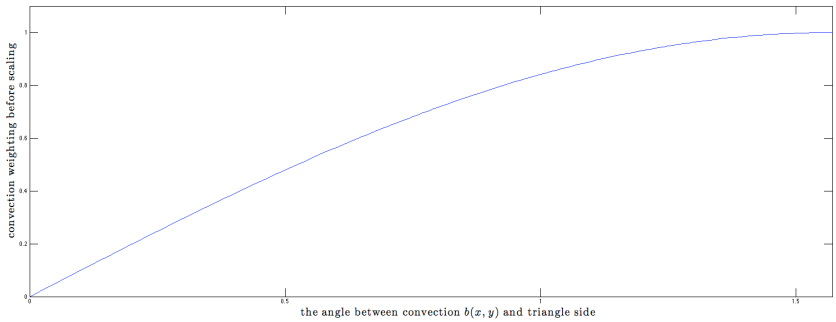
$$w_{i,j} = \begin{cases} q_s\left(n_{i,j} \cdot \frac{b(x,y)}{||b(x,y)||}\right) & \text{if } (v_i, v_j) \in E \\ 0 \quad \text{otherwise} \end{cases} \tag{5.4}$$

where $q_s(\cdot) : [0, 1] \to [1, s^s + 1]$ is a scaling function chosen based on the graph partitioner being used. The variable s is called the *convection weighting parameter* and

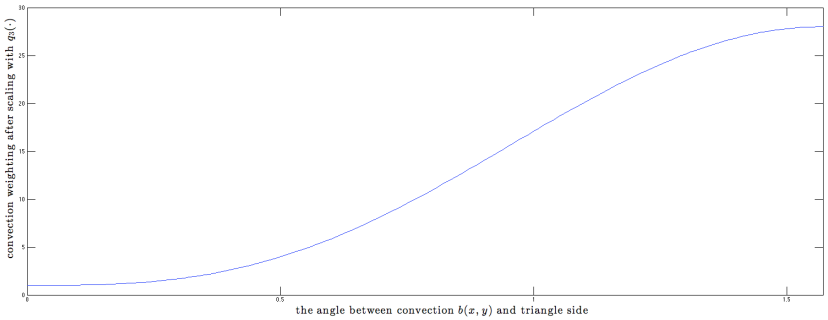
$$q_s(r) = (sr)^s + 1 \tag{5.5}$$

Convection weighting adds weight to graph edges parallel to convection. This corresponds to adding weight to any triangle side that is perpendicular to the direction of convection. Thus convection weighting discourages whatever graph partitioning algorithm you are using from cutting against convection.

We also designed a convection weighting scheme that encourages cutting with convection. However, we found that the nature of METIS and PLTMG work better with a discouraging scheme. Notice that the scaling function $q_s(\cdot)$ both scales the dot product and transforms it. After applying $q_s(\cdot)$, triangle sides with angles in relation to convection $\in [0, \pi/6]$ are mainly unweighted while sides situated with angles $\in [\pi/3, \pi/2]$ are weighted to be discouraged from being cut. Other scaling functions were tried, but $q_s(\cdot)$ proved most sucessful. Figure 5.4 illustrates the transformation. We found that $s = 2.0$ matches this weighting scheme to METIS's partitioner and $s = 3.0$ matches it to PLTMG's partitioner to produce rectangles with aspect ratio 5:1.



(a) $w_{i,j}$ before scaling

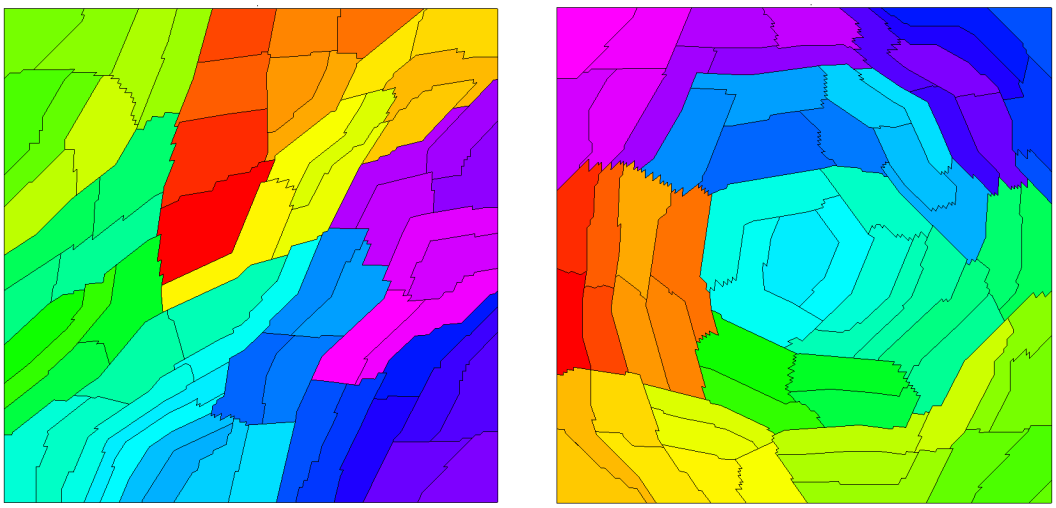


(b) $w_{i,j}$ after scaling with $q_3(\cdot)$

Figure 5.4: The effect of $q(\cdot)_s$ with $s = 3$ on Convection weighting.

Figure 5.5 shows two partitions of the unit square $\in \mathbb{R}^2$. Each has 64 parts and a different PDE. In Figure 5.5a, $b = \beta \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ and in Figure 5.5b, $b =$

$\beta \begin{bmatrix} 0.5 - y \\ x - 0.5 \end{bmatrix}$ where b is the coefficient of ∇u in the PDE (1.1)-(1.3) and β is a scaler.



(a) diagonal convection (b) circular convection

Figure 5.5: METIS using the Convection Weighting scheme.

5.2 Gradient Weighting

The solutions to many PDEs with strong convection have gradients in the direction of convection. This enables us to implement the Convection Weighting scheme by using the gradient instead of the convection coefficient b. Figure 5.6a shows the solution of $-\triangle u - \beta u_x - 1 = 0$ on the unit square with $u = 0$ dirichlet boundary conditions. Notice the correlation between the gradient and convection. Strong convection flowing into the dirichet boundary on the y-axis creates a boundary layer. This boundary layer results in a gradient traveling off the boundary in the direction of convection.

Definition 5.2.1. *Gradient Weighting* for the solution of (1.1)-(1.3). Let T be a triangularization of domain Ω and $G = (V, E)$ be its representative graph. Edge $e_k = (v_i, v_j) = (v_j, v_i)$ is associated with the side shared by triangles t_i and t_j. Let

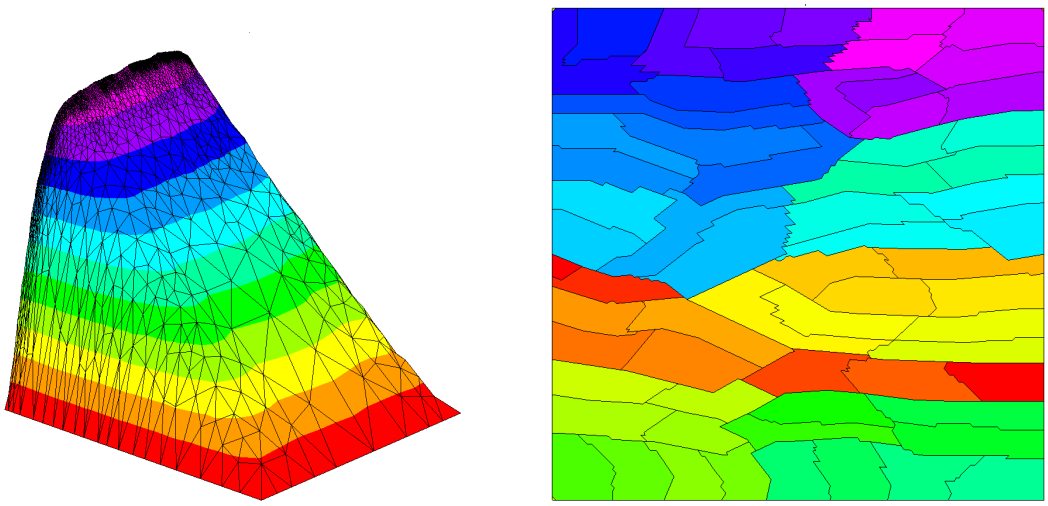
$n_{i,j}$ denote the unit normal of this triangle side. Define the weight matrix W_G as

$$w_{i,j} = \begin{cases} q_s\left(n_{i,j} \cdot \frac{\nabla u(x,y)}{||\nabla u(x,y)||}\right) & \text{if } (v_i, v_j) \in E \\ 0 \quad \text{otherwise} \end{cases} \tag{5.6}$$

where $q_s(\cdot) : [0,1] \to [1, s^s + 1]$ is a scaling function chosen based on the graph partitioner being used.

$$q_s(r) = (sr)^s + 1 \tag{5.7}$$

Similar to Convection Weighting, $s = 2$ matches these weighting schemes to METIS's partitioner and $s = 3$ matches them to PLTMG's partitioner to produce rectangles with aspect ratio 5:1. Figure 5.6 shows METIS using Gradient Weighting.

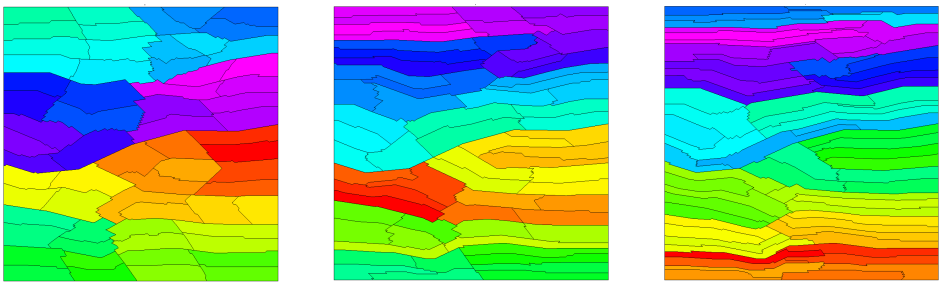


(a) solution u (b) gradient weighted partition

Figure 5.6: METIS using the Gradient Weighting scheme.

Both Convection Weighting and Gradient Weighting cause partitioners to make rectangle shaped parts. The choice of parameter s in the scaling function $q_s(\cdot)$ together with a specific graph partitioner's sensitivity determines the aspect ratio of the rectangles. Elongated rectangles have a longer interface length and greater element cut as shown in Table 5.1. Based on the graph partitioner you use, parameter s should be chosen to create aspect ratios $\in [3, 5]$ which have been shown to maximize convergence increase at the expense of minimally increasing

element cut (only 15 to 34 % more). Once s is determined for your partitioner, you don't need to change it anymore.



(a) $s = 2.0$ (b) $s = 2.75$ (c) s=4.0

Figure 5.7: Metis partitioning using different s parameters.

Gradient Weighting may have a few potential advantages over Convection Weighting. First, if you don't have access to the underlying PDE but you only have access to approximations of the solution u, then you can use the information that you have. Second, Gradient Weighting might improve more PDEs than just convection dominated PDEs. Gradient Weighting can also detect the presence of singularities in the solution. In Section 10.1 we describe how Gradient Weighting may improve the convergence of PDEs with singularities. If this proves true, then the Gradient Weighting scheme will simultaneously deal with convection problems and singular problems.

5.3 Stiffness Matrix Weighting

Stiffness Matrix Weighting is motivated by Convection Weighting but it accomplishes what Convection Weighting can do plus more. When the pure diffusion problem $-\triangle u = f$ is solved with linear elements on a uniform mesh of triangles, the diagonal elements of a specific row within the stiffness matrix have magnitudes four or more times greater than the off diagonal elements for 2-D problems (and are six or more times greater for 3-D). The presence of convection challenges this dominance. When very strong convection is present (and the gradient is approximated with a first order approximation), the ratio of the largest magnitude off

diagonal element to the diagonal element approaches one to one.

When only convection and diffusion are present such as $-\nabla\cdot a\nabla u - b\cdot\nabla u = f$, most rows of the stiffness matrix sum to 0. Therefore as convection increases in one direction and the ratio of the diagonal element and an element corresponding with an unknown in this convection direction approaches 1, the ratios of the diagonal element with the other directions approach 0. We use this fact to build the Stiffness Matrix Weighting scheme.

Definition 5.3.1. *Stiffness Matrix Weighting* for the solution of (1.1)-(1.3). Let T be a triangularization of domain Ω and $G = (V, E)$ be its representative graph. Edge $e_k = (v_i, v_j) = (v_j, v_i)$ is associated with the side s_k shared by triangles t_i and t_j. Let the finite element solution u_h belong to the space of piecewise linear functions defined by linear Lagrange basis functions at the vertices of each triangle. Let A be this system's stiffness matrix defined in (2.9). The degrees of freedom U_m and U_{m+1} are the endpoints of s_k. The remaining two degrees of freedom associated with triangles t_i and t_j are U_{m+2} and U_{m+3}. Refer to Figure 5.8 for the definitions of e_k, s_k, v_i, v_j, t_i, t_j, U_m, U_{m+1}, U_{m+2}, and U_{m+3}.

Define the weight matrix W_G as

$$\begin{aligned} w_{i,j} = {} & \alpha \max\left\{ \frac{|A_{m+2,m}| + |A_{m+2,m+1}|}{2|A_{m+2,m+2}|}, \frac{|A_{m,m+2}|}{2|A_{m,m}|} + \frac{|A_{m+1,m+2}|}{2|A_{m+1,m+1}|} \right\} \\ & + \alpha \max\left\{ \frac{|A_{m+3,m}| + |A_{m+3,m+1}|}{2|A_{m+3,m+3}|}, \frac{|A_{m,m+3}|}{2|A_{m,m}|} + \frac{|A_{m+1,m+3}|}{2|A_{m+1,m+1}|} \right\} + \beta \end{aligned} \tag{5.8}$$

where α and β are scaling constants chosen based on the graph partitioner being used.

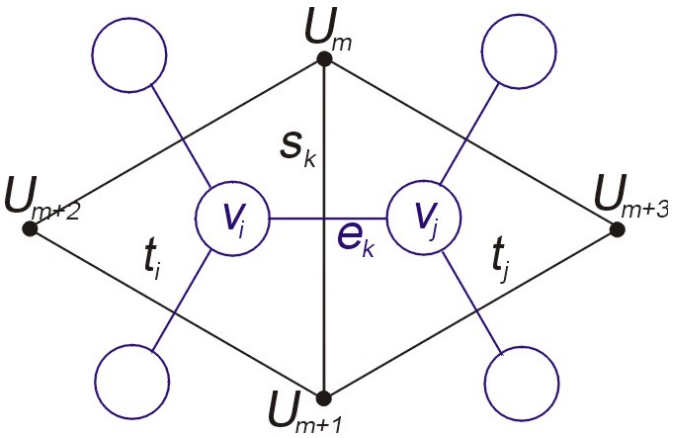


Figure 5.8: Graph G and triangularization T

METIS needs integer weights, so we use $\alpha = 100$, $\beta = 1$ with METIS. Note that when $\alpha \gg \beta$, then regardless of their values, the aspect ratio of the rectangle parts produced is the same as $\alpha = 1$ and $\beta = 0$. PLTMG can use real values so we left $\alpha = 1$ and $\beta = 0$, however the weight needed to be transformed with $q_4(\cdot)$ from (5.7) for PLTMG to operate. The $q_s(\cdot)$ transformation alters the aspect ratio of rectangle parts. The Stiffness Matrix weight function before scaling can be seen in Figures 5.11 and 5.12.

5.3.1 Convection

One of the features of using Stiffness Matrix Weighting is that is detects convection and discourages the partitioner from cutting against convection. It accomplishes this because the formula uses the ratio of stiffness elements corresponding with physical unknowns in one space direction versus stiffness elements corresponding with physical unknowns in the perpendicular space direction. To understand this, we will look at a finite element stiffness matrix in more detail.

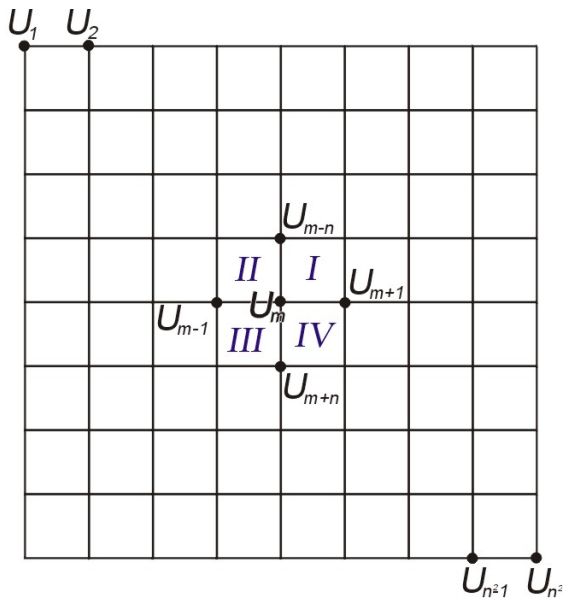


Figure 5.9: Domain Ω with triangularization T

Example 5.3.2. Let domain Ω have a uniform square triangularization as shown in Figure 5.9. Find the stiffness matrix of a finite difference approximation of Poisson's Equation; find $u \in H^2(\Omega)$ such that $-\triangle u - f(x,y) = 0$ on Ω and $u = 0$ on $\partial\Omega$. The length of the side of each square is h.

We will mimic a linear Finite Element stiffness matrix which is second order accurate by using the approximations

$$\partial u_{xx} = \frac{1}{h^2}\left(U_{j+1} - 2U_j + U_{j-1}\right) \tag{5.9}$$

$$\partial u_x = \frac{1}{2}\frac{1}{h}\left(U_{j+1} - U_{j+1}\right) \tag{5.10}$$

The m^{th} row of A is

$$-\frac{1}{h^2}\left(U_{m+1} - 2U_m + U_{m-1}\right) - \frac{1}{h^2}\left(U_{m+n} - 2U_m + U_{m-n}\right) = F_m \tag{5.11}$$

therefore

$$[A]_{m,m-1} = [A]_{m,m-n} = [A]_{m,m+1} = [A]_{m,m+n} = (-1)\frac{1}{h^2} \tag{5.12}$$

$$[A]_{m,m} = (4)\frac{1}{h^2}$$

with the exception that

$$[A]_{m,m-1} = [A]_{m+n-1,m+n} = 0 \text{ when } m \bmod n = 1 \tag{5.13}$$

The stiffness matrix A is $n \times n$ block tridiagonal toeplitz. The off diagonal blocks are the identity matrix times $-\frac{1}{h^2}$. And the diagonal blocks are $n \times n$ tridiagonal toeplitz matrices.

$$A = \frac{1}{h^2}\begin{bmatrix} T & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & & -I \\ & & -I & T \end{bmatrix}, \quad T = \begin{bmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & & -1 \\ & & -1 & 4 \end{bmatrix} \tag{5.14}$$

Therefore in the absence of convection

$$\frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} = \frac{1}{2} \tag{5.15}$$

$$\frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = \frac{1}{2} \tag{5.16}$$

The fractions (5.15) and (5.16) sum to 1 and half of the diagonal support is from corresponding physical unknowns in the x space direction and half is from corresponding physical unknowns in the y space direction. Now let's see what happens when convection is present.

Example 5.3.3. Let domain Ω have a uniform square triangularization as shown in Figure 5.9. Find the stiffness matrix of a finite difference approximation of the convection diffusion equation; find $u \in H^2(\Omega)$ such that $-\triangle u + b \cdot \nabla u - f(x,y) = 0$ on Ω and $u = 0$ on $\partial\Omega$. Let $b = [\beta \ \ 0]^T$ and $\beta > 0$. The length of the side of each square is h.

We will mimic a linear Finite Element stiffness matrix which is second order accurate by using the approximations

$$\partial u_{xx} = \frac{1}{h^2}\left(U_{j+1} - 2U_j + U_{j-1}\right) \tag{5.17}$$

$$\partial u_x = \frac{1}{2}\frac{1}{h}\left(U_{j+1} - U_{j+1}\right) \tag{5.18}$$

The m^{th} row of A is

$$\begin{aligned} -\frac{1}{h^2}\left(U_{m+1} - 2U_m + U_{m-1}\right) - \frac{1}{h^2}\left(U_{m+n} - 2U_m + U_{m-n}\right) \\ +\frac{1}{2}\beta\frac{1}{h}\left(U_{j+1} - U_{j+1}\right) = F_m \end{aligned} \tag{5.19}$$

therefore

$$\begin{aligned} [A]_{m,m-n} &= [A]_{m,m+n} = (-1)\frac{1}{h^2} \\ [A]_{m,m-1} &= (-1)\frac{1}{h^2} - \frac{1}{2}\beta\frac{1}{h} \\ [A]_{m,m+1} &= (-1)\frac{1}{h^2} + \frac{1}{2}\beta\frac{1}{h} \\ [A]_{m,m} &= (4)\frac{1}{h^2} \end{aligned} \tag{5.20}$$

with the exception that

$$[A]_{m,m-1} = [A]_{m+n-1,m+n} = 0 \text{ when } m \bmod n = 1 \tag{5.21}$$

The stiffness matrix A is $n \times n$ block tridiagonal toeplitz. The off diagonal blocks are the identity matrix times $-\frac{1}{h^2}$. And the diagonal blocks are $n \times n$ tridiagonal toeplitz matrices.

$$\begin{aligned} A &= \frac{1}{h^2}\begin{bmatrix} T & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & & -I \\ & & -I & T \end{bmatrix} \\ T &= \begin{bmatrix} 4 & -1+\frac{1}{2}\beta h & & \\ -1-\frac{1}{2}\beta h & \ddots & \ddots & \\ & \ddots & & -1+\frac{1}{2}\beta h \\ & & -1-\frac{1}{2}\beta h & 4 \end{bmatrix} \end{aligned} \tag{5.22}$$

Therefore when convection is present and $\beta h > 2$

$$\frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} = \frac{\beta h}{4} > \frac{1}{2} \tag{5.23}$$

$$\frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = \frac{1}{2} \tag{5.24}$$

The fractions (5.23) and (5.24) sum to greater than 1 and using classic methods to solve $AU = F$ become unstable. Therefore in practice, artificial diffusion is added to the stiffness matrix as described in Section 2.2 in the form of

$$\tilde{A} = \frac{1}{h^2}\begin{bmatrix} \tilde{T} & 0 & & \\ 0 & \ddots & \ddots & \\ & \ddots & & 0 \\ & & 0 & \tilde{T} \end{bmatrix}, \quad \tilde{T} = \begin{bmatrix} \beta h & -\frac{1}{2}\beta h & & \\ -\frac{1}{2}\beta h & \ddots & \ddots & \\ & \ddots & & -\frac{1}{2}\beta h \\ & & -\frac{1}{2}\beta h & \beta h \end{bmatrix} \tag{5.25}$$

Then the new stiffness matrix is $A^* = A + \tilde{A}$.

$$A^* = \frac{1}{h^2}\begin{bmatrix} T^* & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & & -I \\ & & -I & T^* \end{bmatrix}, \quad T^* = \begin{bmatrix} 4+\beta h & -1 & & \\ -1-\beta h & \ddots & \ddots & \\ & \ddots & & -1 \\ & & -1-\beta h & 4+\beta h \end{bmatrix} \tag{5.26}$$

and

$$1 > \frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} = \frac{2+\beta h}{4+\beta h} > \frac{1}{2} \tag{5.27}$$

$$0 < \frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = \frac{2}{4+\beta h} < \frac{1}{2} \tag{5.28}$$

and now the fractions (5.27) and (5.28) sum to 1 when convection is present

$$\frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} + \frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = 1 \tag{5.29}$$

The Stiffness Matrix Weighting scheme mimics the fractions in (5.27)-(5.28) and therefore can detect the convection from the stiffness matrix in (5.26) when its values differ from the values in the stiffness matrix and fractions of (5.14)-(5.16).

The Stiffness Matrix Weighting scheme (formula (5.8)) looks messy, but one can understand (5.8) simply as an approximation of (5.27) for the PDE (1.1)-(1.3) acting on the square mesh pictured as the dotted line in Figure 5.10 with stiffness matrix denoted $\bar{A}$. The weight $w_{i,j}$ for edge e_k which connects graph vertices v_i and v_j can be thought of as

$$w_{i,j} \approx \frac{|\bar{A}_{r,r-1}| + |\bar{A}_{r,r+1}|}{|\bar{A}_{r,r}|} \tag{5.30}$$

This is a crude approximation, but it helps to conceptualize (5.8). A less simplified conceptualization of (5.8) but a more accurate approximation is

$$w_{i,j} \approx \frac{|\bar{A}_{r,r-1}| + |\bar{A}_{r,r+1}| + \frac{1}{4}\left(|\bar{A}_{r,r-n}| + |\bar{A}_{r,r+n}|\right)}{|\bar{A}_{r,r}|} \tag{5.31}$$

Equation (5.8) uses the stiffness matrix from a triangle mesh. The geometry of triangles doesn't separate the x and y directions like squares do so the numerator in (5.30) becomes a better approximation with the additional terms shown in (5.31). (To understand these terms, read Example 5.5.1.) The nature of a triangle mesh prevents $w_{i,j}$ from becoming equal to zero even when there is very strong convection in the y direction. In fact, $w_{i,j} \geq \frac{1}{4}\ \forall i, j$ and when Scharfetter Gummel upwinding is used, $w_{i,j} \leq 1$.

Remark 5.3.4. Therefore, when the Stiffness Matrix Weighting scheme is used on a triangle mesh solving a convection dominated problem, it produces rectangles with aspect ratios of $4:1$ (when $\alpha = 1, \beta = 0$).

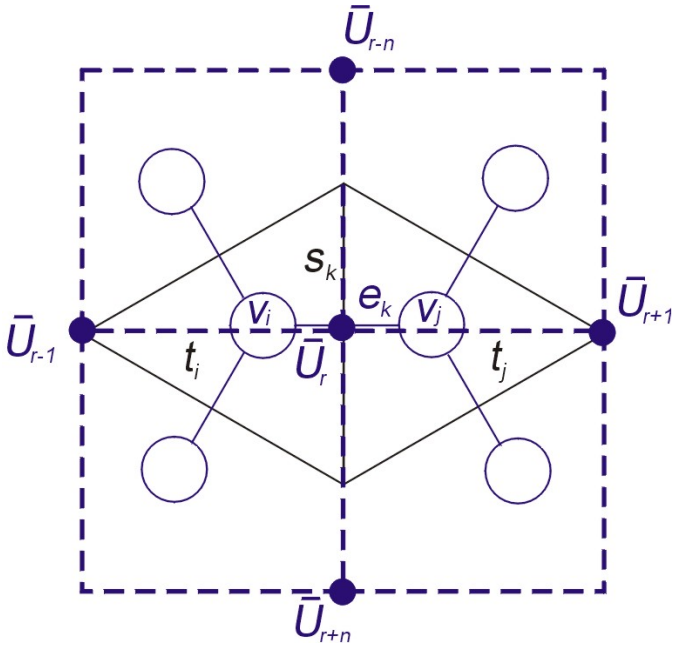


Figure 5.10: An approximation of the stiffness matrix weighting scheme

The preceding discussion explains how the Stiffness Matrix Weighting scheme adds more weight to triangle sides that are perpendicular to convection than sides

that are parallel to convection. (Technically, weighting schemes add weight to graph edges not triangle sides. But the two objects have a one to one correspondence, so informally, one can interchange the two words for clarity.)

In Section 5.3.1, Figure 5.4a illustrated how much weight the Convection and Gradient Weighting schemes added to triangle sides oriented at various angles to convection (before scaling). Similarly, Figures 5.11 and 5.12 below show how much weight the Stiffness Matrix Weighting scheme adds to triangle sides oriented at various angles to convection (before scaling) Both figures assume a uniform equilateral mesh, side h, with upwinding being employed to solve the convection diffusion equation $-\triangle u - b \cdot \nabla u - f(x, y) = 0$.

Each figure has five lines. The constant line is when $b = 0$. The top line is the limit as $||b|| \to \infty$. When I plotted $||b|| = 100/h$ and $||b|| = 1000/h$ they both coincided with this limit. The three in between lines are $||b|| = 1/h$, $||b|| = 3/h$, and $||b|| = 10/h$.

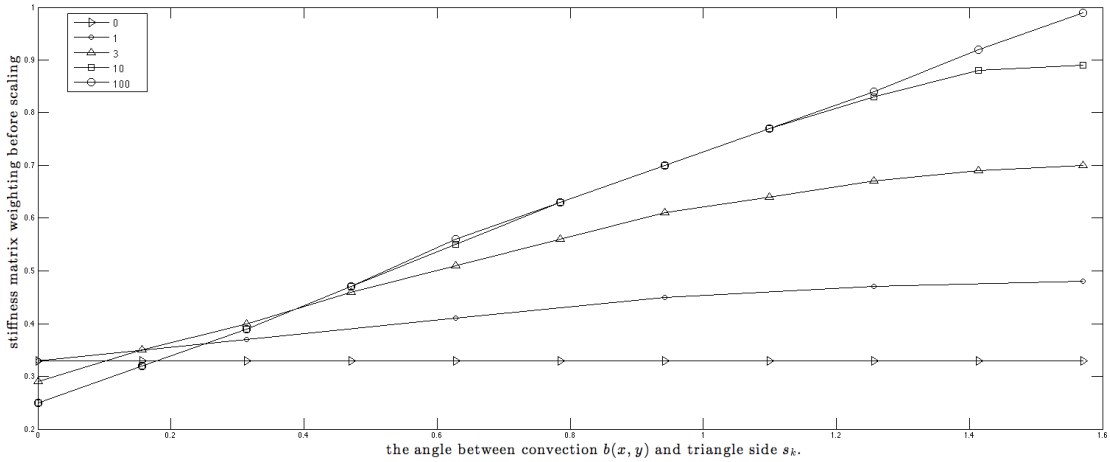


Figure 5.11: Stiffness Matrix weighting when Scharfetter Gummel upwinding is used. Each line represents a different $h||b||$

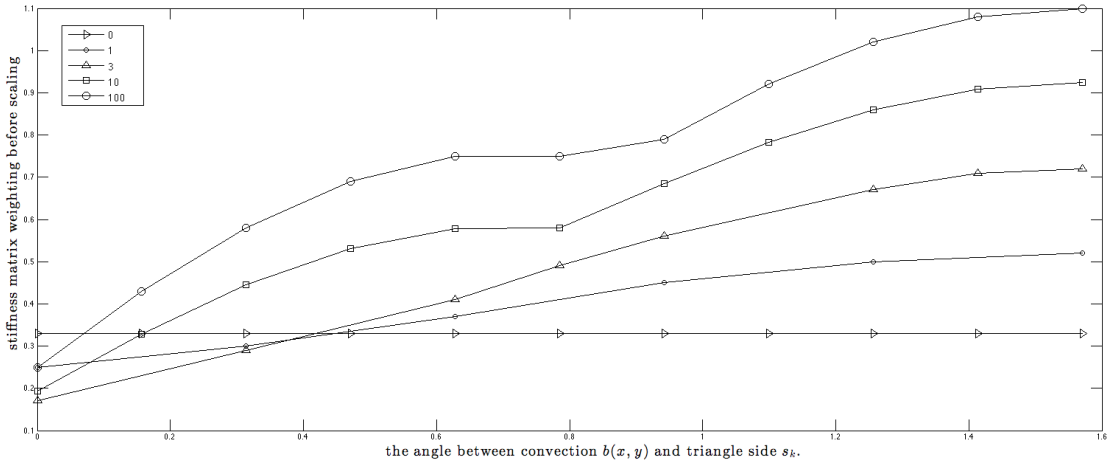


Figure 5.12: Stiffness Matrix weighting when Streamline Diffusion upwinding is used. Each line represents a different $h||b||$

5.3.2 Diffusion

Another feature of using Stiffness Matrix Weighting is that is detects anisotropic diffusion and discourages the partitioner from cutting against the prominent diffusion direction. It accomplishes this in the same way that it detected convection.

Example 5.3.5. Let domain Ω have a uniform square triangularization as shown in Figure 5.9. Find the stiffness matrix of a finite element approximation of the PDE; find $u \in H^2$ such that $-\nabla \cdot a\nabla u - f(x,y) = 0$ on Ω and $u = 0$ on $\partial\Omega$. Let $a = \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix}$ and $\alpha > 1$. The length of the side of each square is h.

We find the stiffness matrix A the same way that we did in Examples 5.3.2 and 5.3.3.

$$A = \frac{1}{h^2}\begin{bmatrix} T & I & & \\ I & \ddots & \ddots & \\ & \ddots & & I \\ & & I & T \end{bmatrix}, \quad T = \begin{bmatrix} 2+2\alpha & -\alpha & & \\ -\alpha & \ddots & \ddots & \\ & \ddots & & -\alpha \\ & & -\alpha & 2+2\alpha \end{bmatrix} \tag{5.32}$$

$$1 > \frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} = \frac{2\alpha}{2+2\alpha} > \frac{1}{2} \tag{5.33}$$

$$0 < \frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = \frac{2}{2+2\alpha} < \frac{1}{2} \tag{5.34}$$

and

$$\frac{|A_{m,m-1}| + |A_{m,m+1}|}{|A_{m,m}|} + \frac{|A_{m,m-n}| + |A_{m,m+n}|}{|A_{m,m}|} = 1 \tag{5.35}$$

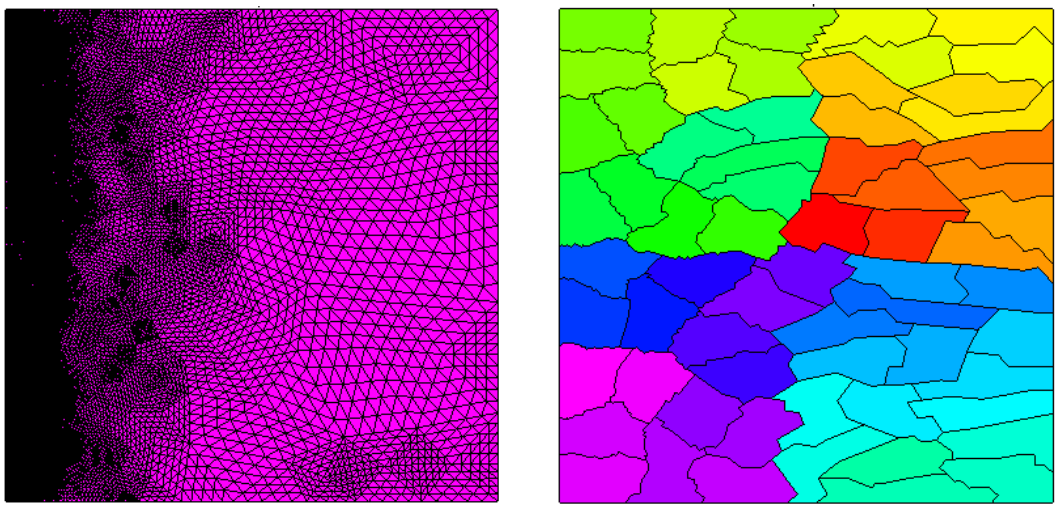
5.3.3 Self Adjusting

Another feature of Stiffness Matrix Weighting is that it regulates itself. As stated previously, favoring convection and creating partitions with rectangle parts increases interface length and element cut. See Table 5.1. Therefore you only want to use rectangles when using them will speed up convergence sufficiently. This corresponds to when the PDE is dominated by convection. But how much convection classifies it as dominating? From the discussions above, we see that convection is scaled by a factor of h when influencing the stiffness matrix. Therefore the question of whether convection is dominating relates to both the coefficient b on ∇u and the mesh size h. In Chapter 9 sections 9.2.2 and 9.2.6, we provide numerical experiments that investigate the relationship between convection dominance, b, h, and P the number of parts.

Our numerical experiments led us to create a model of when strong convection justifies modifying the partition. Namely, use rectangle parts when approximately $h||b|| \geq 1$. This model coincides perfectly with how the Stiffness Matrix Weighting self regulates. As you can see from the weighting scheme (5.8) and (5.27), the scheme bases its weights on both b and h. Figures 5.11 and 5.12 show that $h||b|| = 1$ will encourage rectangles of aspect ratio 1.45:1 and $h||b|| = 3$ will encourage 2.4:1 and $h||b|| = 10$ will encourage 3.6:1 and as $h||b|| \to \infty$ the aspect ratio will converge to 4:1. Figure 5.14 shows rectangle aspect ratio versus convection strength when using Scharfetter Gummel upwinding. (Other upwinding schemes produce similar results.)

Figure 5.13 shows the Stiffness Matrix Weighting scheme regulating itself in action. The unit square was discretized into a non uniform mesh where h gradually gets larger when moving in the x direction. The equation, $-\triangle u + b \cdot \nabla u - f(x,y) = 0$

was then solved. In the middle of the left side ($x \approx 0.25$), $h||b|| = 2$ and in the middle of the right side ($x \approx 0.75$), $h||b|| = 8$.



(a) non uniform mesh (b) self regulating rectangles

Figure 5.13: Stiffness matrix weighting scheme regulating based on h and b.

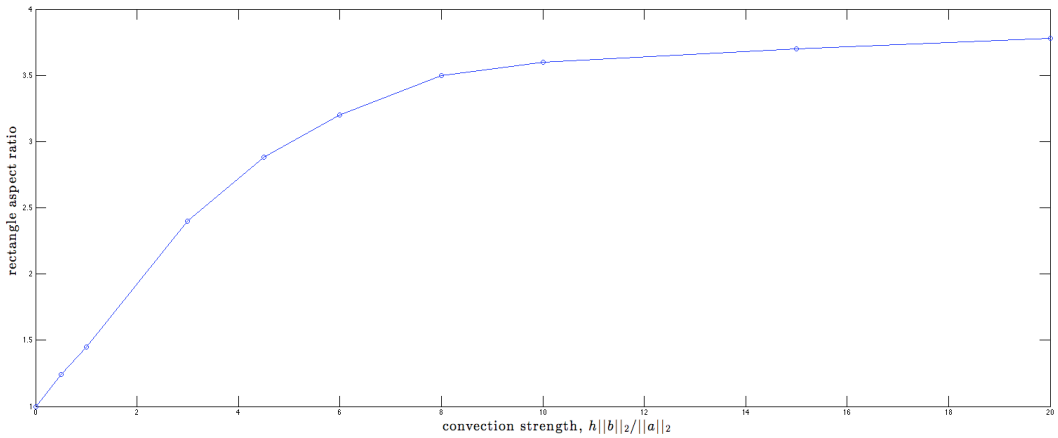


Figure 5.14: Rectangle aspect ratio versus convection strength

5.4 Computation Time

In Lemmas 5.0.11 and 5.0.12, we assumed that computation time is independent of interface length. This is not always true. Each parallel Domain

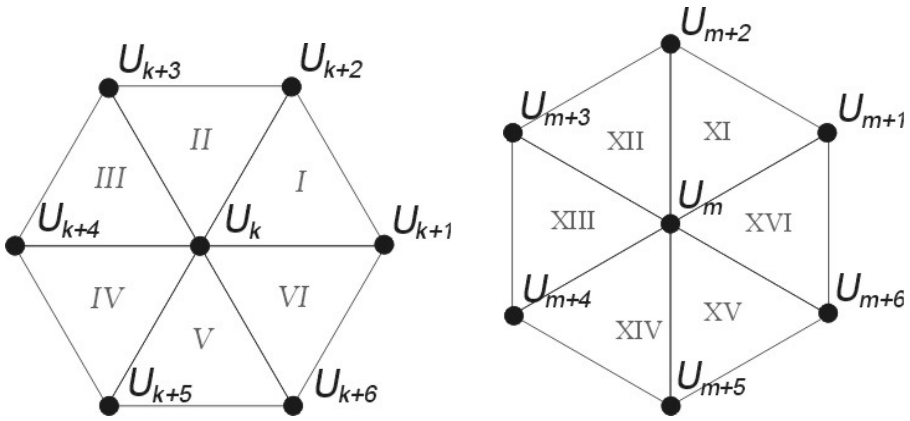
Decomposition Method performs different computations during one iteration than the other methods. A processor updates the unknowns it is responsible for using data that it receives from other processors. We saw that Domain Decomposition Methods pass data proportional to their interface lengths, but they each use this data differently. If a certain DD Method's computation time is proportional to this data, we can still use Lemmas 5.0.11 and 5.0.12.

Instead of splitting the time of one iteration, t, into computation time, t^{comp}, and communication time, t^{comm} which we assumed were independent and proportional respectively, we can split t into t^A and t^B. Let t^A be the portion of the time of one iteration that is independent to interface length and let t^B be the portion that is proportional.

Also, we assumed that t^B was linearly proportional to interface length, $t^B = \alpha\bar{\delta}(\bar{K}_\epsilon)$. If t^B is proportional to either the square of the interface length or its cube, Lemmas 5.0.11 and 5.0.12 can be modified easily to accommodate this.

5.5 Edge Weighting Example

Example 5.5.1. Describe the partition of the unit square in $\mathbb{R}^2$ when using Stiffness Matrix Weighting to solve find $u \in H^2(\Omega)$ such that $-\triangle u + [\beta\ 0]^T \cdot \nabla u + 1 = 0$ on Ω and $u = 0$ on $\partial\Omega$. Assume the domain has a quasi uniform triangularization T with element side length h . Use linear finite elements with Scharfetter-Gummel upwinding.



(a) one (b) two

Figure 5.15: Two triangle mesh stencils

First we will calculate row k of the stiffness matrix A for the interior degree of freedom U_k pictured in Figure 5.15a. Let the coordinate axis be denoted as (r, s) and the basis functions as $v_i(r, s)$. We will use isoparametric elements and map each triangle to the standard reference triangle in quadrant 1 with coordinate axis (x, y) and linear lagrange basis $w_i(x, y)$. Define $A^{\mathrm{d}}_{i,j} = \int_\Omega \nabla v_j \nabla v_i \, drds$ and $A^{\mathrm{c}}_{i,j} = \int_\Omega [\beta \ 0]^T \cdot \nabla v_j v_i \, drds$. Then $A_{i,j} = A^{\mathrm{d}}_{i,j} + A^{\mathrm{c}}_{i,j}$.

For triangles I and II, here are the maps, Jacobians, and reference basis functions. $T_1 = \begin{bmatrix} h & \frac{h}{2} \\ 0 & \frac{\sqrt{3}h}{2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$. $J_1 = \begin{bmatrix} h & 0 \\ \frac{h}{2} & \frac{\sqrt{3}h}{2} \end{bmatrix}$. $J_1^{-1} = \begin{bmatrix} \frac{1}{h} & 0 \\ -\frac{1}{\sqrt{3}h} & \frac{2}{\sqrt{3}h} \end{bmatrix}$. $T_2 = \begin{bmatrix} -\frac{h}{2} & \frac{h}{2} \\ \frac{\sqrt{3}h}{2} & \frac{\sqrt{3}h}{2} \end{bmatrix}$. $J_2 = \begin{bmatrix} -\frac{h}{2} & \frac{\sqrt{3}h}{2} \\ \frac{h}{2} & \frac{\sqrt{3}h}{2} \end{bmatrix}$. $J_2^{-1} = \begin{bmatrix} \frac{1}{h} & -\frac{1}{h} \\ -\frac{1}{\sqrt{3}h} & -\frac{1}{\sqrt{3}h} \end{bmatrix}$. $w_k = 1 - x - y$,

$w_{k+1} = x$, and $w_{k+2} = y$. $\nabla w_k = [-1\ -1]^T$, $\nabla w_{k+1} = [1\ 0]^T$, and $\nabla w_{k+2} = [0\ 1]^T$.

$$\begin{aligned}
A^{\mathrm{d}}_{k,k+2} &= \int_I \nabla v_{k+2} \nabla v_k \, drds + \int_{II} \nabla v_{k+2} \nabla v_k \, drds \\
&= \int_0^1 \int_0^{1-x} J_1^{-1} \nabla w_{k+2} J_1^{-1} \nabla w_k |J_1| \, dydx \\
&\qquad + \int_0^1 \int_0^{1-x} J_2^{-1} \nabla w_{k+2} J_2^{-1} \nabla w_k |J_2| \, dydx \\
&= \int_0^1 \int_0^{1-x} -\frac{2}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dxdy + \int_0^1 \int_0^{1-x} -\frac{2}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dxdy \\
&= -\frac{1}{\sqrt{3}} \qquad (5.36)
\end{aligned}$$

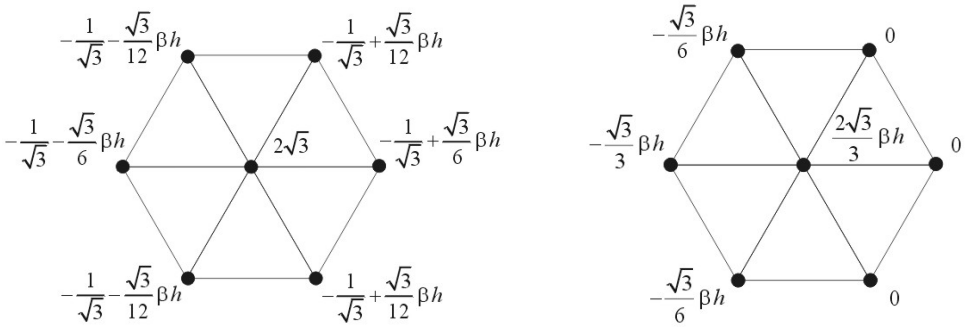
$$A^{\mathrm{d}}_{k,k+1} = A^{\mathrm{d}}_{k,k+3} = A^{\mathrm{d}}_{k,k+4} = A^{\mathrm{d}}_{k,k+5} = A^{\mathrm{d}}_{k,k+6} = -\frac{1}{\sqrt{3}} \qquad (5.37)$$

$$\begin{aligned}
A^{\mathrm{d}}_{k,k} &= 6 \int_I \nabla v_k \nabla v_k \, drds \\
&= 6 \int_0^1 \int_0^{1-x} J_1^{-1} \nabla w_k J_1^{-1} \nabla w_k |J_1| \, dydx \\
&= 6 \int_0^1 \int_0^{1-x} \frac{4}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dydx \qquad (5.38) \\
&= 2\sqrt{3}
\end{aligned}$$

$$\begin{aligned}
A^{\mathrm{c}}_{k,k+6} = A^{\mathrm{c}}_{k,k+2} &= \int_I \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla v_{k+2} v_k \, drds + \int_{II} \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla v_{k+2} v_k \, drds \\
&= \int_0^1 \int_0^{1-x} \begin{bmatrix} \beta \\ 0 \end{bmatrix} J_1^{-1} \nabla w_{k+2} w_k |J_1| \, dydx \\
&\qquad + \int_0^1 \int_0^{1-x} \begin{bmatrix} \beta \\ 0 \end{bmatrix} J_2^{-1} \nabla w_{k+2} w_k |J_2| \, dydx \\
&= \int_0^1 \int_0^{1-x} 0 \, dydx + \frac{\sqrt{3}}{2} \beta h \int_0^1 \int_0^{1-x} (1 - x - y) \, dydx \qquad (5.39) \\
&= \frac{\sqrt{3}}{12} \beta h
\end{aligned}$$

$$
\begin{aligned}
A^{\mathrm{c}}_{k,k+1} &= 2\int_I \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla v_{k+1} v_k \, drds \\
&= 2\int_0^1 \int_0^{1-x} \begin{bmatrix} \beta \\ 0 \end{bmatrix} J_1^{-1} \nabla w_{k+1} w_k |J_1| \, dydx \\
&= \sqrt{3}\beta h \int_0^1 \int_0^{1-x} (1-x-y) \, dydx \qquad (5.40) \\
&= \frac{\sqrt{3}}{6}\beta h \\
A^{\mathrm{c}}_{k,k+4} &= -\frac{\sqrt{3}}{6}\beta h \qquad (5.41) \\
A^{\mathrm{c}}_{k,k+3} &= A^{\mathrm{c}}_{k,k+5} = -\frac{\sqrt{3}}{12}\beta h \qquad (5.42) \\
A^{\mathrm{c}}_{k,k} &= 0 \qquad (5.43)
\end{aligned}
$$

Therefore row k of A is all zeroes except for the seven entries defined above. Figure 5.16a represents the non zeroes in a picture. The number next to node i is the value of $A_{k,i}$.



(a) without upwinding

(b) with Scharfetter-Gummel upwinding

Figure 5.16: Row k of stiffness matrix A with and without upwinding.

Now we will calculate row m of the stiffness matrix A for the interior degree of freedom U_m pictured in Figure 5.15b. For triangles XI and XVI, here are the maps, Jacobians, and reference basis functions. $T_{11} = \begin{bmatrix} \frac{\sqrt{3}h}{2} & 0 \\ \frac{h}{2} & h \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$. $J_{11} = \begin{bmatrix} \frac{\sqrt{3}h}{2} & \frac{h}{2} \\ 0 & h \end{bmatrix}$. $J_{11}^{-1} = \begin{bmatrix} \frac{2}{\sqrt{3}h} & -\frac{1}{\sqrt{3}h} \\ 0 & \frac{1}{h} \end{bmatrix}$. $T_{16} = \begin{bmatrix} \frac{\sqrt{3}h}{2} & \frac{\sqrt{3}h}{2} \\ \frac{h}{2} & -\frac{h}{2} \end{bmatrix}$. $J_{16} = \begin{bmatrix} \frac{\sqrt{3}h}{2} & \frac{h}{2} \\ \frac{\sqrt{3}h}{2} & -\frac{h}{2} \end{bmatrix}$.

$J_{16}^{-1} = \begin{bmatrix} -\frac{1}{\sqrt{3}h} & -\frac{1}{\sqrt{3}h} \\ -\frac{1}{h} & \frac{1}{h} \end{bmatrix}$. $w_m = 1 - x - y$, $w_{m+1} = x$, and $w_{m+6} = y$. $\nabla w_m = [-1 \ -1]^T$, $\nabla w_{m+1} = [1 \ 0]^T$, and $\nabla w_{m+6} = [0 \ 1]^T$.

$$\begin{aligned} A^{\mathrm{d}}_{m,m+1} &= \int_{XI} \nabla v_{m+1} \nabla v_m \, drds + \int_{XVI} \nabla v_{m+1} \nabla v_m \, drds \qquad (5.44) \\ &= \int_0^1 \int_0^{1-x} J_{11}^{-1} \nabla w_{m+1} J_{11}^{-1} \nabla w_m |J_{11}| \, dydx \\ &\qquad + \int_0^1 \int_0^{1-x} J_{16}^{-1} \nabla w_{m+1} J_{16}^{-1} \nabla w_m |J_{16}| \, dydx \\ &= \int_0^1 \int_0^{1-x} -\frac{2}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dxdy + \int_0^1 \int_0^{1-x} -\frac{2}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dxdy \\ &= -\frac{1}{\sqrt{3}} \qquad (5.45) \end{aligned}$$

$$A^{\mathrm{d}}_{m,m+2} = A^{\mathrm{d}}_{m,m+3} = A^{\mathrm{d}}_{m,m+4} = A^{\mathrm{d}}_{m,m+5} = A^{\mathrm{d}}_{m,m+6} = -\frac{1}{\sqrt{3}} \qquad (5.46)$$

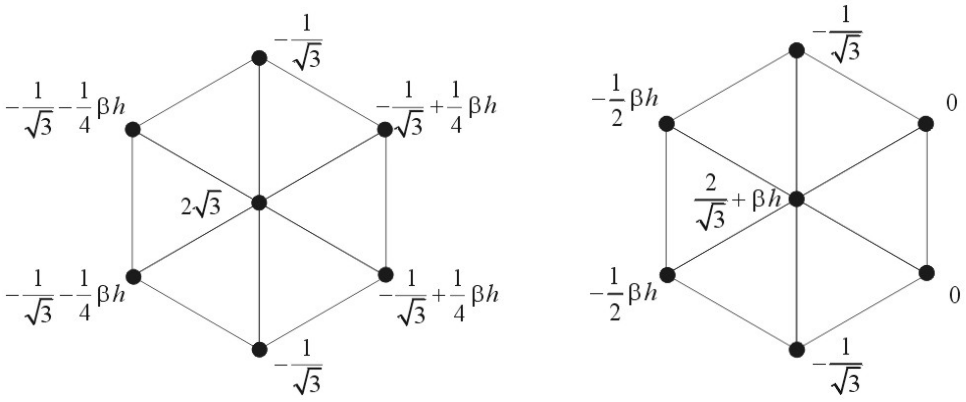
$$\begin{aligned} A^{\mathrm{d}}_{m,m} &= 6 \int_{XI} \nabla v_m \nabla v_m \, drds \\ &= 6 \int_0^1 \int_0^{1-x} J_{11}^{-1} \nabla w_m J_{11}^{-1} \nabla w_m |J_{11}| \, dydx \\ &= 6 \int_0^1 \int_0^{1-x} \frac{4}{3h^2} \left(\frac{\sqrt{3}h^2}{2} \right) dydx \qquad (5.47) \\ &= 2\sqrt{3} \end{aligned}$$

$$\begin{aligned} A^{\mathrm{c}}_{m,m+6} = A^{\mathrm{c}}_{m,m+1} &= \int_{XI} \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla v_{m+1} v_m \, drds + \int_{XVI} \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla v_{m+1} v_m \, drds \\ &= \int_0^1 \int_0^{1-x} \begin{bmatrix} \beta \\ 0 \end{bmatrix} J_{11}^{-1} \nabla w_{m+1} w_m |J_{11}| \, dydx \\ &\qquad + \int_0^1 \int_0^{1-x} \begin{bmatrix} \beta \\ 0 \end{bmatrix} J_{16}^{-1} \nabla w_{m+1} w_m |J_{16}| \, dydx \\ &= \beta \int_0^1 \int_0^{1-x} (1 - x - y) \, dydx + \frac{\beta h}{2} \int_0^1 \int_0^{1-x} (1 - x - y) \, dydx \qquad (5.48) \\ &= \frac{1}{4} \beta h \end{aligned}$$

$$A^{\mathrm{c}}_{m,m+3} = A^{\mathrm{c}}_{m,m+4} = -\frac{1}{4} \beta h \qquad (5.49)$$

$$A^{\mathrm{c}}_{m,m+2} = A^{\mathrm{c}}_{m,m+5} = 0 \qquad (5.50)$$

Therefore row m of A is all zeroes except for the seven entries defined above. Figure 5.17a represents the non zeroes in a picture. The number next to node i is the value of $A_{m,i}$.



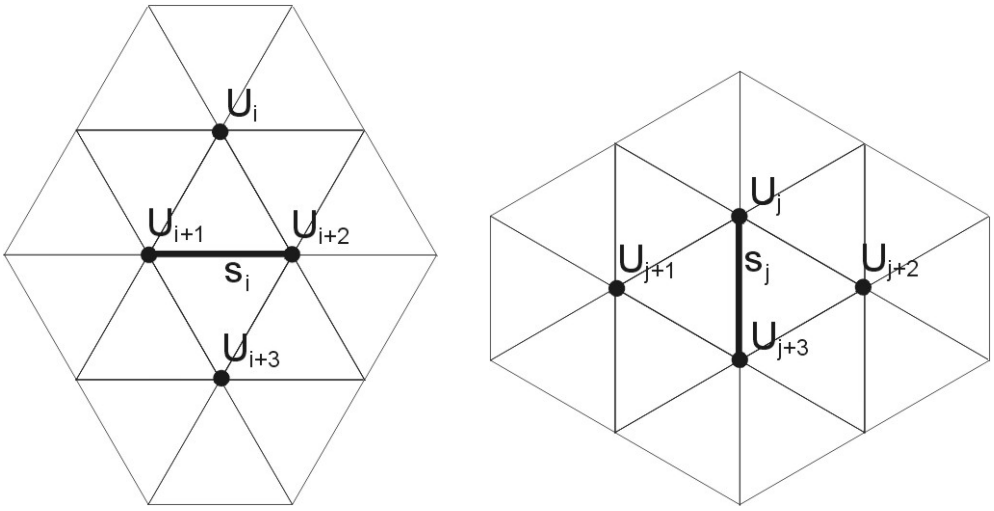
(a) without upwinding

(b) with Scharfetter-Gummel upwinding

Figure 5.17: Row m of stiffness matrix A with and without upwinding.

Let $G = (V, E)$ be the graph representing T. Each edge $e_k \in E$ is associated with a triangle $t_j \in T$. Since the edges have a one to one correspondence with triangle sides, we will refer to edge weights as triangle side weights from now on. The weight of a triangle side depends on the angle between the side and the direction of convection. Triangles in our mesh have sides oriented at all different angles as seen in Figure 5.20. We will calculate the two extreme orientations.

First, consider a triangle side that is parallel to convection as pictured in Figure 5.18a.



(a) convection parallel side (b) convection perpendicular side

Figure 5.18: Different orientations of triangle sides.

From the Stiffness Matrix Weighting formula (5.8) and the stencil in Figure 5.16b

$$\begin{aligned} w(s_i) &= \max\left\{\frac{(|A_{i,i+1}|+|A_{i,i+2}|)}{2|A_{i,i}|}, \frac{|A_{i+1,i}|}{2|A_{i+1,i+1}|}+\frac{|A_{i+2,i}|}{2|A_{i+2,i+2}|}\right\} \\ &\quad + \max\left\{\frac{(|A_{i+3,i+1}|+|A_{i+3,i+2}|)}{2|A_{i+3,i+3}|}, \frac{|A_{i+1,i+3}|}{2|A_{i+1,i+1}|}+\frac{|A_{i+2,i+3}|}{2|A_{i+2,i+2}|}\right\} \\ &= \max\frac{3}{4\sqrt{3}\beta h}\left\{\left(\left|-\frac{\sqrt{3}}{6}\beta h\right|+|0|\right), |0|+\left|-\frac{\sqrt{3}}{6}\beta h\right|\right\} \\ &\quad + \max\frac{3}{4\sqrt{3}\beta h}\left\{\left(\left|-\frac{\sqrt{3}}{6}\beta h\right|+|0|\right), |0|+\left|-\frac{\sqrt{3}}{6}\beta h\right|\right\} \\ &= \frac{1}{4} \end{aligned} \tag{5.51}$$

Second, consider a triangle side that is perpendicular to convection as pictured in Figure 5.18b. From the Stiffness Matrix Weighting formula (5.8) and the

stencil in Figure 5.17b that uses Scharfetter-Gummel upwinding

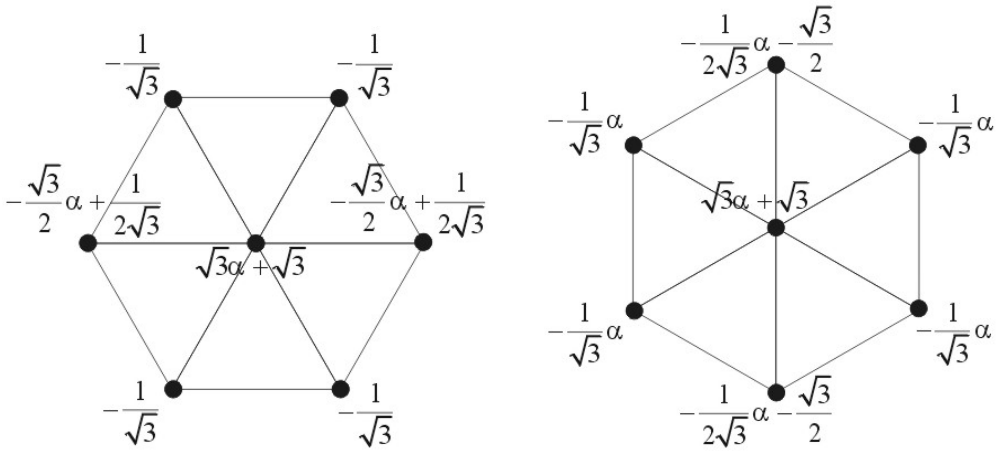
$$\begin{aligned} w(s_j) &= \max\left\{\frac{(|A_{j+1,j}| + |A_{j+1,j+3}|)}{2|A_{j+1,j+1}|}, \frac{|A_{j,j+1}|}{2|A_{j,j}|} + \frac{|A_{j+3,j+1}|}{2|A_{j+3,j+3}|}\right\} \\ &\quad + \max\left\{\frac{(|A_{j+2,j}| + |A_{j+2,j+3}|)}{2|A_{j+2,j+2}|}, \frac{|A_{j,j+2}|}{2|A_{j,j}|} + \frac{|A_{j+3,j+2}|}{2|A_{j+3,j+3}|}\right\} \\ &= \max\frac{1}{\frac{4}{\sqrt{3}} + 2\beta h}\left\{(|0| + |0|), \left|-\frac{1}{2}\beta h\right| + \left|-\frac{1}{2}\beta h\right|\right\} \\ &\quad + \max\frac{1}{\frac{4}{\sqrt{3}} + 2\beta h}\left\{\left(\left|-\frac{1}{2}\beta h\right| + \left|\frac{1}{2}\beta h\right|\right), |0| + |0|\right\} \\ &= \frac{\beta h}{\frac{2}{\sqrt{3}} + \beta h} \approx 1 \text{ when } \beta h \gg 1 \end{aligned} \tag{5.52}$$

These calculations tell us that when $\beta h \gg 1$, triangle sides that are perpendicular to convection will be weighted equal to 1 and triangle sides that are parallel to convection will be weighted equal to $\frac{1}{4}$. Sides oriented at in-between angles will have values in-between. (See Figure 5.11). To minimize edge cut, the partition will have parts shaped like rectangles with aspect ratio $4:1$ pointing in the x axis direction.

Example 5.5.2. Describe the partition of the unit square in $\mathbb{R}^2$ when using Stiffness Matrix Weighting to solve find $u \in H^2(\Omega)$ such that $-\nabla \cdot \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} \nabla u + 1 = 0$ on Ω and $u = 0$ on $\partial\Omega$. Also $\alpha \geq 1$. Assume the domain has a quasi uniform triangularization T with element side length h . Use linear finite elements.

We will follow the same procedure as Example 5.5.1. First we calculate row k of the stiffness matrix A for the interior degree of freedom U_k pictured in Figure 5.15a using isoparametric elements. Row k of A is all zeroes except for seven entries. Figure 5.19a represents the non zeroes in a picture. The number next to node i is the value of $A_{k,i}$.

Next we calculate row m of the stiffness matrix A for the interior degree of freedom U_m pictured in Figure 5.15b using isoparametric elements. Row m of A is all zeroes except for seven entries. Figure 5.19b represents the non zeroes in a picture. The number next to node i is the value of $A_{m,i}$.



(a) Row k (b) Row m

Figure 5.19: Row k and row m of stiffness matrix A.

Let $G = (V, E)$ be the graph representing T. Each edge $e_k \in E$ is associated with a triangle $t_j \in T$. Since the edges have a one to one correspondence with triangle sides, we will refer to edge weights as triangle side weights from now on. The weight of a triangle side depends on the angle between the side and the direction of convection.

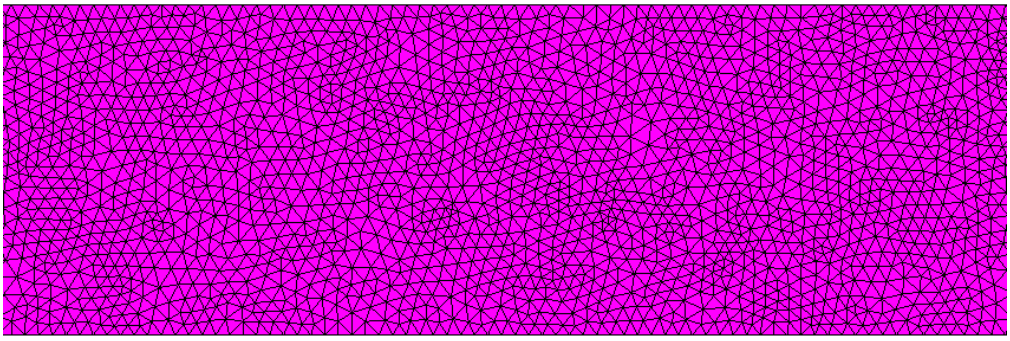


Figure 5.20: A portion of Ω with Triangularization T.

Triangles in our mesh have sides oriented at all different angles as seen in Figure 5.20. We will calculate the two extreme orientations. First consider a triangle side that is parallel to convection as pictured in Figure 5.18a. From the

Stiffness Matrix Weighting formula (5.8) and the stencil in Figure 5.19a

$$\begin{aligned} w(s_i) &= \max\left\{\frac{(|A_{i,i+1}|+|A_{i,i+2}|)}{2|A_{i,i}|}, \frac{|A_{i+1,i}|}{2|A_{i+1,i+1}|}+\frac{|A_{i+2,i}|}{2|A_{i+2,i+2}|}\right\} \\ &\quad + \max\left\{\frac{(|A_{i+3,i+1}|+|A_{i+3,i+2}|)}{2|A_{i+3,i+3}|}, \frac{|A_{i+1,i+3}|}{2|A_{i+1,i+1}|}+\frac{|A_{i+2,i+3}|}{2|A_{i+2,i+2}|}\right\} \\ &= \max\frac{1}{2\sqrt{3}(\alpha+1)}\left\{\left(\left|-\frac{1}{\sqrt{3}}\right|+\left|-\frac{1}{\sqrt{3}}\right|\right), \left|-\frac{1}{\sqrt{3}}\right|+\left|-\frac{1}{\sqrt{3}}\right|\right\} \\ &\quad + \max\frac{1}{2\sqrt{3}(\alpha+1)}\left\{\left(\left|-\frac{1}{\sqrt{3}}\right|+\left|-\frac{1}{\sqrt{3}}\right|\right), \left|-\frac{1}{\sqrt{3}}\right|+\left|-\frac{1}{\sqrt{3}}\right|\right\} \\ &= \frac{2}{3(\alpha+1)} \end{aligned} \tag{5.53}$$

Second, consider a triangle side that is perpendicular to convection as pictured in Figure 5.18b. From the Stiffness Matrix Weighting formula (5.8) and the stencil in Figure 5.19b.

$$\begin{aligned} w(s_j) &= \max\left\{\frac{(|A_{j+1,j}|+|A_{j+1,j+3}|)}{2|A_{j+1,j+1}|}, \frac{|A_{j,j+1}|}{2|A_{j,j}|}+\frac{|A_{j+3,j+1}|}{2|A_{j+3,j+3}|}\right\} \\ &\quad + \max\left\{\frac{(|A_{j+2,j}|+|A_{j+2,j+3}|)}{2|A_{j+2,j+2}|}, \frac{|A_{j,j+2}|}{2|A_{j,j}|}+\frac{|A_{j+3,j+2}|}{2|A_{j+3,j+3}|}\right\} \\ &= \max\frac{1}{2\sqrt{3}(\alpha+1)}\left\{\left(\left|-\frac{\alpha}{\sqrt{3}}\right|+\left|-\frac{\alpha}{\sqrt{3}}\right|\right), \left|-\frac{\alpha}{\sqrt{3}}\right|+\left|-\frac{\alpha}{\sqrt{3}}\right|\right\} \\ &\quad + \max\frac{1}{2\sqrt{3}(\alpha+1)}\left\{\left(\left|-\frac{\alpha}{\sqrt{3}}\right|+\left|-\frac{\alpha}{\sqrt{3}}\right|\right), \left|-\frac{\alpha}{\sqrt{3}}\right|+\left|-\frac{\alpha}{\sqrt{3}}\right|\right\} \\ &= \frac{2\alpha}{3(\alpha+1)} \end{aligned} \tag{5.54}$$

These calculations tell us that triangle sides that are perpendicular to convection will be weighted equal to $\frac{2\alpha}{3(\alpha+1)}$ and triangle sides that are parallel to convection will be weighted equal to $\frac{2}{3(\alpha+1)}$. Sides oriented at in-between angles will have values in-between as shown in Figure 5.21. To minimize edge cut, the partition will have parts shaped like rectangles with aspect ratio $\alpha:1$ when $\alpha \leq 4$ and 4:1 when $\alpha > 4$. Aspect ratio is approximated by dividing the average weight of a side with orientation angle $\in [0.4\pi, 0.5\pi]$ with the average weight of a side $\in [0, 0.1\pi]$. The rectangles point in the x axis direction. Example partitions are displayed in Figure 9.24.

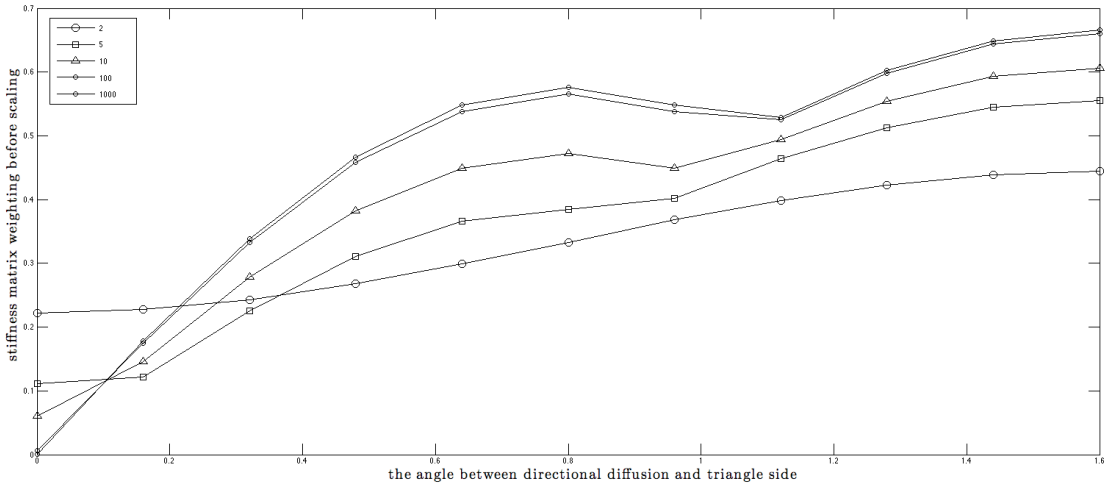


Figure 5.21: Stiffness matrix weighting for anisotropic diffusion. Each line represents a different λ_1/λ_2.

Chapter 6

A Posteriori Error Estimation

Let's begin this chapter with an example. We will solve a second order elliptic PDE on the unit line in $\mathbb{R}^1$ using two different approximations to illustrate a point.

Example 6.0.3. For $\Omega = [0, 1] \in \mathbb{R}^1$, find $u \in H^2(\Omega)$ such that $Lu = f$ on Ω and $u = 0$ on $\partial\Omega$ for some continuous and coercive elliptic differential operator L which produces the exact solution pictured in Figure 6.1.

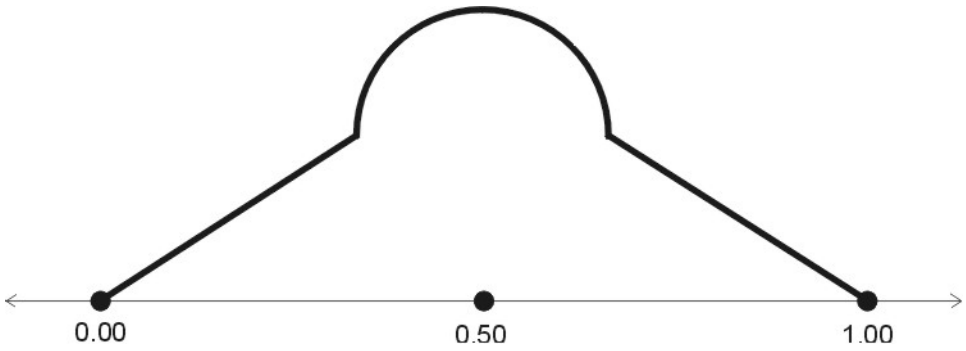
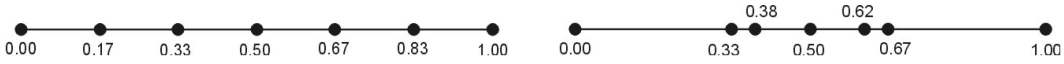


Figure 6.1: Exact solution u.

We would like to solve for u, but instead of finding u, we will find u_h, a discrete approximation using Finite Elements. A finite space of piecewise linear functions of dimension 5 will be used, therefore we divide our domain interval into six elements (subintervals) and use standard linear Lagrange basis functions.

There are an infinite number of choices for element placement. For illustration purposes, we will solve this problem twice using two different meshes pictured in Figure 6.2.



(a) mesh 1 (b) mesh 2

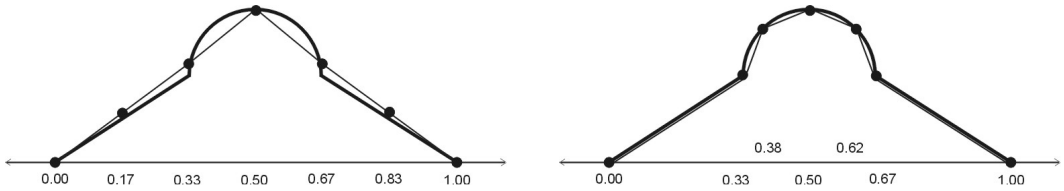
Figure 6.2: Two meshes for the unit line [0,1]

Mesh 1 with linear Lagrange basis functions defines a finite function space, $S_h^{(1)}$ while mesh 2 defines a space, $S_h^{(2)}$. The finite element solution $u_h^{(k)} \in S_h^{(k)}$ for $k = 1, 2$ satisfies Céa's Lemma [21]

$$||u - u_h^{(k)}|| \leq C \inf_{w \in S_h^{(k)}} ||u - w|| \tag{6.1}$$

therefore in general if $S_h^{(1)} \neq S_h^{(2)}$ then $u_h^{(1)} \neq u_h^{(2)}$. This means that the placement of our degrees of freedom will affect our finite element solution and its accuracy.

After solving the associated system of linear equations, we find the finite element solutions $u_h^{(1)}$ and $u_h^{(2)}$. They are different with different errors. These piecewise linear solutions are pictured in Figure 6.3 superimposed on the exact solution.



(a) mesh 1 (b) mesh 2

Figure 6.3: Two Finite Element solutions using different meshes.

Notice that mesh 2 produced a more accurate approximation to u. How could we have known to use mesh 2 instead of mesh 1 from the start? During our solution process, if we can determine where the error of $||u - u_h||$ would be the greatest, then we can position our degrees of freedom there and reduce the final error. This is the essence of a posteriori error estimation and non-uniform meshes.

The reason mesh 1 had difficulty approximating u was because u was curved in the interval $[0.33, 0.67]$. The piecewise linear functions had difficulty approximating this curve. (Piecewise polynomials of degree n cannot interpolate curves with non-zero $n+1$ derivatives without error.) If we have an approximation of the exact solution u, we can place our linear basis function degrees of freedoms in regions where u has the most curvature. (Or in general, place our degree n basis function degrees of freedom in regions where u has the greatest $n+1$ derivative.). This is the basis of the following a posteriori error Estimator.

6.1 An Posteriori Error Estimator

For linear finite elements solving problems in $\mathbb{R}^2$, Randy Bank and Jinchao Xu developed an asymptotically exact estimate of $||\nabla(u-u_h)||_{L^2(\Omega)}$ in [16] [17] which is based on a superconvergent approximation of the order 2 derivatives of u. [9] [16] [17]

$||\nabla(u-u_h)||_{L^2(\Omega)}$ is approximated by $||\nabla(u_2-u_1)||_{L^2(\Omega)}$ where u_k is the usual Lagrange interpolant. On a given triangle element, u_1 will interpolate at the 3 vertices and u_2 will interpolate at the 3 vertices and 3 side midpoints. Denote the 3 sides as $k=1,2,3$. Let l_k be the length of side k and let $\boldsymbol{t}_k$ be side k's unit tangent. Let q_k be a quadratic function equal to 1 at the midpoint of side k and equal to 0 at the 3 vertices and other 2 midpoints. Then

$$u_2 - u_1 = \sum_{k=1}^{3} l_k^2 \boldsymbol{t}_k^T M_t \boldsymbol{t}_k q_k(x,y) \tag{6.2}$$

$$\text{where } M_t = -\frac{1}{2}\begin{bmatrix} \partial_{xx}u_2 & \partial_{xy}u_2 \\ \partial_{yx}u_2 & \partial_{yy}u_2 \end{bmatrix} \tag{6.3}$$

The only values that are unknown in (6.2)-(6.3) are the elements of the Hessian. We approximate these from the recovered gradient of u_h. The function u_h is piecewise linear therefore before we can differentiate ∇u, we must smooth the gradient out. This is done with the operators Q_h and S_m. Q_h is the L_2 projection from the space of discontinuous piecewise constant functions into the space of continuous piecewise linear polynomials. S_m is a smoothing operator based on the

discrete Laplace operator. We approximate M_t with

$$\bar{M_t} = \frac{\alpha_t}{2}(\tilde{M_t} + \tilde{M_t}^T) \tag{6.4}$$

$$\tilde{M_t} = -\frac{1}{2}\begin{bmatrix} \partial_x S^m Q_h \partial_x u_h & \partial_x S^m Q_h \partial_y u_h \\ \partial_y S^m Q_h \partial_x u_h & \partial_y S^m Q_h \partial_y u_h \end{bmatrix} \tag{6.5}$$

Now our error is

$$e_t = \sum_{k=1}^{3} l_k^2 \boldsymbol{t}_k^T \bar{M_t} \boldsymbol{t}_k q_k(x, y) \tag{6.6}$$

And we can use this to approximate our finite element solution's error

$$||u - u_h||_{L^2(\Omega)} \approx ||e_t||_{L^2(\Omega)} \tag{6.7}$$

$$||\nabla(u - u_h)||_{L^2(\Omega)} \approx ||\nabla e_t||_{L^2(\Omega)} \tag{6.8}$$

Chapter 7

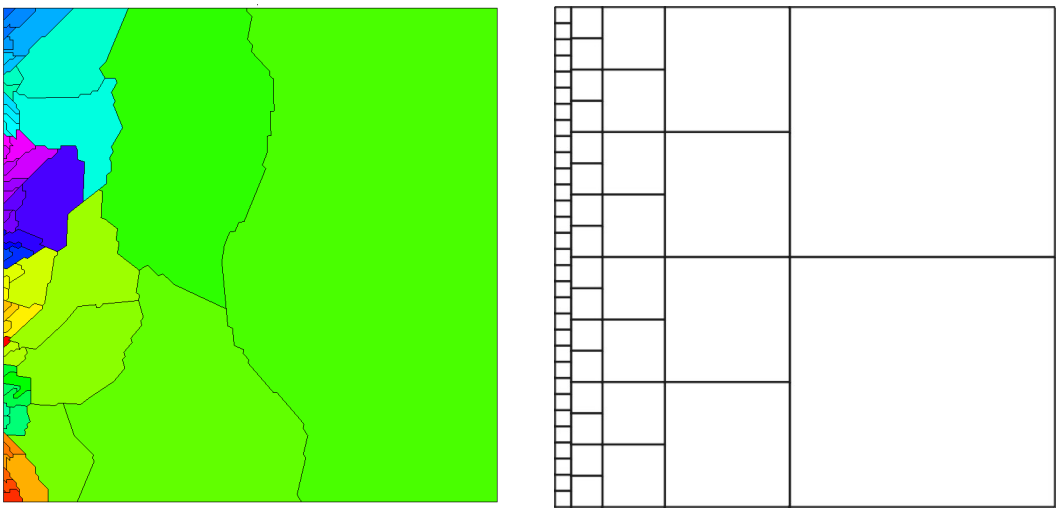
Vertex Weighting Schemes

In Chapter 5, our weighting schemes partitioned domains into parts of equal area and equal numbers of unknowns. In discrete terms, we worked with uniform triangularizations and partitioned their representative graphs with vertex weights equal to 1.

Since we are using triangle Finite Elements with linear Lagrange basis functions, each vertex of G represents one Finite Element with three degrees of freedom. Naturally, we would like each processor to be responsible for an equal number of unknowns. Since we would like to balance the workload, weighting each vertex equally is reasonable.

However, if we allow each processor the ability to refine their meshes after partitioning, then regardless of how many unknowns a processor initially receives, it can subsequently refine its mesh uniformly until it has the same target number of unknowns as all the other processors. This allows us to consider partitions of disproportionate area or disproportionate original numbers of unknowns while maintaining work balance between processors during the DD Method.

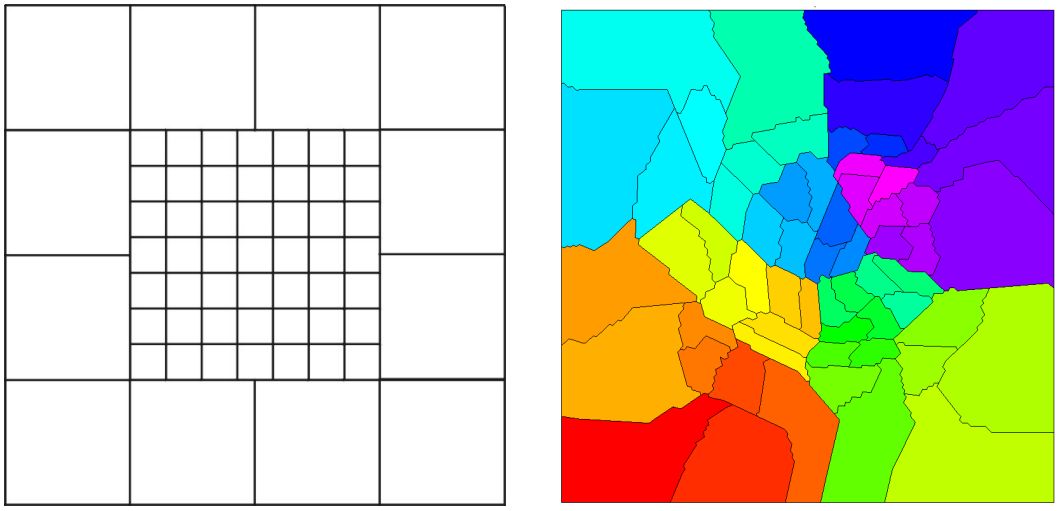
In Chapter 5, we saw that setting the edge weights in M_G to values other than 1 affected the interface length by deviating the shape of partition parts away from the optimal circle shape, and this modification did not affect the global mesh nor consequently the accuracy of the global finite element solution u_h. What are the consequences of setting the vertex weights to values other than 1?



(a) partition 1 (b) partition 2

Figure 7.1: Two partitions of the unit square having parts with disproportionate areas.

Figure 7.1 shows two partitions of the unit square whose 61 parts have disproportionate area. These partitions were created by weighting the area of the unit square by the function $D_V(x, y) = 1/(x + \epsilon)^2$ and then balancing this new weighted area among parts. These partitions of $\Omega = [0, 1] \times [0, 1]$ have the property that $\int_{\Omega_k} D_V dA$ for $k = 1, 2, ..., 61$ is constant and interface length is minimized. Partition 1 was done by METIS on a uniform mesh of 40000 triangles from PLTMG. Partition 2 was done theoretically by hand.



(a) partition 4 (b) partition 3

Figure 7.2: Two partitions of the unit square having parts with disproportionate areas.

Figure 7.2 shows two more partitions of the unit square whose 61 parts have disproportionate area. These partitions were created by weighting the area by the function $D_V(x, y) = 12.25$ for $(x, y) \in [0.25, 0.75] \times [0.25, 0.75]$ and $D_V(x, y) = 1$ otherwise. Partition 4 was done by hand and partition 3 was done by METIS using a triangularization from PLTMG.

After partitioning, we need to refine each part to an equal number of unknowns to balance the work load. Imagine that each get refined to 10^6. Then each global mesh created by combining the 61 parts would have 6.1×10^7 unknowns and all the global meshes from the four different partitions pictured in Figures 7.1 and 7.2 would each have the same number of unknowns but they would be placed in different locations. Global meshes 3 and 4 would have the majority of their unknowns in the middle of the unit square while global meshes 1 and 2 would have the majority of their unknowns near the y axis (left side in the figures).

This will affect the accuracy of the respective finite element solutions $u_h^{(k)}$ for $k = 1, 2, 3, 4$ as we saw in Example 6.0.3. Therefore when considering to weight the area in the continuous partitioning problem or analogously weighting the vertices in the graph partitioning problem, the effect on the accuracy of the final solution

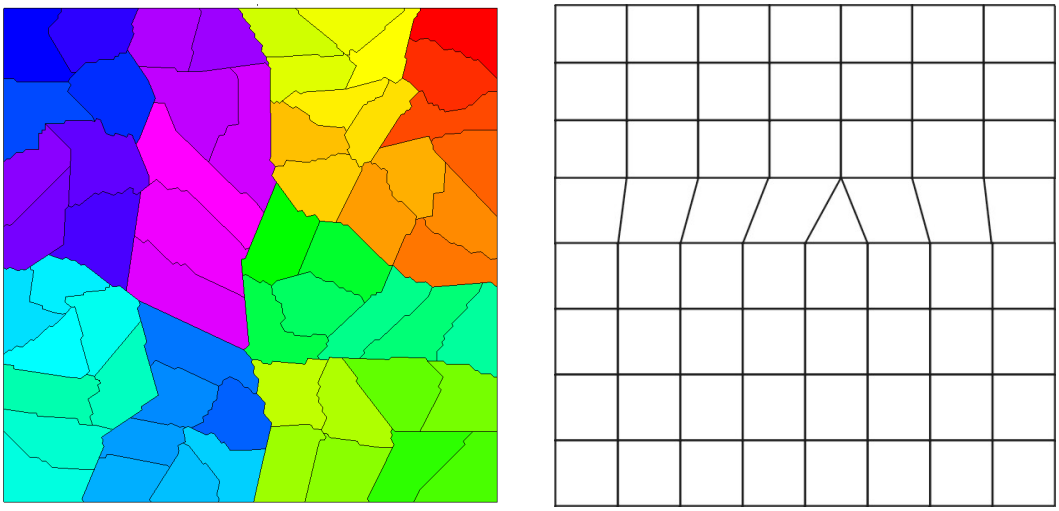
needs to be considered.

The next question is, how does vertex weighting affect edge cut? How does area weighting affect interface length? In order to discuss this, we need to introduce new terminology.

Definition 7.0.1. Given a triangularization $T = \{t_1, ..., t_m\}$ of domain Ω, define $\tilde{K}$ as a partition of T into P parts, $T_1, ..., T_P$ where each T_k is a subset of triangles T and disjoint.

Definition 7.0.2. Given a triangularization T of a domain Ω with partition $\tilde{K}$ into P parts, define *element cut*, denoted $\delta_t(\tilde{K})$, as the number of triangle sides on the interface. $\delta_t(\tilde{K}) = 0.5 \sum_{i=1}^{P} e(T_i) - 0.5e(T)$ where $e(T_k)$ is the number of triangle sides on the boundary of part T_k and $e(T)$ is the number of triangle sides on the boundary of Ω.

Lemma 7.0.3. *Let T be a triangularization and G be its representative graph. Let $\tilde{K}$ and K_ϵ be partitions of each respectively. If each edge of G is weighted 1, then $\delta(K_\epsilon) = \delta_t(\tilde{K})$, edge cut equals element cut.*



(a) partition 5 (b) partition 6

Figure 7.3: Two partitions of the unit square into equal area parts.

Immediately after partitioning the graph representation of our mesh using unequal vertex weights, Table 7.1 summarizes the interface length, edge cut, and

element cut for the four partitions in Figures 7.1 and 7.2 plus the two equal area partitions in Figure 7.3. Partition 5 is METIS partitioning 40000 uniform triangles from PLTMG into 61 equal area parts and partition 6 is a theoretical optimal partitioning of the unit square in 61 parts.

Table 7.1: Interface characteristics immediately after partitioning.

Partition	1	3	5	2	4	6
interface length	7.3	14.1	16.0	9.06	11	13.6
edge cut	938	1727	2037	1192	1447	1790
element cut	938	1727	2037	1192	1447	1790

From Table 7.1, it appears like vertex weighting (partitions 1-4) lowered the element cut versus not weighting (partitions 5-6). However, immediately after partitioning, we are not ready to begin our DD Method. First we must refine each partition to an equal number of degrees of freedom. We started with a uniform mesh of 40000 triangles. After partitioning, let's refine/unrefine each processor's region into $40000/61 \approx 656$ uniform triangles. Table 7.2 summarizes the interface length and element cut after this operation. Edge cut is no longer meaningful since we don't have a graph that represents the new mesh.

Table 7.2: Interface characteristics immediately after refine/unrefine

Partition	1	3	5	2	4	6
interface length	7.3	14.1	16.0	9.06	11	13.6
element cut	2049	2077	2037	1693	1921	1790

As you can see, unequal vertex weighting and then refining each part to have the same number of unknowns does not significantly affect the element cut even though it does affect the interface length. This is what we would expect. Vertex weighting affects the size of each partition but not their shape. Graph partitioning algorithms will still try to make parts shaped like circles or squares because these optimal shapes place the least number of element sides on a given part's boundary.

Even though partitions 2,4, and 6 have parts of differing sizes, all partitions use 61 square parts and each part contains 656 elements. The elements are roughly

uniform equilateral triangles, so each square part will place $\sqrt{656\sqrt{3}/4} \approx 16.85$ triangle sides on each square part's side regardless of the part's size. Therefore each partition will have $\approx (16.85 \cdot 61 \cdot 4 - B)/2$ element cut where B is the number of triangle sides on the boundary of the unit square.

Theorem 7.0.4. *Let Ω be a domain with triangularization T and representative graph G. The area of Ω equals A and the perimeter equals S. Let $K_\epsilon^{(1)}$ and $K_\epsilon^{(2)}$ be two different partitions of G into P parts using two different vertex weighting schemes. In both cases, let all the edge weights equal 1. Let $\tilde{K}_1$ and $\tilde{K}_2$ be $K_\epsilon^{(1)}$'s and $K_\epsilon^{(2)}$'s coinciding partitions of T into parts $T_1, ..., T_P$. Refine/unrefine T_k for $k = 1, ..., P$ such that $|T_k| = m \, \forall k$. Let B_i denote the number of triangle sides on the boundary of $\tilde{K}_i$. When $B_1 \approx B_2$ or $2.5P\sqrt{m} \gg B_i$ for $i = 1, 2$ then $\delta_t(\tilde{K}_1) \approx \delta_t(\tilde{K}_2)$.*

Proof. Since $K_\epsilon^{(1)}$ and $K_\epsilon^{(2)}$ are partitions from graphs with edge weights equal 1, their parts will be roughly shaped as squares in order to minimize edge cut. With a uniform equilateral triangle mesh, each square part will have roughly $\sqrt{m\sqrt{3}/4}$ triangle sides on their boundary. Each partition has P parts. By Lemma 5.0.4, the element cut of each partition is

$$\delta_t\left(\tilde{K}_i\right) = P\sqrt{m\sqrt{3}} - 0.5B_i \tag{7.1}$$

where B_i is the number of triangle sides on the boundary of Ω of $\tilde{K}_i$. □

Corollary 7.0.5. *Use the same setup as Theorem 7.0.4. When $B_1 \approx B_2$ or $2.5P\sqrt{m} \gg B_i$ for $i = 1, 2$, then using vertex weighting compared to not using vertex weighting does not significantly affect the element cut.*

Proof. Apply Theorem 7.0.4 and let $\tilde{K}_\epsilon^{(2)}$ be a weighted partition with $w(v_k) = 1$ for all k (which is like an unweighted partition). □

Corollary 7.0.6. *Using the same setup as Theorem 7.0.4, $\delta_t(\tilde{K}_i) < \delta_t(\tilde{K}_j)$ iff $B_i > B_j$.*

Proof. This result follow from (7.1) □

In light of Theorem 7.0.4 which says $\delta_t(\tilde{K}_1) \approx \delta_t(\tilde{K}_2)$, Corollary 7.0.6 points out that vertex weighting can be used to slightly reduce (or increase) element cut. An example is given by partitions 2 and 6 pictured in Figures 7.1 and 7.3. Table 7.2 shows that partition 2 has less element cut than the unweighted partition 6 because partition 2 placed a significant number of its triangle sides on the original domain's boundary.

Using vertex weights other than 1 is the basis of the following two Weighting Schemes; Error Weighting and Flow Weighting.

7.1 Error Weighting

In Chapter 6, Example 6.0.3 demonstrated that the placement of degrees of freedom associated with the solution u_h of (1.1)-(1.3) affects its accuracy $||e_h||_{L^2(\Omega)} = ||u - u_h||_{L^2(\Omega)}$. Roughly speaking, when approximating the solution u of (1.1)-(1.3) with with piecewise linear basis functions, regions where the function u is curved will contain more error than regions where the function is flat. (See Chapter 6 for a more rigorous discussion.) In order to minimize the overall error, we need to place more triangles in these troublesome regions than other regions (because the degrees of freedom are at each triangle vertex). This is the essence of the Error Weighting scheme.

Definition 7.1.1. *Error Weighting* for the solution of (1.1)-(1.3). Let T be a triangularization of Ω and $G = (V, E)$ its representative graph. Each vertex v_i corresponds with triangle t_i. Define the weight matrix W_G as

$$w(v_i) = \alpha||\nabla(u - u_h)||_{L^2(t_i)} + \beta \tag{7.2}$$

where α and β are scaling constants chosen based on the graph partitioner being used.

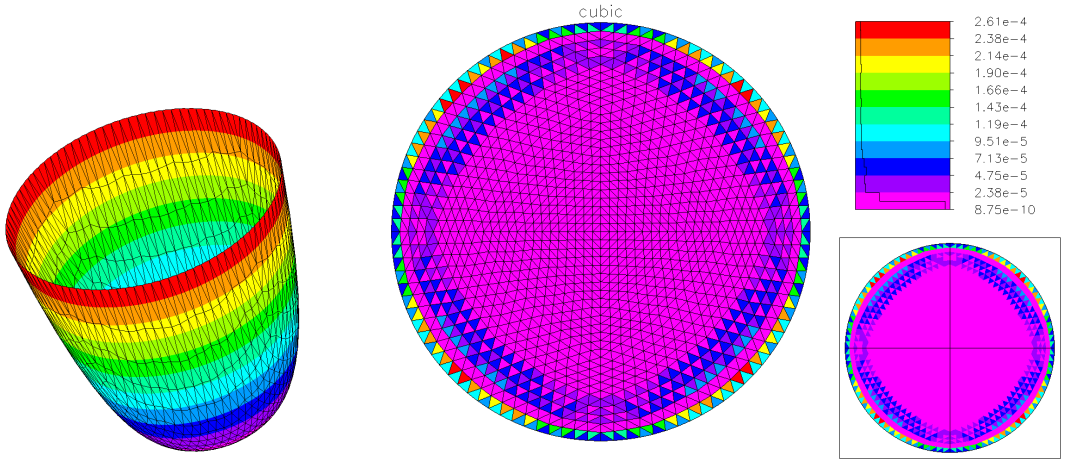
After the Error Weighting scheme partitions a uniform mesh, each part most likely will contain a different number of elements (triangles) which means that each processor will receive a different number of degrees of freedom. At first, this appears that the work is unbalanced. However, as part of the Error

Weighting scheme, it is understood that after partitioning, each processor will refine its subdomain to an agreed upon number of degrees of freedom. Then all processors will have equal work. From the perspective of the global mesh, this accomplishes placing the degrees of freedom where they are most needed to reduce the overall error.

Example 7.1.2. Consider the PDE; find $u \in H^2(\Omega)$ where Ω is the unit circle in $\mathbb{R}^2$ such that

$$\begin{aligned} -\triangle u + 16x^2 + 16y^2 &= 0 \text{ on } \Omega \\ u &= 1 \text{ on } \partial\Omega \end{aligned} \tag{7.3}$$

The exact solution to this PDE is $u = (x^2 + y^2)^2$ pictured in Figure 7.4a.

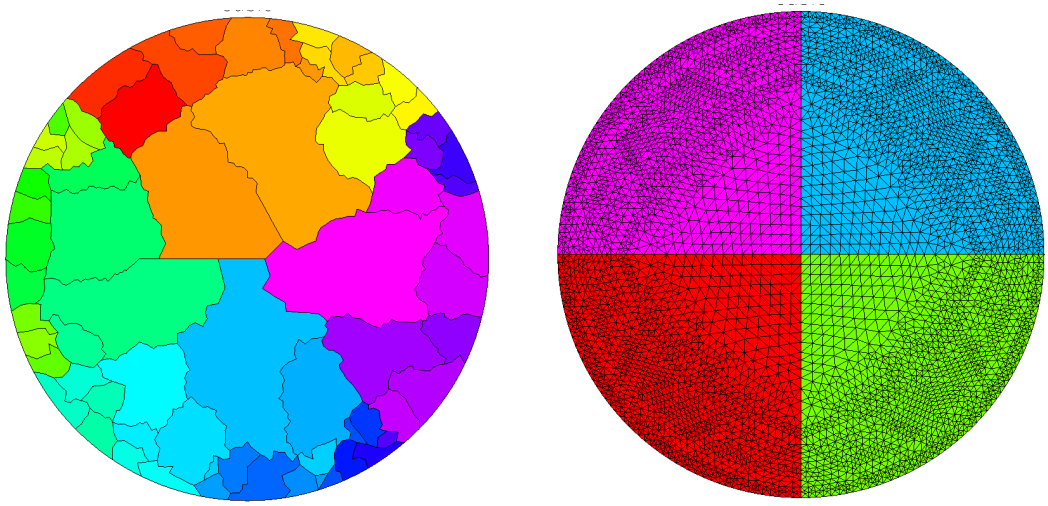


(a) $u = \left(x^2 + y^2\right)^2$ (b) Linear finite element error

Figure 7.4: Solving Poisson's Equation.

The solution u is flattest near the origin and most curved near the boundary. Figure 7.4b is PLTMG displaying an approximation of this error based on the current coarse solution u_k pictured in Figure 7.4a and the current mesh pictured in Figure 7.4.

If we use Error Weighting to partition this domain so we can solve it in parallel then the parts near the boundary will be smaller than the parts near the middle because Error Weighting will balance the error among the parts.



(a) Error Weighting Partition (b) Global mesh

Figure 7.5: Error Weighting placing degrees of freedom where they're needed.

Immediately after partitioning, the parts near the boundary contain less triangles and less degrees of freedom than the parts near the middle. But before we begin the Domain Decomposition iteration, each part will uniformly add more triangles until all the parts have the same amount of triangles. This will cause the triangles in the smaller parts to be more dense. When you combine all the local meshes together, it will no longer be overall uniform. The global mesh is pictured in Figure 7.5b. You can see that most of the degrees of freedom are near the boundary of the unit circle where they are needed. The finite element solution produced from this global mesh will more closely approximate u than a finite element solution from a uniform global mesh. (Figure 6.3 also demonstrated this in Example 6.0.3.)

This is how Error Weighting produces more accurate finite element solutions. Additionally in many cases, Error Weighting also speeds up convergence of the DD solve. This is explored in the numerical experiments of Chapter 9 and it is discussed in the next section.

Also, when using adaptive meshing after partitioning, Error Weighting prevents the need for time expensive load rebalancing. This is one of the innovations of the Bank-Holst paradigm and is explained in [5] [6].

7.2 Flow Weighting

Flow Weighting is motivated by the observation that Error Weighting speeds up convergence of the DD solve for certain problems, while at other times, Error Weighting doesn't affect convergence or may even slow down convergence. An analysis of this situation led us to understand the effects of Error Weighting so that we can mimic these effects at will.

Definition 7.2.1. *Flow Weighting* for the solution of (1.1)-(1.3). Let T be a triangularization of Ω and $G = (V, E)$ its representative graph. Each vertex v_i corresponds with triangle t_i. Define a function called *flow* $z(\cdot) : \Omega \to [0, 1] + \epsilon$ for some small $\epsilon > 0$. Define the weight matrix W_G as

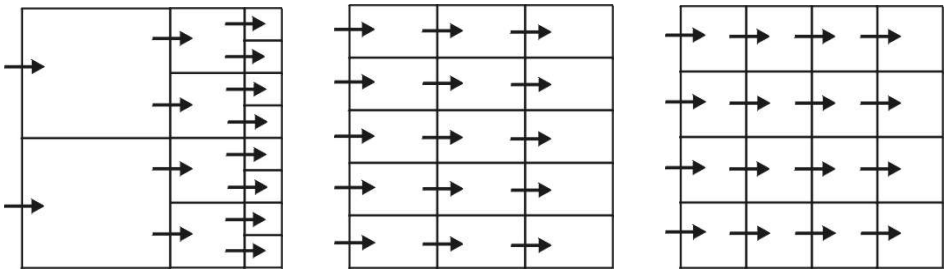
$$w(v_i) = \alpha \left(z \left(p \left(t_i \right) \right) \right)^{-s} + \beta \tag{7.4}$$

$$\text{where } p(t_i) = (x, y, z) \text{ center of triangle } t_i \tag{7.5}$$

and α and β are scaling constants chosen based on the graph partitioner being used. The variable s is called the *flow function parameter*.

The user supplies the flow function. An effective flow function is one that characterizes the direction of convection or anisotropic diffusion. In most cases, the best choice is to parameterize the domain as follows. Let $x_0 \in \mathbb{R}^2$ be a point furthest downwind of convection such that $x_0 \in \partial\Omega$. Then for $t \in [0, 1]$ parameterize a curve through Ω going upwind, $z_0(t) = x_0 - \hat{b}t$ where $\hat{b}$ is the unit vector in the direction of convection. Finally, for every point $x \in \Omega$, define $z(x) = z_0^{-1} \left(x_0 + \hat{b}\hat{b}^T (x - x_0) \right) + \epsilon$ for some small $\epsilon > 0$.

In Chapter 5, we saw that Convection Weighting encouraged the creation of rectangle parts in the direction of convection which facilitated information travel and Chapter 9 shows that it speeds up DD convergence. Similarily, Flow Weighting facilitates information travel and improves convergence.



(a) Flow Weighting (b) Convection Weighting (c) Unweighted

Figure 7.6: Information takes only 3 steps to travel across weighted partitions.

Figure 7.6 shows the unit square partitioned three different ways into about 15 parts each. Assume we are solving $-\triangle u + u_x - 1 = 0$ on this domain with dirichlet boundary conditions. Figure 7.6a depicts the partition of the unit square using Flow Weighting with $z(x, y) = 1.001 - x$ and parameters $s = 2$, $\alpha = 1$, $\beta = 0$. Figure 7.6b uses Convection Weighting and Figure 7.6c uses no weighting.

Both Flow Weighting and Convection Weighting facilitate the convection in the x direction. Information travels from the left dirichlet boundary condition through the domain in 3 hops for both schemes. It takes 4 hops for the information to travel across the unweighted partition. Numerical experiments show that both Flow and Convection Weighting improve the DD convergence of solving the convection-diffusion equation. See Chapter 9.

Care must be taken when using Flow Weighting. If the finite element solution to your PDE would have uniform error $||u - u_h||_{L^2(t_k)} \approx c \ \forall t_k$ on a global uniform mesh T, then Flow Weighting will deviate the global mesh from uniform and degrade the accuracy. The next theorem illuminates this.

Theorem 7.2.2. *Given a PDE whose solution has uniform error* $||u-u_h||_{L^2(t_k)} = c$ *for* $k = 1, ..., n$ *over domain* Ω *with uniform triangularization* T *into* $t_1, ..., t_n$*, let* $\tilde{K}^{(1)}$ *be a partition of* T *created from unweighting the representative graph and* $\tilde{K}^{(2)}$ *be a partition from using Flow Weighting. After partitioning, refine each part to the same number of unknowns. If* $u_h^{(i)}$ *is the final global finite element solution on* $\tilde{K}^{(i)}$ *and* $||e_h^{(i)}|| = ||u - u_h^{(i)}||_{L^2(\Omega)}$ *then as* $n \to \infty$

$$\frac{||e_h^{(2)}||}{||e_h^{(1)}||} = \int_\Omega z(x,y)^{-s}\, dxdy \int_\Omega z(x,y)^s\, dxdy \left(\int_\Omega 1 dxdy\right)^{-1} \tag{7.6}$$

Proof.

$$\frac{h^{(2)}(t_k)}{h^{(1)}(t_k)} = \sqrt{\frac{\int_{t_k} 1dxdy \,/\, \int_{\Omega} 1dxdy}{\int_{t_k} z(x,y)^{-s}\, dxdy \,/\, \int_{\Omega} z(x,y)^{-s}\, dxdy}} \tag{7.7}$$

For a finite element solution

$$||e_h^{(i)}||_{L^2(t_k)} \approx c_1 \left(h^{(i)}(t_k)\right)^2 ||u||_{H^2(t_k)} \tag{7.8}$$

therefore

$$\frac{||e^{(2)}(t_k)||}{||e^{(1)}(t_k)||} = \frac{\int_{t_k} 1dxdy \,/\, \int_{\Omega} 1dxdy}{\int_{t_k} z(x,y)^{-s}\, dxdy \,/\, \int_{\Omega} z(x,y)^{-s}\, dxdy} \tag{7.9}$$

and

$$\frac{||e_h^{(2)}||}{||e_h^{(1)}||} = \sum_{k=1}^{n} \left(\frac{||e^{(2)}(t_k)||}{||e^{(1)}(t_k)||} \int_{t_k} 1dxdy \right) \tag{7.10}$$

Substituting (7.9) into (7.10) and letting $n \to \infty$ completes the proof.

□

To illustrate the magnitude of the implication of Theorem 7.2.2, we provide an example. We solved (7.6) numerically with the flow function

$$z(x,y) = x + 10^{-3} \tag{7.11}$$

on the unit square (which produces partitions that look like Figure 7.1b). When solving a PDE on the unit square whose solution has uniform error on a uniform mesh $\left(||u - u_h||_{L^2(t_k)} \approx c \;\; \forall t_k \text{ with } k = 1, ..., n\right)$, then using Flow Weighting with flow function (7.11) increases your final solution's total error as compared to using an unweighted partition. Table 7.3 displays the error increase factor for different flow function parameters.

Table 7.3: Error increase factor on the unit square when using $z(x,y) = x + 10^{-3}$ with different flow function parameters s.

s	0	0.5	0.75	1.0	1.25	1.5	2.0
$\|\|e_h^{(2)}\|\|/\|\|e_h^{(1)}\|\|$	1.0	1.29	1.88	3.46	8.24	24.6	334

7.3 Interface Reconciliation

Theorem 7.0.4 proves that when $B_1 \approx B_2$ or $2.5P\sqrt{m} \gg B_i$ then $\delta_t(\tilde{K}_1) \approx \delta_t(\tilde{K}_2)$ for two different partitions created from two different vertex weighting schemes. In layman's terms, it stated that using vertex weighting does not affect element cut because regardless of part size, each part will still try to shape itself as a circle and if each part has m elements, they will each place the same number of those elements on their boundary regardless of part size. This assumes that the interface is not reconciled.

Some Domain Decomposition Methods work with mismatched grids on the interface. If part T_i and part T_j share the boundary Γ and part T_i has m unknowns on Γ and part T_j has n unknowns on Γ. Then when part T_i communicates its m unknowns to part T_j, part T_j interpolates those values onto its n boundary degrees of freedom before incorporating them into computation.

Other methods require that the grids match on the interface. Therefore after partitioning and after refining each part to have the same number of unknowns as each other part, another step may be required. Reconciliation is the process whereby when two parts share a boundary, part T_i refines its mesh to include part T_j's boundary degrees of freedom, and part T_j refines its mesh to include part i's boundary degrees of freedom. This is what the Bank-Holst Paradigm does. This of course increases the element cut.

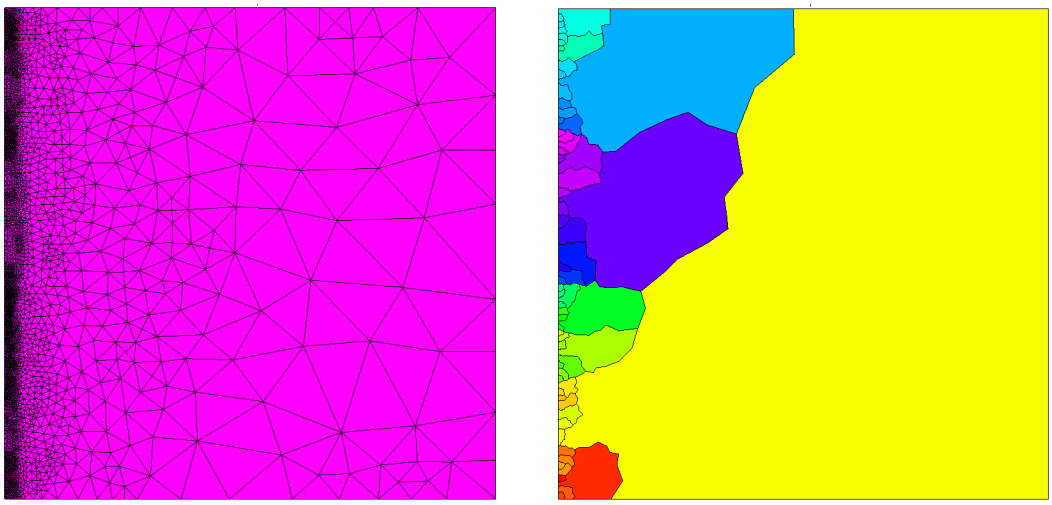
7.4 Adaptive meshing

So far, we've only discussed uniform meshing. All of the weighting schemes discussed in both this chapter and Chapter 5 acted on uniform partitions. After partitioning, each processor then refined its subdomain further using more uniform meshing. For Edge Weighting schemes, the global meshes remained uniform. In the case of Vertex Weighting schemes, this lead to global meshes that were not overall uniform but were uniform in each subdomain.

Adaptive meshing can be employed both before and/or after partitioning. Employment after partitioning is a subject for future study described in Section

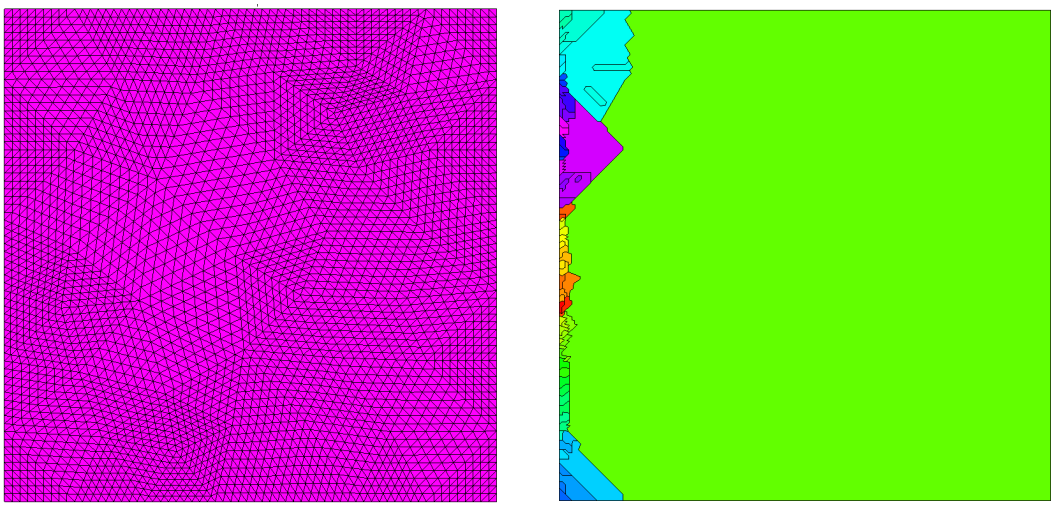
10.2. Adaptive meshing before partitioning was investigated during this research.

Approximately, partitioning an adaptive mesh with vertex weights equal 1 creates the same partition as partitioning a uniform mesh with vertex weighting matching the adaptive mesh's density.



(a) Adaptive mesh (b) Partition from adaptive mesh

Figure 7.7: Partitioning an adaptive mesh with vertex weight equal 1.



(a) Uniform mesh (b) Partition from uniform mesh

Figure 7.8: Partitioning a uniform mesh with vertex weights matching adaptive mesh.

The partition in Figure 7.8b was created by setting the vertex weights corresponding to each uniform triangle equal to the number of adaptive triangles it would cover if the uniform mesh was superimposed on top of the adaptive mesh. Formally $w(v_i) = \int_{t_i} D_v(x,y)\,dxdy$ where D_v is the density of adaptive triangles per area at position (x,y).

Adaptive meshing provides an alternative way of weighting vertices. Instead of weighting vertices in the graph partitioning algorithm, you can start with an adaptive mesh that contains the weighting information and then partition with the weights equal 1. From Figures 7.7 and 7.8, you see that this does a decent job. If your goal is to balance the parts very precisely, then adaptive meshing should be used because it does a better job.

Lemma 7.4.1. *Given a domain $\Omega \in \mathbb{R}^d$ and an area weighting function function $D_v : \Omega \to \mathbb{R}$. Let $T^{(1)}$ be an adaptive triangularization of Ω based on D_v into n elements and let $T^{(2)}$ be a uniform triangularization of Ω into n elements. Let n be a multiple of P. Let $\tilde{K}^{(1)}_{\epsilon_1}$ and $\tilde{K}^{(2)}_{\epsilon_2}$ be partitions of $T^{(1)}$ and $T^{(2)}$ respectively into P parts that minimize ϵ_i in*

$$\begin{aligned} &\text{for } i = 1,2 \\ &\text{for each } T^{(i)}_k \in \tilde{K}^{(i)}_{\epsilon_i} \quad \int_{T^{(i)}_k} D_v(x,y)\,dxdy = \frac{1+\delta_k}{P}\int_{\Omega} D_v(x,y)\,dxdy \qquad (7.12) \\ &\qquad\qquad \text{where } |\delta_k| < \epsilon_i \end{aligned}$$

then it follows that $|\epsilon_1| \leq |\epsilon_2|$.

Proof. If $T^{(1)}$ is a perfect adaptive mesh, then $\int_{t^{(1)}_k} D_v(x,y)\,dxdy = c$ for all $k = 1, ..., n$. Let each $T_k \in \tilde{K}^{(1)}_{\epsilon_i}$ contain n/P triangles, then $\epsilon_1 = 0$ and this guarantees that $|\epsilon_1| \leq |\epsilon_2|$. □

Chapter 8

Convergence Analysis

In Chapters 5 and 7, we presented new Weighting Schemes to be used with Graph Partitioning algorithms. Chapter 5 showed how to add weights to the edges of a representative graph and Chapter 7 explained how to add weights to the vertices. Each set of techniques can speed up the convergence of Domain Decomposition Methods.

Adding weight to the edges encourages partitioning algorithms to create partition parts shaped like rectangles instead of optimally shaped squares or circles. This increases element cut and consequently communication time and sometimes increases computation time. But, when rectangles are used appropriately, they speed up convergence which reduces the number of iterations needed to achieve one's desired level of accuracy and ultimately shortens the solve time.

Chapter 3 introduced many Domain Decomposition methods. We would like our Weighting Schemes to improve the solve time for all of them. In this chapter, we prove that the edge weighting schemes from Chapter 5 speed up the convergence of the Additive Schwarz and Multiplicative Schwarz Methods in simple cases. In Chapter 9, we provide numerical experiments that demonstrate faster convergence for the Bank-Holst Paradigm DD solver using these edge weighting schemes.

Adding weight to the vertices encourages partitioning algorithms to create partition parts of different sizes without deviating the parts from the optimal shape of squares or circles. This affects the final global mesh because it non-uniformly

distributes the degrees of freedom. When used appropriately, this redistribution can increase the accuracy of the final finite element solution and speed up convergence at the same time. At other times, the final accuracy cannot be improved but one can decide to decrease the solution accuracy slightly in exchange for faster convergence and a shorter solve time.

In Chapter 9, we provide numerical experiments that demonstrate faster convergence for the Bank-Holst Paradigm DD solver using these vertex weighting schemes.

8.1 Preconditioned Richardson Iteration

Richardson Iteration solves a system of linear questions $Ax = f$ by starting with an initial guess $x^{(0)}$ and creating a sequence of approximations that converge to the true solution x.

Algorithm 9 Preconditioned Richardson Iteration

Initialize $x^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

2: $\quad x^{(k)} = x^{(k-1)} + B(f - Ax^{(k-1)})$.

3: **end for**

Matrix B is called the *preconditioner* and is an approximation of A^{-1}.

Theorem 8.1.1. *Richardon's Method will converge iff the spectral radius* $\rho(I - BA) < 1$.

Proof. The error of the k^{th} iteration $e^{(k)} = x - x^{(k)}$.

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + B(f - Ax^{(k)}) \\ x - x^{(k+1)} &= x - x^{(k)} - B(Ax - Ax^{(k)}) \\ e^{(k+1)} &= (I - BA)e^{(k)} \end{aligned}$$

Therefore

$$e^{(k+1)} = (I - BA)^{(k+1)}e^{(0)} \tag{8.1}$$

and $e^{(k+1)} \to 0$ iff $\rho(I - BA) < 1$. □

Definition 8.1.2. Since $||e^{(k+1)}|| \leq ||(I-BA)||^{(k+1)}||e^{(0)}||$ from (8.1), the *asymptotic convergence rate* of preconditioned richardson iteration with precondition B is $\rho(I - BA)$.

8.2 Additive Schwarz

The Additive Schwarz Method was presented in Chapter 3 in Algorithm 2. It was presented in its continuous Strong Form. By following the procedures in Chapter 2, we can convert it into its Weak Form, then Galerkin Approximation, then Finite Element Form, and finally its Matrix Form.

Let's consider the case of partitioning Ω into two subdomains. Assume the subdomain meshes are matching in the overlapping region. Let $A_1U_1 = B_1$ and $A_2U_2 = B_2$ be the stiffness matrices, degrees of freedom, and load vectors for subdomains 1 and 2 respectively. And let $A_{1,2}$ be the interaction of subdomain 1 with the degrees of freedom on Γ_1 and $A_{2,1}$ be the interaction of subdomain 2 with Γ_2. See Figure 3.1a and (2.8) for clarification.

Then we want to solve the block system

$$\begin{bmatrix} A_1 & A_{1,2} \\ A_{2,1} & A_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \tag{8.2}$$

Algorithm 10 Finite Elements Additive Schwarz

Initialize $U_1^{(0)}$ and $U_2^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

Perform steps 2 and 3 simultaneously on different processors.

2: Solve $A_1U_1^k = B_1 - A_{1,2}U_2^{k-1}$ for U_1^k

3: Solve $A_2U_2^k = B_2 - A_{2,1}U_1^{k-1}$ for U_2^k

4: Communicate U_i from Γ_j to neighbor processors.

5: **end for**

This is equivalent to solving $AU = B$ with Richardson Iteration and a block

Jacobi preconditioner.

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^k = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ A_{2,1} & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^{k-1} - \begin{bmatrix} 0 & A_{1,2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^{k-1}$$

$$DU^k = (E+F)U^{k-1} + B$$

$$U^k = \left(D^{-1}(E+F)\right)U^{k-1} + D^{-1}B$$

where $A = D - E - F$ and D is block diagonal, E strictly block lower, and F strictly block upper.

8.3 Multiplicative Schwarz

The Multiplicative Schwarz Method was presented in Chapter 3 in Algorithm 1. It was presented in its continuous Strong Form. By following the procedures in Chapter 2, we can convert it into its Weak Form, then Galerkin Approximation, then Finite Element Form, and finally its Matrix Form.

Let's consider the case of partitioning Ω into two subdomains. Assume the subdomain meshes are matching in the overlapping region. Let $A_1U_1 = B_1$ and $A_2U_2 = B_2$ be the stiffness matrices, degrees of freedom, and load vectors for subdomains 1 and 2 respectively. And let $A_{1,2}$ be the interaction of subdomain 1 with the degrees of freedom on Γ_1 and $A_{2,1}$ be the interaction of subdomain 2 with Γ_2. See Figure 3.1a and (2.8) for clarification.

Then we want to solve the block system

$$\begin{bmatrix} A_1 & A_{1,2} \\ A_{2,1} & A_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \tag{8.3}$$

Algorithm 11 Finite Elements Multiplicative Schwarz

Initialize $U_2^{(0)}$ to an initial guess

1: **for** $k = 1, 2, ...$ until convergence **do**

2: Solve $A_1U_1^k = B_1 - A_{1,2}U_2^{k-1}$ for U_1^k

3: Solve $A_2U_2^k = B_2 - A_{2,1}U_1^k$ for U_2^k

4: **end for**

This is equivalent to solving $AU = B$ with Richardson Iteration and a block Gauss Seidel preconditioner.

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^k = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} - \begin{bmatrix} 0 & 0 \\ A_{2,1} & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^k - \begin{bmatrix} 0 & A_{1,2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^{k-1}$$

$$(D - E)U^k = FU^{k-1} + B$$

$$U^k = \left((D - E)^{-1}F\right) U^{k-1} + (D - E)^{-1}B$$

where $A = D - E - F$ and D is block diagonal, E strictly block lower, and F strictly block upper.

8.4 Edge Weighting Convergence

In this section, we will prove that both the Additive and Multiplicative Schwarz Domain Decomposition Methods converge faster when using the edge weighting schemes presented in Chapter 5 versus not using them when there is convection and/or diffusion in the direction of the rectangular parts of partitioning.

Theorem 8.4.1. *Let matrix A be toepliz block tridiagonal with $m \times m$ blocks and m even, whose off diagonal blocks are $m \times m$ diagonal matrices and whose diagonal blocks are $m \times m$ toepliz tridiagonal matrices.*

$$A = \begin{bmatrix} T & Y & & \\ X & \ddots & \ddots & \\ & \ddots & & Y \\ & & X & T \end{bmatrix}, \quad T = \begin{bmatrix} a & c & & \\ b & \ddots & \ddots & \\ & \ddots & & c \\ & & b & a \end{bmatrix}$$

$X = xI$, $Y = yI$, $bc > 0$, and $xy > 0$. Let D be a 2×2 block diagonal matrix whose diagonal blocks are all equal $\tilde{A}$ where $\tilde{A}$ is the upper left $\frac{m}{2} \times \frac{m}{2}$ block submatrix of A. Then the m^2 eigenvalues of $(I - D^{-1}A)$ are

$$\begin{aligned} \lambda &= 0, \pm \frac{U_{\frac{m}{2}-1}(\lambda_T)}{U_{\frac{m}{2}}(\lambda_T)} \\ \text{where } \lambda_T &= \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{k\pi}{m+1}\right) \text{ for } k = 1, ..., m \end{aligned} \tag{8.4}$$

The eigenvalue 0 is repeated m^2-2m times. $U_k(\cdot)$ is the k^{th} Chebyshev polynomial of the second kind.

Proof. Let $A = D - E - F$ where $-E$ is the lower left $\frac{m}{2} \times \frac{m}{2}$ block submatrix of A and $-F$ is the upper right $\frac{m}{2} \times \frac{m}{2}$ block submatrix of A. Then

$$I - D^{-1}A = D^{-1}(E+F) =$$

$$= \left[\begin{array}{c|c} \tilde{A}^{-1} & 0 \\ \hline 0 & \tilde{A}^{-1} \end{array}\right] \left[\begin{array}{cc|cc} 0 & & 0 & \\ & & -Y & \\ \hline & -X & & \\ 0 & & 0 & \end{array}\right]$$

$$= \left[\begin{array}{ccc|ccc} & & & -yM_{1,\frac{m}{2}} & & \\ & 0 & & \vdots & & 0 \\ & & & -yM_{\frac{m}{2},\frac{m}{2}} & & \\ \hline & -xM_{1,1} & & & & \\ 0 & \vdots & & & & 0 \\ & -xM_{\frac{m}{2},1} & & & & \end{array}\right] \tag{8.5}$$

where $[M]_{i,j}$ is the block of $[\tilde{A}]^{-1}$ in block row i block column j. $[M]_{i,j}$ is a matrix of size $m \times m$. According to [25] on page 9

$$[M]_{i,j} = \begin{cases} (-1)^{i+j} \frac{Y^{j-i}}{\left(\sqrt{YX}\right)^{j-i+1}} \frac{U_{i-1}(\tilde{T}) U_{\frac{m}{2}-j}(\tilde{T})}{U_{\frac{m}{2}}(\tilde{T})} & \text{if } i \leq j \\ (-1)^{i+j} \frac{X^{i-j}}{\left(\sqrt{YX}\right)^{i-j+1}} \frac{U_{j-1}(\tilde{T}) U_{\frac{m}{2}-i}(\tilde{T})}{U_{\frac{m}{2}}(\tilde{T})} & \text{if } i > j \end{cases} \tag{8.6}$$

where $\tilde{T} = T/(2\sqrt{YX})$ and $U_k(\cdot)$ is the k^{th} Chebyshev polynomial of the second kind. By symmetry, $[M]_{\frac{m}{2},\frac{m}{2}} = [M]_{1,1}$. Then by (8.6)

$$[M]_{1,1} = \frac{1}{\sqrt{xy}} \frac{U_{\frac{m}{2}-1}(\tilde{T})}{U_{\frac{m}{2}}(\tilde{T})} \tag{8.7}$$

The eigenvalues of $I - D^{-1}A$ are $m^2 - 2m$ zeros corresponding with the

$m^2 - 2m$ columns of zeros in $I - D^{-1}A$ and $2m$ non zeros.

$$\lambda(I - D^{-1}A) = 0,\ \lambda\left(\begin{bmatrix} 0 & -yM_{1,1} \\ -xM_{1,1} & 0 \end{bmatrix}\right) \tag{8.8}$$

$$\text{and } \lambda\left(\begin{bmatrix} 0 & -yM_{1,1} \\ -xM_{1,1} & 0 \end{bmatrix}\right) = \pm\sqrt{xy}\lambda(M_{1,1})$$

Since $U_k(\cdot)$ is a polynomial

$$\begin{aligned} \lambda\left(U_k(X)\right) &= \lambda\left(2^k \prod_{j=1}^{k} (X - r_j I)\right) \\ \text{where } r_j &= \cos\left(\frac{j\pi}{k+1}\right) \\ &= 2^k \prod_{j=1}^{k} \left(\lambda(X) - r_j\right) \\ &= U_k\left(\lambda(X)\right) \end{aligned} \tag{8.9}$$

Since $\tilde{T}$ is a tridiagonal toeplitz matrix,

$$\lambda\left(\tilde{T}\right) = \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{k\pi}{m+1}\right) \text{ for } k = 1, ..., m \tag{8.10}$$

Complete the proof by substituting (8.10) into (8.7) into (8.8).

□

Theorem 8.4.2. *Let matrix A and function U be define as in Theorem 8.4.1. If $b, c, x, y < 0$ and $a = -b - c - x - y$, then the spectral radius*

$$\rho(I - D^{-1}A) = \frac{U_{\frac{m}{2}-1}\left(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right)\right)}{U_{\frac{m}{2}}\left(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right)\right)} \tag{8.11}$$

Proof. From Theorem 8.4.1, we have

$$\begin{aligned} \lambda\left(I - D^{-1}A\right) &= 0, \pm\frac{U_{\frac{m}{2}-1}(\lambda_T)}{U_{\frac{m}{2}}(\lambda_T)} \\ \text{where } \lambda_T &= \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{k\pi}{m+1}\right) \text{ for } k = 1, ..., m \end{aligned} \tag{8.12}$$

Since $b, c, x, y < 0$ and $a = -b - c - x - y$ and $a > 0$, we have $\lambda_T > 1$.

$$\begin{aligned}
\lambda_T &= \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{k\pi}{m+1}\right) \text{ for } k = 1, ..., m \\
&\geq \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right) \\
&> \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}(-1) \\
&= \frac{a - 2\sqrt{bc}}{2\sqrt{xy}} \\
&\geq \frac{-x - y}{2\sqrt{xy}} \\
&\geq 1
\end{aligned} \tag{8.13}$$

Now since

$$\begin{aligned}
U_k(x) &= 2^k \prod_{j=1}^{k} (x - r_{k,j}) \\
\text{where } r_{k,j} &= \cos\left(\frac{j\pi}{k+1}\right)
\end{aligned} \tag{8.14}$$

we have

$$\frac{U_{\frac{m}{2}-1}(\lambda_T)}{U_{\frac{m}{2}}(\lambda_T)} = \frac{1}{2\left(\lambda_T - r_{\frac{m}{2},\frac{m}{2}}\right)} \prod_{j=1}^{\frac{m}{2}-1} \frac{\left(\lambda_T - r_{\frac{m}{2}-1,j}\right)}{\left(\lambda_T - r_{\frac{m}{2},j}\right)} \tag{8.15}$$

Since all $\lambda_T > 1$ and $r_{\frac{m}{2}-1,j} < r_{\frac{m}{2},j} < 1$ for $j \in \{1, ..., \frac{m}{2} - 1\}$ and $r_{\frac{m}{2},\frac{m}{2}} < 0$ we have

$$\begin{aligned}
&\frac{\left(\lambda_{T,i} - r_{\frac{m}{2}-1,j}\right)}{\left(\lambda_{T,i} - r_{\frac{m}{2},j}\right)} > \frac{\left(\lambda_{T,j} - r_{\frac{m}{2}-1,j}\right)}{\left(\lambda_{T,j} - r_{\frac{m}{2},j}\right)} \text{ for } \lambda_{T,i} < \lambda_{T,j} \\
\text{and } &\frac{1}{2\left(\lambda_{T,i} - r_{\frac{m}{2},\frac{m}{2}}\right)} > \frac{1}{2\left(\lambda_{T,j} - r_{\frac{m}{2},\frac{m}{2}}\right)} \text{ for } \lambda_{T,i} < \lambda_{T,j}
\end{aligned} \tag{8.16}$$

Therefore

$$\max\left|\lambda\left(I - D^{-1}A\right)\right| = \frac{U_{\frac{m}{2}-1}\left(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right)\right)}{U_{\frac{m}{2}}\left(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right)\right)} \tag{8.17}$$

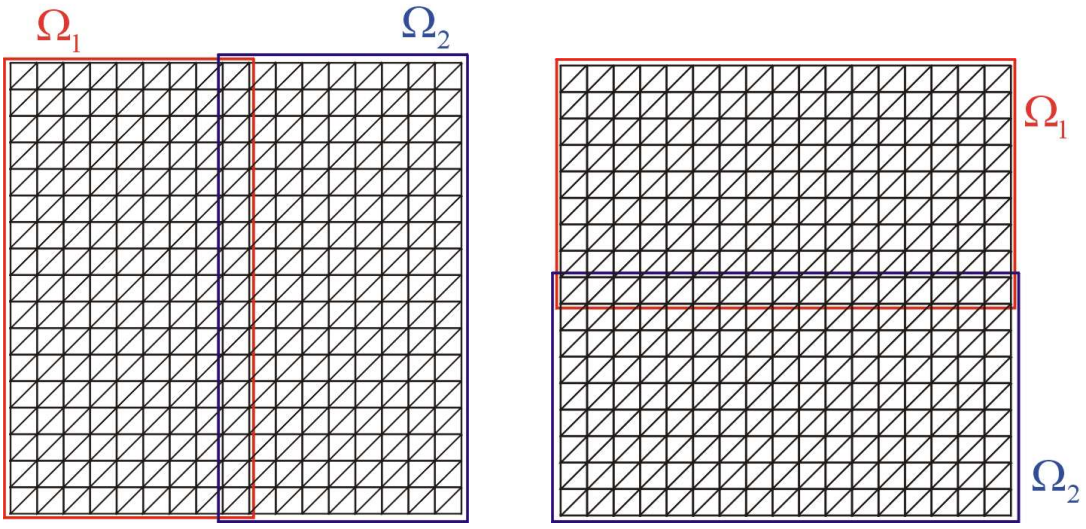
□

Theorem 8.4.3. *Let two partitions of the unit square* $\Omega = [0, 1] \times [0, 1] \in \mathbb{R}^2$ *be* $\bar{K}_0^{(1)} = \{[0, 0.5] \times [0, 1], [0.5, 1] \times [0, 1]\}$ *and* $\bar{K}_0^{(2)} = \{[0, 1] \times [0, 0.5], [0, 1] \times$

$[0.5, 1]\}$ *which are then enlarged slightly to overlap. When solving the boundary value problem, find* $u \in H^2(\Omega)$ *such that*

$$-\nabla \cdot \begin{bmatrix} \beta & 0 \\ 0 & 1 \end{bmatrix} \nabla u - 1 = 0 \text{ on } \Omega$$
$$u = 0 \text{ on } \partial\Omega \tag{8.18}$$

with uniform right triangle Finite Elements side length h *and Additive Schwarz Domain Decomposition, if* $\beta > 1$ *then the asymptotic convergent rate of using* $\bar{K}_0^{(2)}$ *will be less than using* $\bar{K}_0^{(1)}$*. Enlarge the parts of each partition to include one layer of Finite Element overlap as in Figure 8.1.*



(a) Partition $\bar{K}_0^{(1)}$ (b) Partition $\bar{K}_0^{(2)}$

Figure 8.1: Different partitions of the unit square.

Proof. First we will compute the global stiffness matrix for the entire mesh on Ω as in (2.9). Let $m = 1/h$

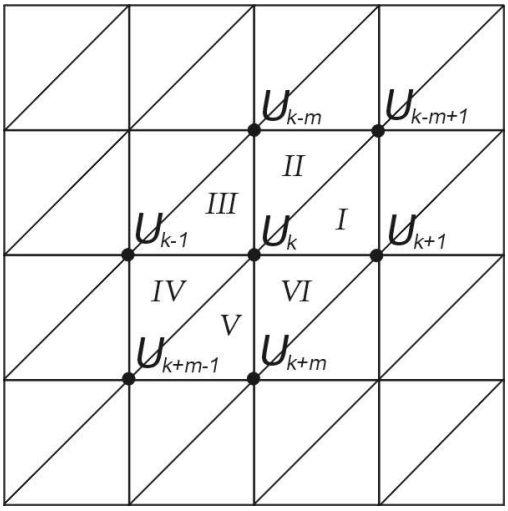


Figure 8.2: A portion of the mesh pictured in Figure 8.1

The k^{th} row of A when $(k \mod m) > 1$ is

$$\begin{aligned}
[A]_{k,k} &= \int_\Omega \nabla v_k \cdot \nabla v_k \, dxdy \qquad (8.19)\\
&= \sum_{i=1}^{6} \int_i \nabla v_k \cdot \nabla v_k \, dxdy \\
&= (2+2\beta)\frac{1}{h^2} \\
[A]_{k,k+m-1} &= [A]_{k,k-m+1} \qquad (8.20)\\
&= \int_I \nabla v_{k-m+1} \cdot \nabla v_k \, dxdy + \int_{II} \nabla v_{k-m+1} \cdot \nabla v_k \, dxdy \\
&= 0 \\
[A]_{k-1} &= [A]_{k+1} \qquad (8.21)\\
&= \int_I \nabla v_{k+1} \cdot \nabla v_k \, dxdy + \int_{VI} \nabla v_{k+1} \cdot \nabla v_k \, dxdy \\
&= -\beta\frac{1}{h^2} \\
[A]_{k,k-m} &= [A]_{k+m} \qquad (8.22)\\
&= \int_V \nabla v_{k+1} \cdot \nabla v_k \, dxdy + \int_{IV} \nabla v_{k+1} \cdot \nabla v_k \, dxdy \\
&= -1\frac{1}{h^2}
\end{aligned}$$

The k^{th} row of A when $(k \mod m) = 1$ has $[A]_{k,k-1} = 0$ and when $(k \mod m) = 0$ has $[A]_{k,k+1} = 0$ with the other values being the same as above. For the k^{th} row

of A when $m^2 - m \geq k > m$ ignore the elements defined above where either index $i < 1$ or $j < 1$ in $[A]_{i,j}$. Therefore A is a $m \times m$ block matrix

$$A = \frac{1}{h^2}\begin{bmatrix} T & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & & -I \\ & & -I & T \end{bmatrix}, \quad T = \begin{bmatrix} 2+2\beta & -\beta & & \\ -\beta & \ddots & \ddots & \\ & \ddots & & -\beta \\ & & -\beta & 2+2\beta \end{bmatrix} \tag{8.23}$$

T is a $m \times m$ matrix. When we use partition $\bar{K}_0^{(1)}$ and rearrange unknowns, then using Additive Schwarz corresponds to solving $AU = F$ with Richardson's Method and a block Jacobi preconditioner (see section 8.2). The iteration matrix is $I - D^{-1}A$ as defined in theorem 8.4.1 with

$$\text{for } \bar{K}_0^{(1)} \quad b, c, x, y = -1/h^2, -1/h^2, -\beta/h^2, -\beta/h^2 \tag{8.24}$$

and using partition $\bar{K}_0^{(2)}$ corresponds with

$$\text{for } \bar{K}_0^{(2)} \quad b, c, x, y = -\beta/h^2, -\beta/h^2, -1/h^2, -1/h^2 \tag{8.25}$$

Theorem 8.4.6 tells us the asymptotic convergence rate for each partition by (8.11). It involves the quantity $\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{m\pi}{m+1}\right)$.

$$\begin{aligned} \text{for } \bar{K}_0^{(1)} \quad \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{m\pi}{m+1}\right) &= \frac{2+2\beta}{2\beta} + \frac{1}{\beta}\cos\left(\frac{m\pi}{m+1}\right) \\ &= 1 + \frac{1}{\beta}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \end{aligned} \tag{8.26}$$

$$\begin{aligned} \text{for } \bar{K}_0^{(2)} \quad \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{m\pi}{m+1}\right) &= \frac{2+2\beta}{2} + \beta\cos\left(\frac{m\pi}{m+1}\right) \\ &= 1 + \beta\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \end{aligned} \tag{8.27}$$

We see that (8.26) $<$ (8.27) and by the arguments (8.15) and (8.16) we can use (8.11) to finish the proof

$$\begin{aligned} \rho_{(1)}(I - D_{(1)}^{-1}A) &= \frac{U_{\frac{m}{2}-1}\left(1 + \frac{1}{\beta}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)}{U_{\frac{m}{2}}\left(1 + \frac{1}{\beta}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)} \\ > \rho_{(2)}(I - D_{(2)}^{-1}A) &= \frac{U_{\frac{m}{2}-1}\left(1 + \beta\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)}{U_{\frac{m}{2}}\left(1 + \beta\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)} \end{aligned} \tag{8.28}$$

(Note that both asymptotic convergence rates depend on h since $m = 1/h$.) $\square$

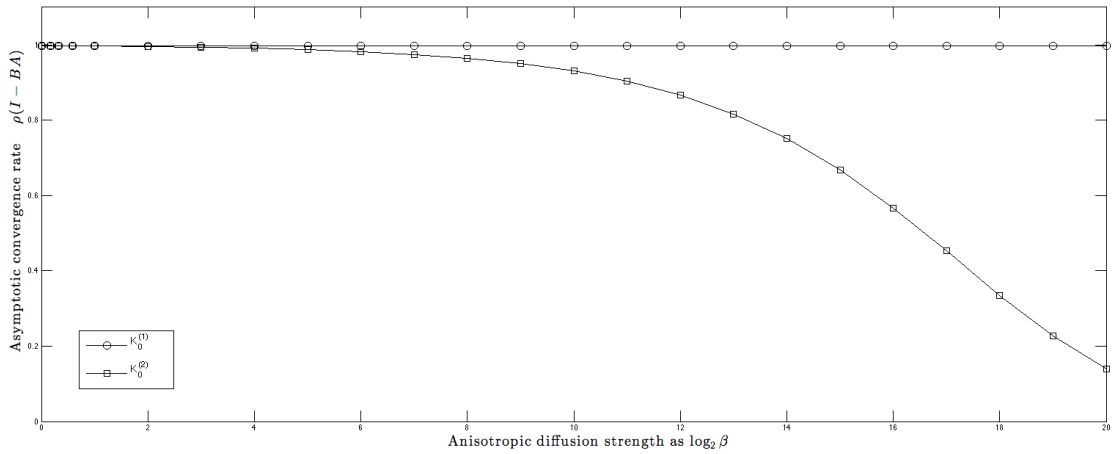


Figure 8.3: Solving $-\beta u_{xx} - u_{yy} - 1 = 0$ for 10^6 unknowns on two processors with $h = 10^{-3}$.

Theorem 8.4.4. *Let two partitions of the unit square* $\Omega = [0,1] \times [0,1] \in \mathbb{R}^2$ *be* $\bar{K}_0^{(1)} = \{[0, 0.5] \times [0,1], [0.5,1] \times [0,1]\}$ *and* $\bar{K}_0^{(2)} = \{[0,1] \times [0, 0.5], [0,1] \times [0.5,1]\}$ *which are then enlarged slightly to overlap. When solving the boundary value problem, find* $u \in H^2(\Omega)$ *such that*

$$\begin{aligned} -\triangle u + \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla u - 1 = 0 \ \textit{on} \ \Omega \\ u = 0 \ \textit{on} \ \partial\Omega \end{aligned} \tag{8.29}$$

with uniform square mesh Finite Difference and Additive Schwarz Domain Decomposition, if $\beta > 1/h$ *where* h *is the mesh size, then the asymptotic convergent rate of using* $\bar{K}_0^{(2)}$ *will be less than using* $\bar{K}_0^{(1)}$. *Enlarge the parts of each partition to include one layer of Finite Difference overlap as in Figure 8.1.*

Proof. Let $m = 1/h$. Using Figure 8.1 and applying Finite Difference with a first order approximation of ∇u, we find the global stiffness matrix A to be $m \times m$ block matrix

$$A = \frac{1}{h^2} \begin{bmatrix} T & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & & -I \\ & & -I & T \end{bmatrix}, \quad T = \begin{bmatrix} 3+\beta h & -1 & & \\ -\beta h & \ddots & \ddots & \\ & \ddots & & -1 \\ & & -\beta h & 3+\beta h \end{bmatrix} \tag{8.30}$$

T is a $m \times m$ matrix. When we use partition $\bar{K}_0^{(1)}$ and rearrange unknowns, then using Additive Schwarz corresponds to solving $AU = F$ with Richardson's Method and a block Jacobi preconditioner (see section 8.2). The iteration matrix is $I - D^{-1}A$ as defined in theorem 8.4.1 with

$$\text{for } \bar{K}_0^{(1)} \quad b, c, x, y = -1/h^2, -1/h^2, -\beta/h, -1/h^2 \tag{8.31}$$

and using partition $\bar{K}_0^{(2)}$ corresponds with

$$\text{for } \bar{K}_0^{(2)} \quad b, c, x, y = -\beta/h, -1/h^2, -1/h^2, -1/h^2 \tag{8.32}$$

Theorem 8.4.6 tells us the asymptotic convergence rate for each partition by (8.11). It involves the quantity $\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right)$.

$$\begin{aligned} \text{for } \bar{K}_0^{(1)} \quad \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right) &= \frac{3+\beta h}{2\sqrt{\beta h}} + \frac{1}{\sqrt{\beta h}} \cos\left(\frac{m\pi}{m+1}\right) \\ &= \frac{1+\beta h}{2\sqrt{\beta h}} + \frac{1}{\sqrt{\beta h}} \left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \\ &= 1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2\sqrt{\beta h}} + \frac{1}{\sqrt{\beta h}} \left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \end{aligned} \tag{8.33}$$

$$\begin{aligned} \text{for } \bar{K}_0^{(2)} \quad \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{m\pi}{m+1}\right) &= \frac{3+\beta h}{2} + \sqrt{\beta h} \cos\left(\frac{m\pi}{m+1}\right) \\ &= \frac{3+\beta h - 2\sqrt{\beta h}}{2} + \sqrt{\beta h} \left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \\ &= 1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2} + \sqrt{\beta h} \left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right) \end{aligned} \tag{8.34}$$

Recognize that $1 + \beta h - 2\sqrt{\beta h} = \left(1 - \sqrt{\beta h}\right)^2 > 0$ and it's given that $\beta h > 1$ then we see that (8.33) $<$ (8.34) and by the arguments (8.15) and (8.16) we can use (8.11) to finish the proof

$$\begin{aligned} \rho_{(1)}(I - D_{(1)}^{-1}A) &= \frac{U_{\frac{m}{2}-1}\left(1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2\sqrt{\beta h}} + \frac{1}{\sqrt{\beta h}}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)}{U_{\frac{m}{2}}\left(1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2\sqrt{\beta h}} + \frac{1}{\sqrt{\beta h}}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)} \\ > \rho_{(2)}(I - D_{(2)}^{-1}A) &= \frac{U_{\frac{m}{2}-1}\left(1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2} + \sqrt{\beta h}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)}{U_{\frac{m}{2}}\left(1 + \frac{1+\beta h - 2\sqrt{\beta h}}{2} + \sqrt{\beta h}\left(1 + \cos\left(\frac{m\pi}{m+1}\right)\right)\right)} \end{aligned} \tag{8.35}$$

□

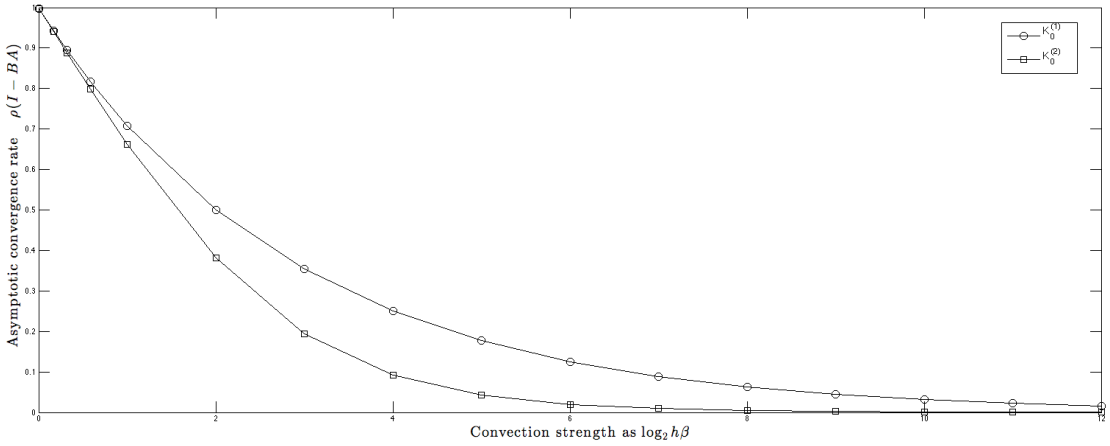


Figure 8.4: Solving $-\triangle u - \beta u_x - 1 = 0$ for 10^6 unknowns on two processors with $h = 10^{-3}$.

Theorem 8.4.5. *Let matrix A be toepliz block tridiagonal with $m \times m$ blocks and m even, whose off diagonal blocks are $m \times m$ diagonal matrices and whose diagonal blocks are $m \times m$ toepliz tridiagonal matrices.*

$$A = \begin{bmatrix} T & yI & & \\ xI & \ddots & \ddots & \\ & \ddots & & yI \\ & & xI & T \end{bmatrix}, \quad T = \begin{bmatrix} a & c & & \\ b & \ddots & \ddots & \\ & \ddots & & c \\ & & b & a \end{bmatrix}$$

Now consider A as a 2×2 block matrix and let D be its diagonal, $-E$ its strictly lower part, and $-F$ its strictly upper part such that $A = D - E - F$. Then the m^2 eigenvalues of $(I - (D - E)^{-1}A)$ are

$$\begin{gathered} \lambda = 0, \left(\frac{U_{\frac{m}{2}-1}(\lambda_T)}{U_{\frac{m}{2}}(\lambda_T)} \right)^2 \\ \text{where } \lambda_T = \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}} \cos\left(\frac{k\pi}{m+1} \right) \text{ for } k = 1, ..., m \end{gathered} \tag{8.36}$$

The eigenvalue 0 is repeated $m^2 - 2m$ times and the other eigenvalues are each repeated 2 or more times. $U_k(\cdot)$ is the k^{th} Chebyshev polynomial of the second kind.

Proof. Since ED^{-1} is strictly lower,

$$\begin{aligned}(D-E)\left(D^{-1}+D^{-1}ED^{-1}\right) &= (I-ED^{-1})(I+ED^{-1}) \\ &= I-(ED^{-1})(ED^{-1}) \\ &= I\end{aligned} \tag{8.37}$$

implies that

$$(D-E)^{-1} = D^{-1}+D^{-1}ED^{-1} \tag{8.38}$$

Let Ã be the the upper left block of D.

$$I-(D-E)^{-1}A = (D-E)^{-1}(F) =$$

$$= \left[\begin{array}{c|c} \tilde{A}^{-1} & 0 \\ \hline 0 & \tilde{A}^{-1} \end{array}\right] \left[\begin{array}{c|c} 0 & 0 \\ \hline 0 & 0 \end{array}\right] \begin{array}{c} \\ -xM_{1,1}M_{\frac{m}{2},\frac{m}{2}} \\ \\ \end{array} \begin{array}{c} \\ -yI \\ \\ \end{array}$$

$$= \left[\begin{array}{c|c|c} & -yM_{1,\frac{m}{2}} & \\ 0 & \vdots & 0 \\ & -yM_{\frac{m}{2},\frac{m}{2}} & \\ \hline & xyM_{1,1}M_{\frac{m}{2},\frac{m}{2}} & \\ 0 & & 0 \end{array}\right] \tag{8.39}$$

where $[M]_{i,j}$ is the block of $[\tilde{A}]^{-1}$ in block row i block column j. $[M]_{i,j}$ is a matrix of size $m \times m$. By symmetry, $[M]_{\frac{m}{2},\frac{m}{2}} = [M]_{1,1}$. Using (8.6)

$$[M]_{1,1} = \frac{1}{\sqrt{xy}} \frac{U_{\frac{m}{2}-1}(\tilde{T})}{U_{\frac{m}{2}}(\tilde{T})} \tag{8.40}$$

The eigenvalues of $I-(D-E)^{-1}A$ are m^2-m zeros corresponding with the m^2-m columns of zeros in $I-(D-E)^{-1}A$ and m non zeros.

$$\lambda\left(I-(D-E)^{-1}A\right) = 0,\ \lambda\left(xyM_{1,1}^2\right) \tag{8.41}$$

Since $U_k(\cdot)$ is a polynomial

$$\lambda\left(U_k(X)\right) = U_k\left(\lambda(X)\right)$$

Since $\tilde{T}$ is a tridiagonal toeplitz matrix,

$$\lambda\left(\tilde{T}\right) = \frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{k\pi}{m+1}\right) \text{ for } k = 1, ..., m \tag{8.42}$$

Complete the proof by substituting (8.42) into (8.40) into (8.41).

□

Theorem 8.4.6. *Let matrix A and function U be define as in Theorem 8.4.5. If $b, c, x, y < 0$ and $a = -b - c - x - y$, then the spectral radius*

$$\rho\left(I - (D-E)^{-1}A\right) = \left(\frac{U_{\frac{m}{2}-1}(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{m\pi}{m+1}\right))}{U_{\frac{m}{2}}(\frac{a}{2\sqrt{xy}} + \frac{\sqrt{bc}}{\sqrt{xy}}\cos\left(\frac{m\pi}{m+1}\right))}\right)^2 \tag{8.43}$$

Proof. The proof follows the form of the proof for Theorem 8.4.6. □

Theorem 8.4.7. *Let two partitions of the unit square $\Omega = [0,1] \times [0,1] \in \mathbb{R}^2$ be $\bar{K}_0^{(1)} = \{[0, 0.5] \times [0, 1], [0.5, 1] \times [0, 1]\}$ and $\bar{K}_0^{(2)} = \{[0, 1] \times [0, 0.5], [0, 1] \times [0.5, 1]\}$ which are then enlarged slightly to overlap. When solving the boundary value problem, find $u \in H^2(\Omega)$ such that*

$$\begin{aligned} -\nabla \cdot \begin{bmatrix} \beta & 0 \\ 0 & 1 \end{bmatrix} \nabla u - 1 = 0 \text{ on } \Omega \\ u = 0 \text{ on } \partial\Omega \end{aligned} \tag{8.44}$$

with uniform right triangle Finite Elements side length h and Multiplicative Schwarz Domain Decomposition, if $\beta > 1$ then the asymptotic convergent rate of using $\bar{K}_0^{(2)}$ will be less than using $\bar{K}_0^{(1)}$. Enlarge the parts of each partition to include one layer of Finite Element overlap as in Figure 8.1.

Proof. The proof follows the form of the proof for Theorem 8.4.3. □

Theorem 8.4.8. *Let two partitions of the unit square $\Omega = [0,1] \times [0,1] \in \mathbb{R}^2$ be $\bar{K}_0^{(1)} = \{[0, 0.5] \times [0, 1], [0.5, 1] \times [0, 1]\}$ and $\bar{K}_0^{(2)} = \{[0, 1] \times [0, 0.5], [0, 1] \times$*

$[0.5, 1]\}$ *which are then enlarged slightly to overlap. When solving the boundary value problem, find* $u \in H^2(\Omega)$ *such that*

$$\begin{aligned} -\triangle u + \begin{bmatrix} \beta \\ 0 \end{bmatrix} \nabla u - 1 = 0 \ \textit{on} \ \Omega \\ u = 0 \ \textit{on} \ \partial\Omega \end{aligned} \tag{8.45}$$

with uniform square mesh Finite Difference and Multiplicative *Schwarz Domain Decomposition, if* $\beta > 1/h$ *where* h *is the mesh size, then the asymptotic convergent rate of using* $\bar{K}_0^{(2)}$ *will be less than using* $\bar{K}_0^{(1)}$*. Enlarge the parts of each partition to include one layer of Finite Difference overlap as in Figure 8.1.*

Proof. The proof follows the form of the proof for Theorem 8.4.4. □

Chapter 9

Numerical Experiments

In the preceding chapter, we proved that the methods presented in Chapter 5 improve the convergence rate of simple Domain Decomposition Methods applied to simple uniform meshes and domains. In this chapter, we will show the effect of the Weighting Schemes presented in Chapters 5 and 7 on more complicated Domain Decomposition Methods, more complicated meshes, and more complicated domains. Once again, we find that these methods improve convergence rates and in some cases, they improve solution accuracy also.

All the numerical experiments in this chapter are performed using PLTMG 11.0, METIS 5.1.0, and SG running on CCoM's computing resource BOOM. PLTMG 11.0 is a package for solving elliptic partial differential equations in general regions of the plane created by Randolph Bank [18]. BOOM is a resource of the Center for Computational Mathematics at the University of California San Diego. BOOM is a ROCKS-based 720-core/1440-GB (60 dual-cpu/six-core/24GB nodes) 64-bit Xeon Cluster (from Dell). METIS 5.1.0 is a serial software package for partitioning large irregular graphs, partitioning large meshes, and computing fill-reduced ordering of sparse matrices created by George Karypis [34]. METIS 5.1.0 is described in more detail in Section 4.3. SG is a visualization tool created by Michael Holst which provides most of the visualizations for this dissertation [30].

9.1 PLTMG 11.0

PLTMG 11.0 solves elliptic partial differential equations by the methods presented in this paper. It starts with an equation in the form of (1.1)-(1.3) and then solves the Weak Form of the problem with a Galerkin approximation from a Finite Element space. The Finite Element space consists of triangles using piecewise continuous polynomial functions created from the standard Lagrange basis functions. The numerical experiments in this research and discussions use only piecewise linear function spaces. If the underlying boundary value problem is not self-adjoint, some upwinding terms based on the Scharfetter-Gummel discretization scheme are added to the discretization.

PLTMG uses sparse matrix storage and solves the system of n equations in n unknowns using Newton's Method even if the system is linear. Therefore finding the degrees of freedom comes from solving the Matrix Form $AU = F$. PLTMG has options for solving this system which include the preconditioned Conjugate Gradient Method. For parallel solves, PLTMG uses the Bank-Holst Paradigm DD solver.

The numerical experiments and discussions in this research mainly use the features above. However, PLTMG has more useful features. A few examples are adaptive refining in both h and p based on a posterior error estimates [11] [12] and a multilevel partitioning algorithm that uses recursive spectral bisection at the coarse level. PLTMG can also solve four other major problem classes including obstacle problems, continuation problems, parameter identification problems, and optimal control problems.

9.1.1 Code Modifications

PLTMG 11.0 has its own graph partitioner, however, the numerical experiments in this chapter used METIS for partitioning meshes because its properties are more well known. You call the METIS partitioning function dynamically during run time with the command, call METIS_ PartGraphRecursive(nvtxs, ncon, xadj, adjncy, vwgt, vs, adjwgt, nparts, tpwgts, ubvec, options, objval, part). The

parameter adjwgt contains the graph edge weights and vwgt contains the graph vertex weights. To call METIS from PLTMG, the following code changes were made; In PLTMG's subroutine ldbal inside the file mg2.f, the following variables were added

```
real(kind=4) :: ubvec
real(kind=4), dimension(nproc) :: tpwgts
integer(kind=4) :: nvtxs,ncon,nparts,objval
integer(kind=4), dimension(ntf+1) :: xadj,vwgt,part,vs
integer(kind=4), dimension(3*ntf) :: adjncy,adjwgt
integer(kind=4), dimension(40) :: options
external METIS_METIS_SetDefaultOptions
external METIS_PartGraphRecursive
common /chris/wght(3,400000)
```

then in subroutine ldbal, immediately after the call to cequvt(ntf, nproc, itnode, itedge, e, p, q, kequvc, kequv), you insert the following code

```
call METIS_SetDefaultOptions(options)
options(8) = 10
index = 1
nvtxs = ntf
ncon = 1
nparts = nproc
ubvec = 1.001
do i=1,nproc
  tpwgts(i) = 1.0/nproc
enddo
do i=1,ntf
  do j=1,3
    if (itedge(j,i)/4>0) then
      adjncy(index)=itedge(j,i)/4 - 1
      n=itedge(j,i)/4
      m=itedge(j,i)-4*n
      adjwgt(index)=wght(j,i)+wght(m,n)
      index=index+1
    endif
  enddo
  xadj(i+1)=index - 1
  vwgt(i)= 1
enddo
```

```
      call METIS_PartGraphRecursive(nvtxs,ncon,xadj,adjncy,
     +    vwgt,vs,adjwgt,nparts,tpwgts,ubvec,options,objval,part)
      go to 50
```

The common variable chris/wght contains the edge and vertex weights. For the Convection Weighting and Gradient Weighting scheme, this variable is set at the very end of the subroutine eleasm inside the file mg1.f. The variable $[ux\ uy]^T$ is a vector in either the direction of the convection or gradient corresponding to whichever scheme you desire.

```
      s = 2.0
      xdiff=vx(itnode(2,itri))-vx(itnode(3,itri))
      ydiff=vy(itnode(2,itri))-vy(itnode(3,itri))
      wght(1,itri)=abs(-uy*xdiff+ux*ydiff)/sqrt(xdiff**2+ydiff**2)
     +   /sqrt(ux**2+uy**2)
      wght(1,itri)=(wght(1,itri)*s)**s+1.0

      xdiff=vx(itnode(1,itri))-vx(itnode(3,itri))
      ydiff=vy(itnode(1,itri))-vy(itnode(3,itri))
      wght(2,itri)=abs(-uy*xdiff+ux*ydiff)/sqrt(xdiff**2+ydiff**2)
     +   /sqrt(ux**2+uy**2)
      wght(2,itri)=(wght(2,itri)*s)**s+1.0

      xdiff=vx(itnode(2,itri))-vx(itnode(1,itri))
      ydiff=vy(itnode(2,itri))-vy(itnode(1,itri))
      wght(3,itri)=abs(-uy*xdiff+ux*ydiff)/sqrt(xdiff**2+ydiff**2)
     +   /sqrt(ux**2+uy**2)
      wght(3,itri)=(wght(3,itri)*s)**s+1.0
```

For the Stiffness Matrix weighting, the variable chris/wght is set at the very end of the subroutine linsys but before the call to deallocate(js,jns). The subroutine linsys is inside the file mg1.f

```
        do i=1,ntf
          i1=itdof(1,i)
          i2=itdof(2,i)
          i3=itdof(3,i)
          alpha=100.0
          beta=0.5
          wght(1,i)=
     +      MAX((abs(geta(i1,i2,nb,ndf,map,a,ja,jns))
```

```
     +        + abs(geta(i1,i3,nb,ndf,map,a,ja,jns)))
     +        /2.0/abs(geta(i1,i1,nb,ndf,map,a,ja,jns)),
     +        abs(geta(i2,i1,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i2,i2,nb,ndf,map,a,ja,jns))
     +        + abs(geta(i3,i1,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i3,i3,nb,ndf,map,a,ja,jns)))
              if (wght(1,i)>1.0) wght(1,i)=1.0
              wght(1,i)=alpha*wght(1,i)+beta

              wght(2,i)=
     +        MAX((abs(geta(i2,i1,nb,ndf,map,a,ja,jns))
     +        + abs(geta(i2,i3,nb,ndf,map,a,ja,jns)))
     +        /2.0/abs(geta(i2,i2,nb,ndf,map,a,ja,jns)),
     +        abs(geta(i1,i2,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i1,i1,nb,ndf,map,a,ja,jns))
     +        + abs(geta(i3,i2,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i3,i3,nb,ndf,map,a,ja,jns)))
              if (wght(2,i)>1.0) wght(2,i)=1.0
              wght(2,i)=alpha*wght(2,i)+beta

              wght(3,i)=
     +        MAX((abs(geta(i3,i2,nb,ndf,map,a,ja,jns))
     +        + abs(geta(i3,i1,nb,ndf,map,a,ja,jns)))
     +        /2.0/abs(geta(i3,i3,nb,ndf,map,a,ja,jns)),
     +        abs(geta(i2,i3,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i2,i2,nb,ndf,map,a,ja,jns))
     +        + abs(geta(i1,i3,nb,ndf,map,a,ja,jns))
     +        /2.0/abs(geta(i1,i1,nb,ndf,map,a,ja,jns)))
              if (wght(3,i)>1.0) wght(3,i)=1.0
              wght(3,i)=alpha*wght(3,i)+beta
            enddo
```

The function geta(i,j,...) extracts the element $[A]_{i,j}$ from the sparse matrix representation of A in PLTMG. The function is

```
        function geta(i,j,nb,ndf,map,a,ja,jap)
c
            use mthdef
            implicit real(kind=rknd) (a-h,o-z)
            implicit integer(kind=iknd) (i-n)
            integer(kind=iknd), dimension(*) :: ja,jap
            integer(kind=iknd), dimension(ndf) :: map
            real(kind=rknd), dimension(*) :: a
```

```
c
c       this routine returns A(i,j)
c
            nb_low = map(i)
            nb_high = map(j)
            if (nb_low == nb_high) then
              geta = a(nb_low)
            endif
            l_shift = 0
            if (nb_low > nb_high) then
              l_shift = ja(nb+1)-ja(1)
              nb_low = map(j)
              nb_high = map(i)
            endif
            if (nb_low /= nb_high) then
              n_index = -1
              do k=ja(nb_low),ja(nb_low+1)-1
                if (nb_high == ja(k)) n_index=k
              enddo
              if (n_index /= -1) then
                geta = a(jap(n_index)+l_shift)
              else
                geta = 0.0
              endif
            endif
        return
        end
```

Occasionally we ran experiments using PLTMG's graph partitioner, then we didn't call METIS in ldbal and we make the following changes at the end of the subroutine mtxasm in the file mg2.f

```
        do i=ibeg,iend
            it=p(i)
            do jj=1,3
                jt=itedge(jj,it)/4
                if(jt>0) then
                    j=q(jt)
                    if(j>=i.and.j<=iend) then
                        kmin=min(map(it),map(jt))
                        kmax=max(map(it),map(jt))
                        if(kmax>kmin) then
```

```
                        err=wght(jj,it)
                        a(kmin)=a(kmin)+err
                        a(kmax)=a(kmax)+err
                        call jamap0(kmin,kmax,ij,ji,ja,0_iknd)
                        a(ij)=a(ij)-err
                    endif
                endif
            endif
        enddo
    enddo
```

The four lines containing the variable *err* have been changed. Originally, instead of *err*, this was just 1.0.

Additionally, PLTMG has preprocessing and postprocessing to the partitioning procedure. Since these were not changed to incorporate the new edge weights, we turned them off. In the beginning of the subroutine cequvt inside the file mg1.f, change "ee = ef * ee/ real(nproc, rknd)" to "ee=0.0" to turn off preprocessing. And in the subroutine ldbal, remove the line "call smth0(ntf, itedge, e, nproc, msize, itnode)" to turn off postprocessing. And, we would set the tolerance for the eigenvalue solve from "tol = 1.0 e-2_rknd" inside the subroutine lbev inside the file mg1.f to "tol = 1.0 e-10".

9.2 Edge Weighting Experiments

The Edge Weighting techniques explained in Chapter 5 include Convection Weighting, Gradient Weighting, and Stiffness Matrix Weighting. In this Chapter, we describe the results of experiments conducted to test their performance on various PDEs, various domains, and various problem sizes. Assume all norms without subscripts to be L^2 or ℓ^2 norms.

9.2.1 Convection

Experiment 1. *Does using Convection, Gradient, and Stiffness Matrix Weighting improve the convergence of Bank-Holst paradigm DD solver when solving convection dominated elliptic partial differential equations?*

We solved, with and without using weighting, a convection dominated elliptic partial differential equation (1.1)-(1.3). Find $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^6\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.1}$$

$$u = 0 \text{ on } \partial\Omega_D \tag{9.2}$$

$$n \cdot \nabla u = 0 \text{ on } \partial\Omega_N \tag{9.3}$$

$\partial\Omega_D$ is the left and right side of the unit square ($x = 0, 1$) while $\partial\Omega_N$ is the top and bottom ($y = 0, 1$). The solution is displayed in Figure 9.1.

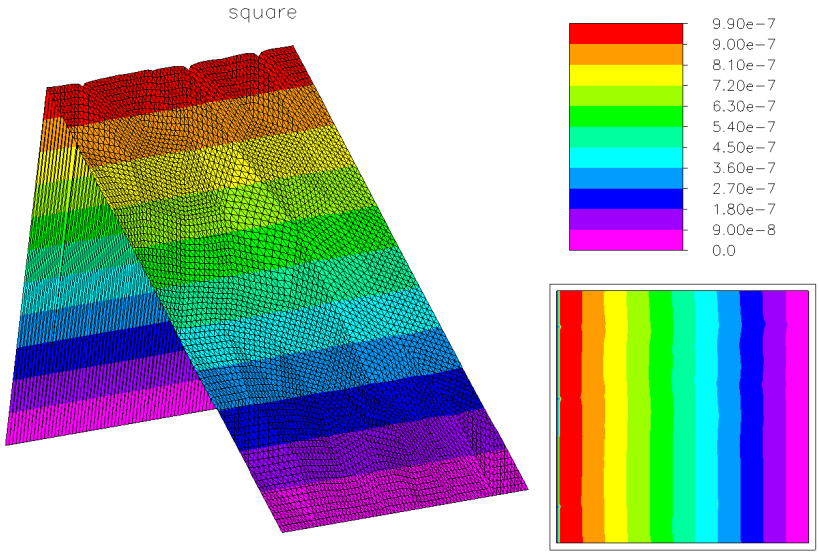
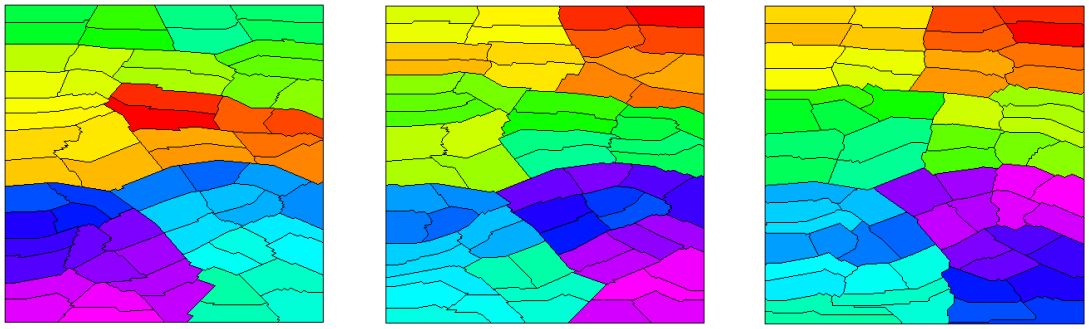


Figure 9.1: Solution to (9.1)-(9.3)

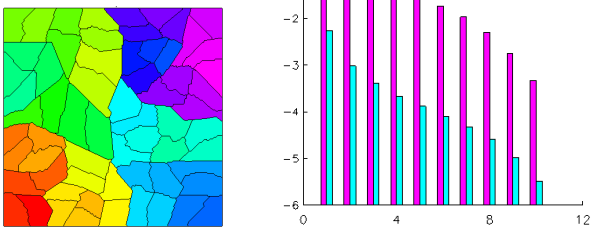
For this problem, all three edge weighting schemes create the same partition because for every $(x, y) \in \Omega$, the gradient is in the direction of the convection. They each made rectangles in the x direction with aspect ratio 4:1. Convection and Gradient weighting used parameter $s = 2$.

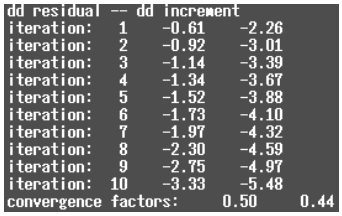


(a) Convection weighting (b) Gradient weighting (c) Stiffness Matrix weighting

Figure 9.2: The three edge weighting schemes partitioning for (9.1)-(9.3)

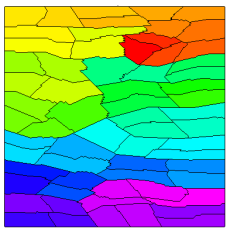
We partitioned the domain into 64 parts starting from a uniform mesh of 2.0×10^4 unknowns (4.0×10^4 triangles). Afterward, each part refined their mesh uniformly to 2.0×10^5 unknowns. The total problem size was around 10^7 degrees of freedom.

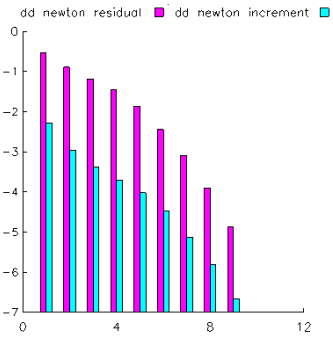




(a) Unweighted partition (b) DD convergence (c) DD convergence

Figure 9.3: Unweighted partitioning scheme solving (9.1)-(9.3)

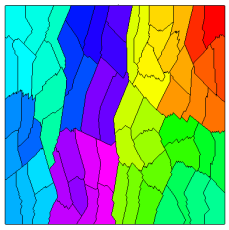


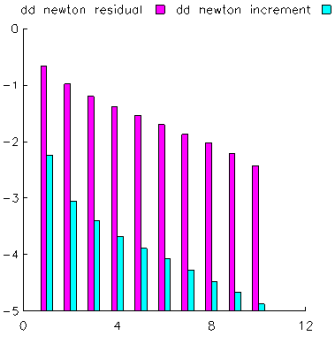


dd residual	--	dd increment
iteration: 1	-0.53	-2.29
iteration: 2	-0.89	-2.96
iteration: 3	-1.19	-3.38
iteration: 4	-1.45	-3.70
iteration: 5	-1.87	-4.02
iteration: 6	-2.45	-4.49
iteration: 7	-3.09	-5.12
iteration: 8	-3.90	-5.82
iteration: 9	-4.87	-6.67
convergence factors:	0.29	0.28

(a) Weighted partition (b) DD convergence (c) DD convergence

Figure 9.4: Convection, Gradient, or Stiffness Matrix weighted partitioning scheme solving (9.1)-(9.3)





dd residual	--	dd increment
iteration: 1	-0.66	-2.24
iteration: 2	-0.98	-3.05
iteration: 3	-1.20	-3.41
iteration: 4	-1.38	-3.69
iteration: 5	-1.53	-3.89
iteration: 6	-1.70	-4.07
iteration: 7	-1.87	-4.28
iteration: 8	-2.02	-4.47
iteration: 9	-2.21	-4.66
iteration: 10	-2.43	-4.88
convergence factors:	0.64	0.51

(a) Perpendicular weighted partition (b) DD convergence (c) DD convergence

Figure 9.5: Perpendicular convection weighted partitioning scheme solving (9.1)-(9.3)

The values in figure parts (b) and (c) from Figures 9.3-9.5 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. In all three cases, $||r_0|| = 0.66$ and $||u_0|| = 6.5 \times 10^{-5}$.

From this experiment, you can see that using parts that favor the convection direction improves convergence. The asymptotic convergence rates of the residual and u increment were 0.29 and 0.28 respectively when using Convection, Gradient, or Stiffness Matrix Weighting. Without them, the asymptotic convergence rates

were 0.50 and 0.44 respectively. For comparison, we also partitioned the mesh using the perpendicular to the convection. The asymptotic convergence rate of this this partition was 0.64 and 0.51 respectively. In all three cases, the global mesh produced a finite element solution with error $||e_h|| = ||u - u_h|| = 2.9 \times 10^{-9}$ which means that all three final global solutions have equal accuracy.

A reasonable DD Method iteration stopping criteria is when $||\delta u_k|| = ||u_k - u_{k-1}|| < \frac{1}{10}||e_h||$. According to this criteria, it took the unweighted scheme 10 iterations to converge while it took the weighted scheme 8 iterations and the perpendicular weighted scheme 12 iterations. Overall, the weighted scheme found the same final solution as the others in less time.

9.2.2 Convection Strength

Experiment 2. *How much convection needs to be present for a PDE to be convection dominated and benefit from using a convection weighted partitioning scheme?*

In Experiment 1 (section 9.2.1), the final global mesh had $75\% \cdot 400000 \cdot 64 = 1.92 \times 10^7$ triangles. This made the average side of each triangle 3.47×10^{-4} and therefore the convection strength was $||b||h/||a|| = 10^6 \cdot 3.47 \times 10^{-4}/1 = 347$ where b is the coefficient vector of the gradient of u and a is the matrix that scales the Lapacian in the PDE.

We solved the same equation from Experiment 1, (9.1)-(9.3) (section 9.2.1), while varying β where $b = [\beta\ 0]^T$ using Convection Weighted partitioning with convection parameter $s = 2$. The results are summarized in Table 9.1.

Table 9.1: 64 Processors; DD convergence rate versus convection strength

$\|\|b\|\|h$	$\|\|b\|\|$	unweighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	weighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	unweighted convergence $\|\|\delta u_{k+1}\|\|/\|\|\delta u_k\|\|$	weighted convergence $\|\|\delta u_{k+1}\|\|/\|\|u_k\|\|$	iteration reduction log / log
347	10^6	0.50	0.29	0.44	0.28	0.64
34.7	10^5	0.50	0.28	0.44	0.28	0.64
3.47	10^4	0.45	0.24	0.39	0.24	0.66
1.10	$10^{3.5}$	0.29	0.21	0.25	0.19	0.83
0.347	10^3	0.26	0.27	0.16	0.17	1.0
0.0347	10^2	0.33	0.38	0.23	0.28	1.2

From this experiment, it appears that Convection Weighting improves convergence when $||b||h/||a|| \geq 1.0$. And Convection Weighting begins to degrade convergence when $||b||h/||a|| < 1.0$. Interestingly, this coincides with when the Stiffness Matrix Weighting scheme begins making squares in the convection direction instead of rectangles (see Section 5.3.3). More data about how much convection is needed is presented in Experiment 6 (section 9.2.6).

9.2.3 Convection Weighting Parameter

Experiment 3. *What is the optimal rectangle aspect ratio when using Convection, Gradient, and Stiffness Matrix weighting? Does the ideal aspect ratio depend on convection strength? Does it depend on problem size and/or number of processors? Questions 1 and 2 are discussed in this section. Question 3 is discussed in Experiment 7 (section 9.2.7).*

We solved the same problem repeatedly with a combination of different aspect ratios, different convection strengths, different numbers of degrees of freedom, and different numbers of processors. We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.4}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.5}$$

The results from using 64 processors with each processor refined to 2.0×10^5 unknowns is displayed in Table 9.2 and Figure 9.6. (The results from other numbers of processors and problem sizes are reported in Section 9.2.7.) The results from 64 processors demonstrate that the ideal aspect ratio is independent of convection strength. From this data, the ideal aspect ratio is between 3 and 5 inclusive. You want the smallest aspect ratio that produces satisfactory convergence rates. Increasing the aspect ratio unnecessarily increases interface length and communication time. When using METIS, an aspect ratio $3 \leq r \leq 5$ is accomplished by setting the Convection Weighting parameter s to a value $1.5 \leq s \leq 2$. Other graph partitioners will require you to calibrate s differently to produce the desired rectangle aspect ratio. For example, with PLTMG, you need $2.5 \leq s \leq 3.0$. The

Stiffness Matrix Weighting scheme is naturally calibrated to produce aspect ratios of 4:1 with METIS (see Example 5.5.1 and Section 5.3).

Table 9.2: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on $||b||h$ and convection weighting parameter s.

$\|\|b\|\|h$	$\|\|b\|\|$	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	16:1 4.0	25:1 5.0	4:1 stiffness
3470	10^7	0.34	0.29	0.23	0.23	0.26	0.25	0.28	0.26	0.24
		1.0	0.87	0.73	0.73	0.80	0.78	0.85	0.80	0.76
347	10^6	0.34	0.29	0.23	0.23	0.25	0.25	0.26	0.25	0.24
		1.0	0.87	0.69	0.78	0.78	0.78	0.80	0.78	0.76
34.7	10^5	0.34	0.28	0.23	0.22	0.25	0.25	0.26	0.25	0.24
		1.0	0.85	0.73	0.71	0.78	0.78	0.80	0.78	0.76
3.47	10^4	0.30	0.24	0.20	0.19	0.21	0.21	0.23	0.21	0.20
		1.0	0.82	0.73	0.71	0.75	0.75	0.80	0.77	0.73
1.10	$10^{3.5}$	0.26	0.22	0.15	0.14	0.15	0.14	0.16	0.16	0.14
		1.0	0.89	0.71	0.69	0.71	0.69	0.74	0.74	0.69
0.347	10^3	0.14	0.13	0.12	0.12	0.13	0.11	0.14	0.20	0.13
		1.0	0.96	0.93	0.93	0.96	0.89	1.0	1.22	0.96
0.0	0.0	0.37	0.39	-	-	0.51	-	0.69	-	0.37
		1.0	1.06	-	-	1.48	-	2.68	-	1.0

DD iteration reduction factor is calculated by $\log(r_0)/\log(r_k)$ where r_k is the convergence rate for $s = k$. Note that when $s = 0$, the rectangles have aspect ratio 1:1 and the resultant partition is the same as an unweighted partition. These reduction factors are displayed in Table 9.2 and Figure 9.6.

The $||b||h = 0$ line indicates that when the PDE doesn't have convection nor anisotropic diffusion, using rectangle parts degrades the DD convergence rate. Therefore when $||b||h < 1.0$, the optimal parameter is $s = 0$ which turns convection weighting off.

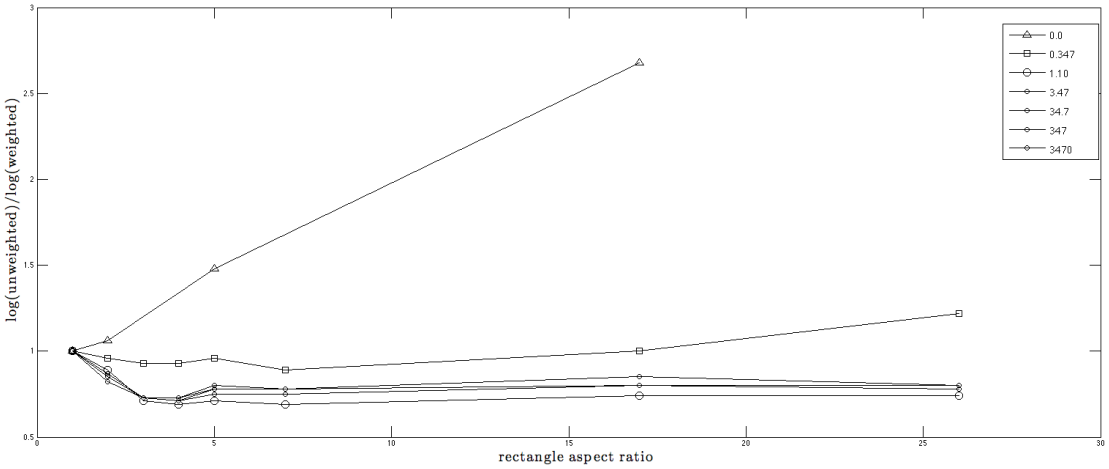
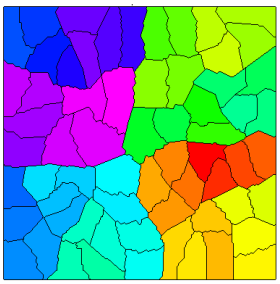
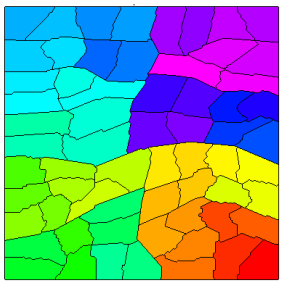


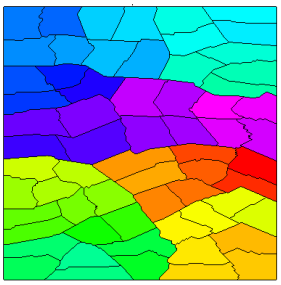
Figure 9.6: The factor of DD iteration reduction versus rectangle aspect ratio . Each line represents using a different convection strength $||b||h$.



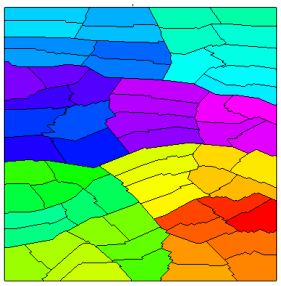
(a) $s = 0$, aspect 1:1



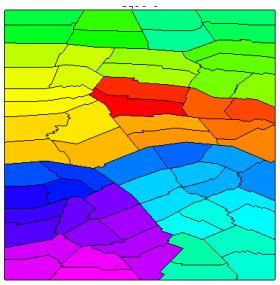
(b) $s = 1$, aspect 2:1



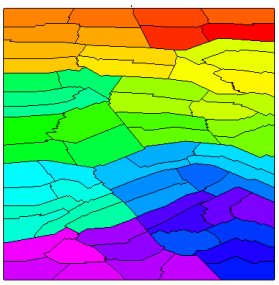
(c) $s = 1.56$, aspect 3:1



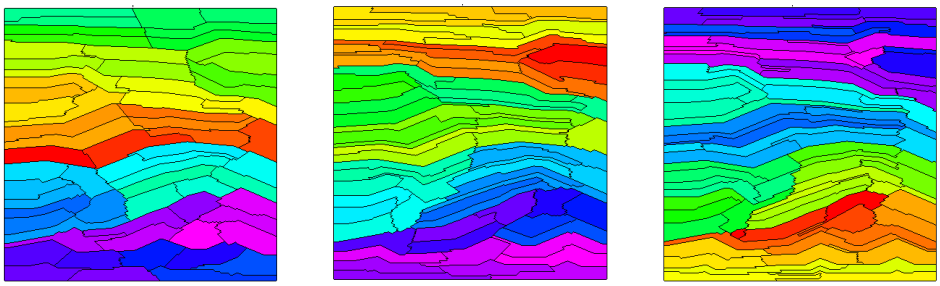
(d) $s = 1.83$, aspect 4:1



(e) $s = 2$, aspect 5:1



(f) $s = 2.23$, aspect 7:1



(g) $s = 3$, aspect 10:1 (h) $s = 4$, aspect 16:1 (i) $s = 5$, aspect 25:1

Figure 9.7: Different convection parameter s values partitioning the unit square into 64 parts.

9.2.4 Boundary Conditions

Experiment 4. *Does the type of boundary condition affect the success of using Convection, Gradient, and Stiffness Matrix weighting?*

We solved the same equation as Experiments 1 and 2 (sections 9.2.1 and 9.2.2) but this time, we used all Dirichlet boundary conditions instead of using half Neumann. Find $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^4 \ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.6}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.7}$$

Both Convection Weighting and Stiffness Matrix Weighting produced the same partitions while Gradient Weighting produced a partition that was slightly different than the others because of the Dirichlet boundary condition on the top and bottom of the unit square. These partitions are displayed in Figure 9.9. The solution to this PDE is pictured in Figure 9.8.

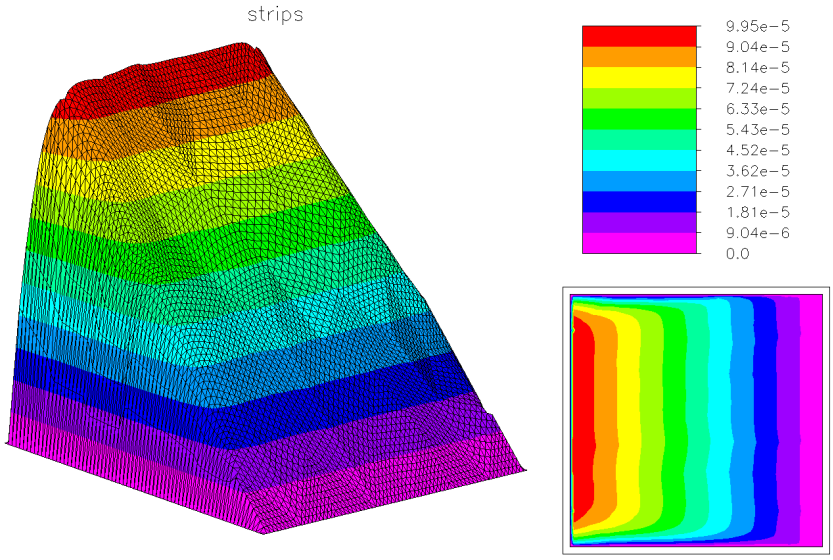
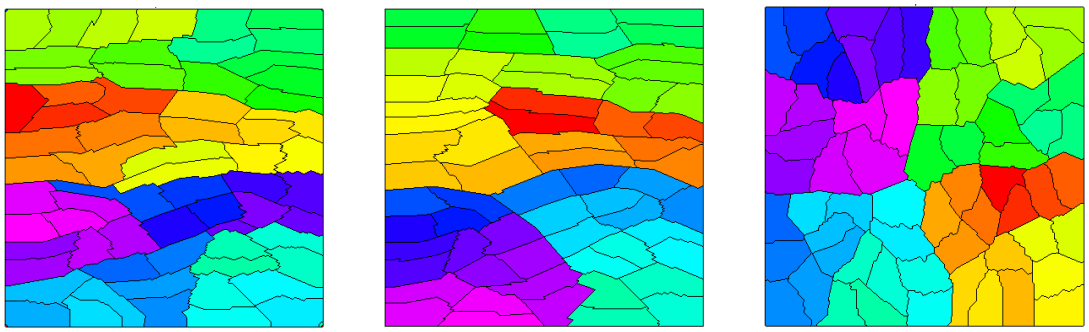
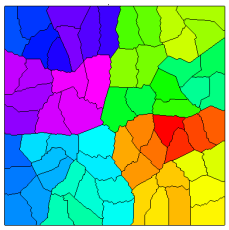


Figure 9.8: Solution to equation (9.6)

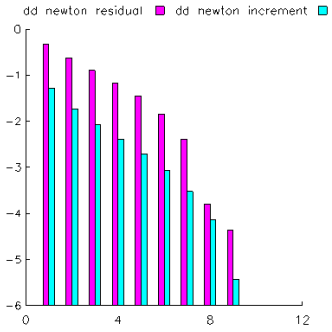


(a) Gradient weighting (b) Convection and Stiffness (c) Unweighted

Figure 9.9: The three edge weighting schemes partitioning for (9.6)-(9.7)



(a) Unweighted partition

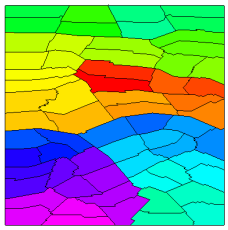


(b) DD convergence

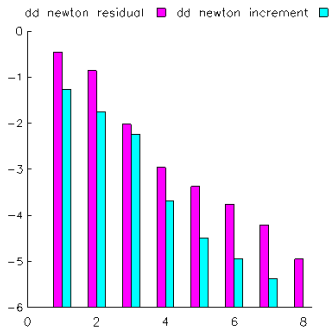
```
dd residual -- dd increment
iteration:   1    -0.32    -1.29
iteration:   2    -0.62    -1.73
iteration:   3    -0.89    -2.08
iteration:   4    -1.16    -2.39
iteration:   5    -1.46    -2.70
iteration:   6    -1.84    -3.06
iteration:   7    -2.38    -3.52
iteration:   8    -3.80    -4.14
iteration:   9    -4.36    -5.43
convergence factors:     0.31     0.30
```

(c) DD convergence

Figure 9.10: Unweighted partitioning scheme solving (9.6)-(9.7)



(a) Weighted partition



(b) DD convergence

```
dd residual -- dd increment
iteration:   1    -0.45    -1.26
iteration:   2    -0.86    -1.75
iteration:   3    -2.02    -2.24
iteration:   4    -2.95    -3.69
iteration:   5    -3.37    -4.50
iteration:   6    -3.75    -4.94
iteration:   7    -4.20    -5.37
iteration:   8    -4.95    -5.99
convergence factors:     0.23     0.21
```

(c) DD convergence

Figure 9.11: Convection/ Stiffness Matrix weighted partitioning scheme solving (9.6)-(9.7)

The values in figure parts (b) and (c) from Figures 9.10-9.11 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. In both cases, $||r_0|| = 8.73$ and $||u_0|| = 5.43 \times 10^{-3}$.

From this experiment, we can see that using parts that favor the convection direction improves convergence even with these new boundary conditions. The asymptotic convergence rates of the residual and u increment were 0.23 and 0.21 respectively when using Convection or Stiffness Matrix Weighting. Gradient Weighting produced similar results with convergence rates of 0.21 and 0.20. With-

out weighting, the asymptotic convergence rates are 0.31 and 0.30 respectively. In both cases, the global mesh produced $||e_h|| = ||u - u_h|| = 2.47 \times 10^{-7}$ which means that both final global solutions have equal accuracy.

A reasonable DD Method iteration stopping criteria is when $||\delta u_k|| = ||u_k - u_{k-1}|| < \frac{1}{10}||e_h||$. According to this criteria, it took the unweighted scheme 10 iterations to converge while it took the weighted scheme 7 iterations. Overall, the weighted scheme found the same final solution in less time.

9.2.5 Force Functions

Experiment 5. *Does the presence of a forcing function in the PDE affect the success of using Convection, Gradient, and Stiffness Matrix weighting?*

We solved the same equation as Experiment 4 (section 9.2.4) but added a forcing function. Find $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^4 \ 0]^T \cdot \nabla u - f(x,y) = 0 \text{ on } \Omega = [0, 2\pi] \times [0, 2\pi] \in \mathbb{R}^2 \tag{9.8}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.9}$$

$$f(x,y) = 2\sin(x)\sin(y) - 10^4 \cos(x)\sin(y) \tag{9.10}$$

The exact solution is $u(x,y) = \sin(x)\sin(y)$ and is shown in Figure 9.12.

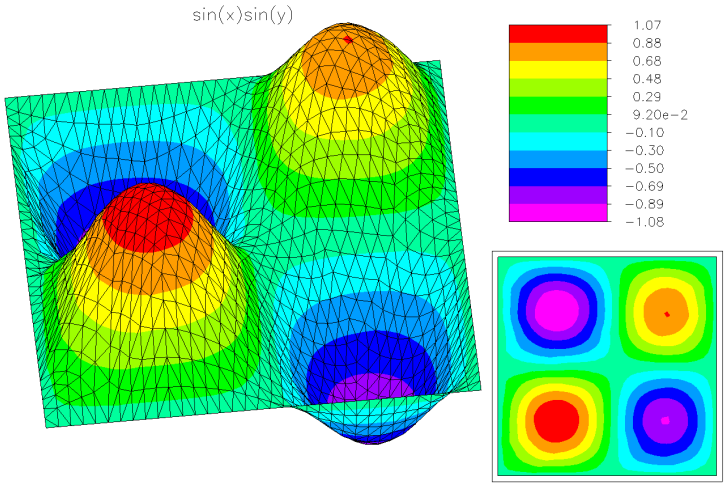
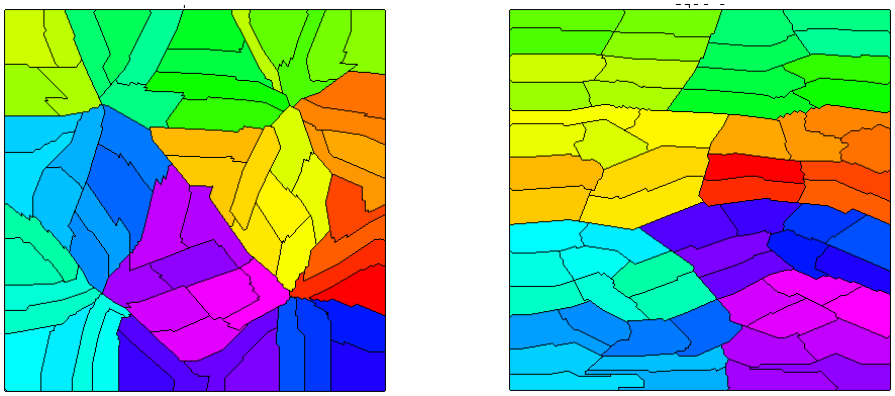


Figure 9.12: $u(x,y) = \sin(x)\sin(y)$

The results of this experiment are similar to Experiment 4 (section 9.2.4) with one difference. The Gradient Weighting scheme did not improve convergence compared to an unweighted scheme.



(a) Gradient weighting partition

(b) Convection or Stiffness Matrix weighting partition

Figure 9.13: Gradient versus Convection or Stiffness Matrix Weighting

This is expected because the final solution's gradient is not in the direction of convection whereas it was in the previous Experiment 4 (section 9.2.4). Figure 9.13a shows what the Gradient Weighting partition looked like for this problem. Using Convection or Stiffness Matrix Weighting did improve convergence similar to Experiment 4 (section 9.2.4).

When using Convection or Stiffness Matrix Weighting, the asymptotic convergence rates of the residual and u increment were 0.30 and 0.29 respectively. Without weighting, the asymptotic convergence rates are 0.48 and 0.41 respectively. (Gradient weighting produced 0.50 and 0.42.) Using the Convection or Stiffness Matrix weighting scheme found the solution in 10 DD iterations while an unweighted scheme took 12 iterations. For both, $||r_0|| = 6.5 \times 10^4$, $||u_0|| = 4.5$, and $||e_h|| = 8.0 \times 10^{-5}$.

One would expect this experiment to produce the same results as Experiment 4 (section 9.2.4) because changing $f(x, y)$ in the PDE does not affect the stiffness matrix and asymptotic convergence relates to the stiffness matrix.

9.2.6 Number of Processors

Experiment 6. *Does the number of processors affect the success of using Convection, Gradient, and Stiffness Matrix weighting? Does the number of processors affect how much convection is needed for Convection, Gradient, and Stiffness Matrix weighting to improve convergence?*

We solved the same equations as Experiment 2 (section 9.2.2) but varied the number of processors. Find $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.11}$$

$$u = 0 \text{ on } \partial\Omega_D \tag{9.12}$$

$$n \cdot \nabla u = 0 \text{ on } \partial\Omega_N \tag{9.13}$$

on 128, 256, and 512 processors using the Convection Weighted scheme with convection parameters $s = 2, 3, 3$ respectively and a variety of values for β. In Experiment 2 (section 9.2.2), we solved (9.11)-(9.13) on 64 processors with different values for β and displayed the results in Table 9.1. Tables 9.3-9.5 show the results for 128, 256, and 512 processors respectively. Each processor refined to 2.0×10^5 unknowns. The global problem sizes were $\approx 2.5 \times 10^6$, 5.0×10^7, and 1.0×10^8 degrees of freedom respectively.

Table 9.3: 128 Processors; DD convergence rate versus convection strength

$\|\|b\|\|h$	$\|\|b\|\|$	unweighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	weighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	unweighted convergence $\|\|\delta u_{k+1}\|\|/\|\|\delta u_k\|\|$	weighted convergence $\|\|\delta u_{k+1}\|\|/\|\|u_k\|\|$	iteration reduction log / log
2450	10^7	0.67	0.50	0.55	0.45	0.75
245	10^6	0.67	0.50	0.55	0.45	0.75
24.5	10^5	0.67	0.50	0.55	0.45	0.75
2.45	10^4	0.58	0.39	0.47	0.35	0.72
0.245	10^3	0.26	0.30	0.18	0.19	1.0
0.0245	10^2	0.39	0.34	0.20	0.24	1.1

Table 9.4: 256 Processors; DD convergence rate versus convection strength

$\|\|b\|\|h$	$\|\|b\|\|$	unweighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	weighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	unweighted convergence $\|\|\delta u_{k+1}\|\|/\|\|\delta u_k\|\|$	weighted convergence $\|\|\delta u_{k+1}\|\|/\|\|u_k\|\|$	iteration reduction log / log
173	10^6	0.72	0.56	0.59	0.47	0.70
17.3	10^5	0.72	0.55	0.59	0.47	0.70
1.73	10^4	0.62	0.38	0.51	0.33	0.61
0.173	10^3	0.29	0.31	0.18	0.21	1.1
0.0173	10^2	0.36	0.59	0.26	0.43	1.3

Table 9.5: 512 Processors; DD convergence rate versus convection strength

$\|\|b\|\|h$	$\|\|b\|\|$	unweighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	weighted convergence $\|\|r_{k+1}\|\|/\|\|r_k\|\|$	unweighted convergence $\|\|\delta u_{k+1}\|\|/\|\|\delta u_k\|\|$	weighted convergence $\|\|\delta u_{k+1}\|\|/\|\|u_k\|\|$	iteration reduction log / log
123	10^6	0.73	0.61	0.60	0.51	0.76
12.3	10^5	0.73	0.61	0.59	0.50	0.76
1.23	10^4	0.62	0.42	0.51	0.35	0.64
0.123	10^3	0.30	0.33	0.19	0.25	1.2
0.0123	10^2	0.38	0.66	0.28	0.48	1.7

This experiment shows that using Convection, Gradient, and Stiffness Matrix weighting improves convergence even when the number of processors employed changes. And, we see again that convection strength needs to be $||b||h/||a|| \geq 1$ to gain an improvement. And when $||b||h/||a|| < 1$, forcing the partition to have rectangles degrades convergence.

9.2.7 Convection Scalability

Experiment 7. *What is the optimal rectangle aspect ratio when using Convection, Gradient, and Stiffness Matrix weighting? Does the ideal aspect ratio depend on convection strength? Does it depend on problem size and/or number of processors? Questions 1 and 2 were discussed in Experiment 3 (section 9.2.3). Question 3 is discussed in this section.*

We solved the same problem repeatedly with a combination of different aspect ratios, different numbers of degrees of freedom, and different numbers of

processors while keeping $||b||h = 34.2$ constant. For the base cases of 2.0×10^5 in Table 9.6 and 64 processors in Table 9.2.7 this is accomplished by letting $\beta = 10^5$. Then as you double the number of unknowns or double the number of processors, you increase β by a factor of $\sqrt{2}$.

We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.14}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.15}$$

Table 9.6: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on the number of unknowns per processor and convection weighting parameter s.

unknowns per proc	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	16:1 4.0	25:1 5.0
2.0×10^5	0.34	0.28	0.23	0.22	0.25	0.25	0.26	0.25
	1.0	0.85	0.73	0.71	0.78	0.78	0.80	0.78
4.0×10^5	0.37	0.31	0.25	0.24	0.27	0.27	0.29	0.29
	1.0	0.85	0.72	0.70	0.76	0.76	0.80	0.80
8.0×10^5	0.38	0.31	0.25	0.25	0.28	-	0.29	-
	1.0	0.83	0.70	0.70	0.76	-	0.78	-

When $s = 0$, aspect ratio is $1:1$ which is the same as using no weighting. Therefore the factor of DD iteration reduction equals $\log(r_0)/\log(r_k)$ where r_k is the convergence rate for $s = k$. These factors are displayed in Figure 9.14 and Table 9.6.

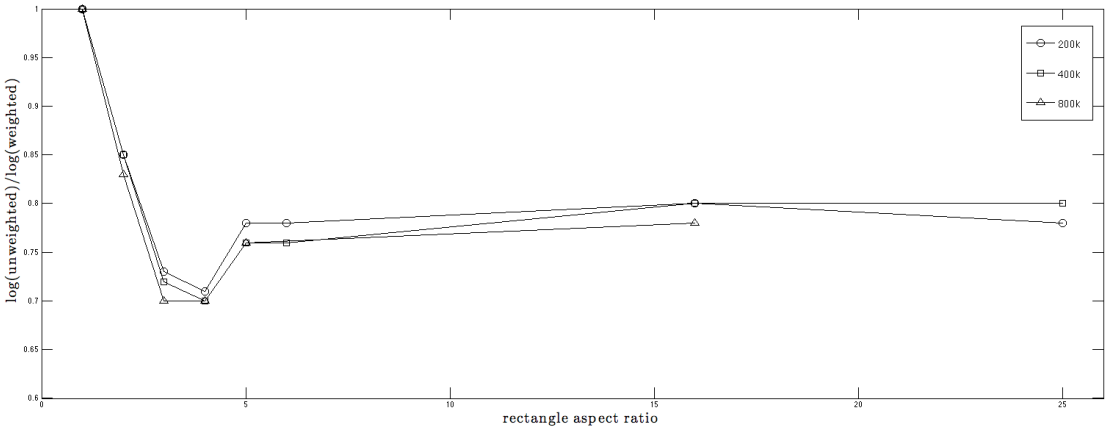


Figure 9.14: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different number of unknowns per processor.

From this experiment, it appears that the number of degrees of freedom per processor does not affect the optimal rectangle aspect ratio of $2 \leq r \leq 5$ that we concluded in Experiment 3 (section 9.2.3).

Table 9.7: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on number of processors and convection weighting parameter s.

processors	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	10:1 3.0	13:1 3.5	16:1 4.0	25:1 5.0
64	0.34	0.28	0.23	0.22	0.25	0.25	-	-	0.26	0.25
	1.0	0.85	0.73	0.71	0.78	0.78	-	-	0.80	0.78
128	0.52	0.37	0.34	0.31	0.34	0.35	-	-	0.34	0.35
	1.0	0.66	0.61	0.56	0.61	0.62	-	-	0.61	0.62
256	0.64	0.62	0.58	0.42	0.41	0.41	-	-	0.37	0.39
	1.0	0.93	0.82	0.51	0.50	0.50	-	-	0.45	0.47
512	0.70	-	0.68	-	0.66	0.65	0.45	-	.41	0.41
	1.0	-	0.92	-	0.86	0.83	0.45	-	0.40	0.40
900	0.71	-	-	-	0.70	-	0.54	0.42	0.43	0.46
	1.0	-	-	-	0.96	-	0.56	0.39	0.40	0.40

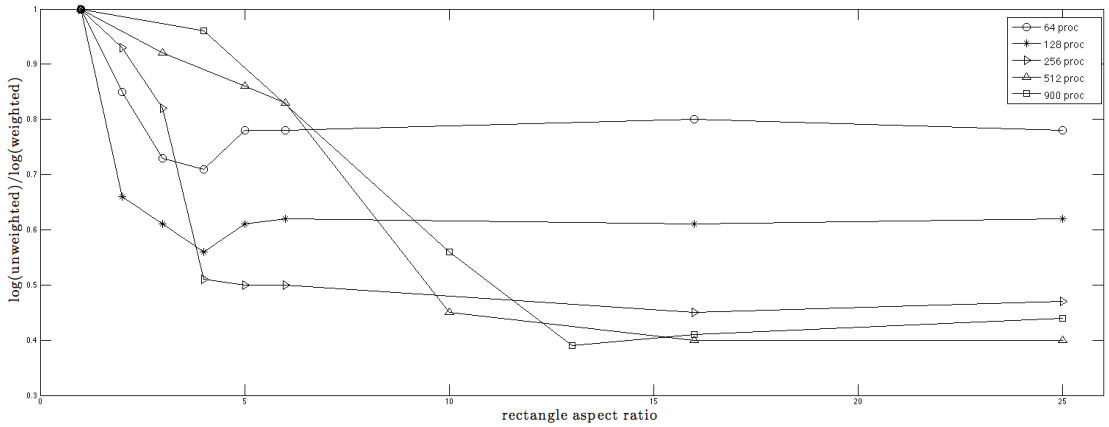
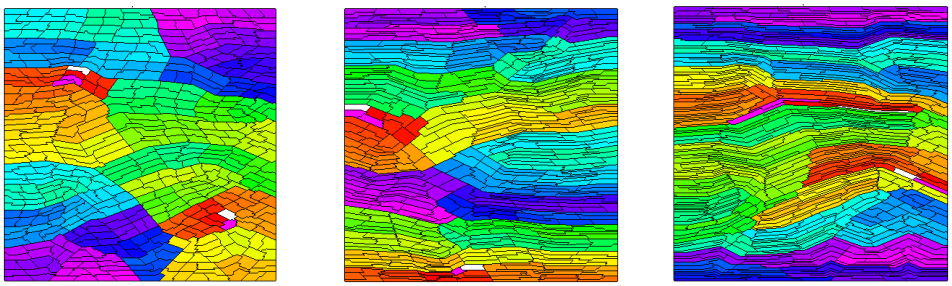


Figure 9.15: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different number of processors.

From these experiments, it appears that the optimal rectangle aspect ratio is dependent on the number of processors. If you set the convection weighting parameter s to $1.5 \leq s \leq 2$ as was previously discussed to create rectangles of aspect ratio $3 \leq r \leq 5$, you will not get optimal convergence when using 512 or 900 processors. The smallest choice for s to achieve optimal convergence with 512 processors appears to be $s = 3$ and for 900 processors, $s = 3.5$. Figure 9.7 displays what different s partitions look like when dividing the unit square into 64 parts. Figure 9.16 pictures some s partitions of 512 parts.



(a) $s = 2$, aspect 5:1 (b) $s = 3$, aspect 10:1 (c) $s = 4$, aspect 16:1

Figure 9.16: Different s values partitioning the unit square into 512 parts.

9.2.8 Diffusion

Experiment 8. *Do Edge Weighting schemes improve the convergence of Bank-Holst paradigm DD solver when solving anisotropic diffusion elliptic partial differential equations?*

We solve find $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} 10^2 & 0 \\ 0 & 1 \end{bmatrix} \nabla u + 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.16}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.17}$$

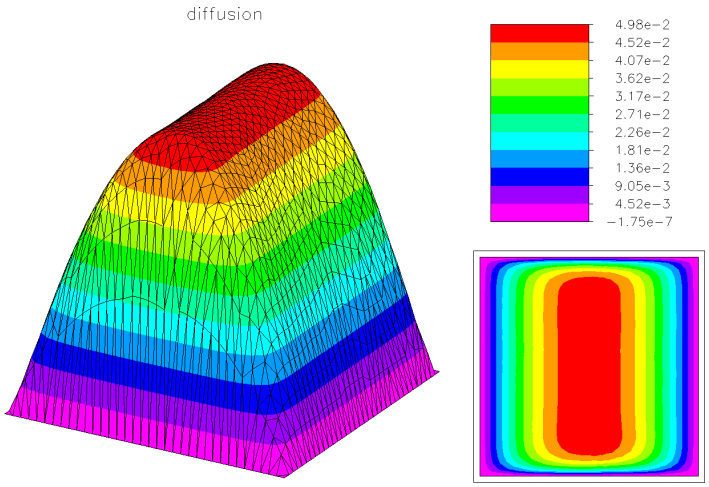
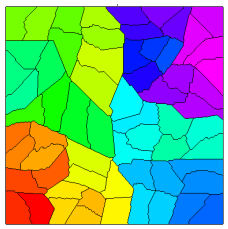
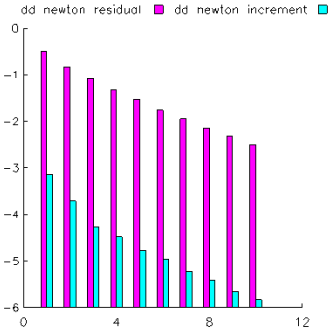


Figure 9.17: Solution to equation (9.16)

Using Stiffness Matrix Weighting or Gradient Weighting detects the anisotropic diffusion and when partitioning both create rectangle parts in the prominent diffusion direction. Convection Weighting performed identical to the unweighted scheme since there is no Convection. Below are the results using Stiffness Matrix Weighting. Gradient Weighting is similar.



(a) Unweighted partition

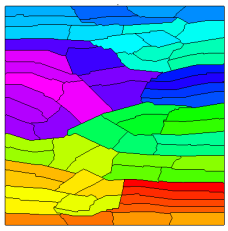


(b) DD convergence

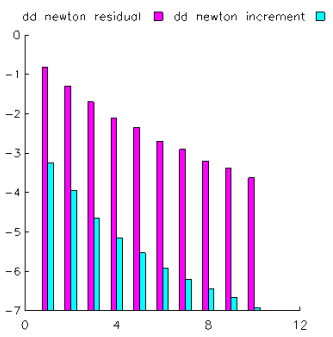
dd residual	--	dd increment
iteration:	1	-0.49 -3.15
iteration:	2	-0.83 -3.71
iteration:	3	-1.08 -4.26
iteration:	4	-1.32 -4.48
iteration:	5	-1.53 -4.77
iteration:	6	-1.76 -4.96
iteration:	7	-1.94 -5.22
iteration:	8	-2.15 -5.41
iteration:	9	-2.32 -5.65
iteration:	10	-2.50 -5.83
convergence factors:		0.60 0.50

(c) DD convergence

Figure 9.18: Unweighted partitioning scheme solving (9.16)-(9.17)



(a) Weighted partition



(b) DD convergence

dd residual	--	dd increment
iteration:	1	-0.81 -3.24
iteration:	2	-1.30 -3.94
iteration:	3	-1.70 -4.64
iteration:	4	-2.10 -5.16
iteration:	5	-2.36 -5.54
iteration:	6	-2.69 -5.92
iteration:	7	-2.90 -6.20
iteration:	8	-3.21 -6.44
iteration:	9	-3.38 -6.66
iteration:	10	-3.62 -6.93
convergence factors:		0.49 0.39

(c) DD convergence

Figure 9.19: Stiffness Matrix weighted partitioning scheme solving (9.16)-(9.17)

The values in figure parts (b) and (c) from Figures 9.18-9.19 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. In both experiments, $||r_0|| = 87.7$ and $||u_0|| = 5.81 \times 10^{-3}$.

From these experiments, you can see that using parts that favor the diffusion direction improves convergence. The asymptotic convergence rates of the residual and u increment were 0.49 and 0.39 respectively when using Stiffness Matrix Weighting. Gradient Weighting produced similar results. Without weighting, the asymptotic convergence rates are 0.60 and 0.50 respectively. Using Stiffness Matrix Weighting found the solution in 12 iterations while an unweighted scheme took 17

iterations. In both cases, the global mesh produced $||e_h|| = ||u - u_h|| = 2.5 \times 10^{-9}$ which means that both final global solutions have equal accuracy. Overall, using weighting found the solution in less time.

Next we solved find $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} 50.5 & 49.5 \\ 49.5 & 50.5 \end{bmatrix} \nabla u + 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.18}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.19}$$

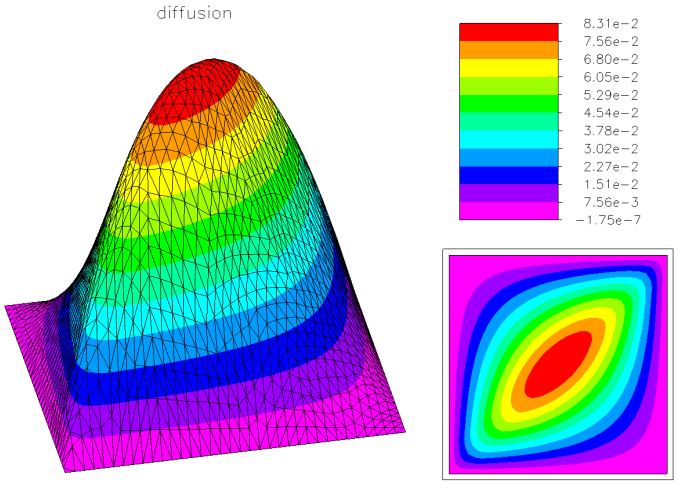
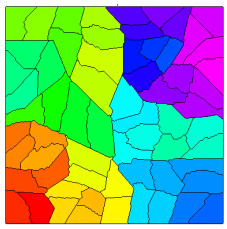
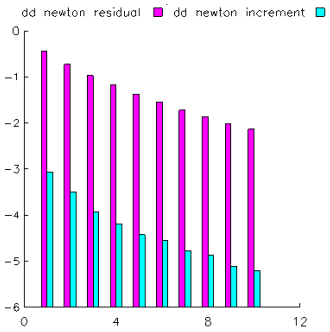


Figure 9.20: Solution to equation (9.18)

Here, $a = \frac{1}{2} \begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 10^2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$. This is equation (9.16) with the diffusion rotated $\pi/4$ radians. Again we solved it with and without Stiffness Matrix Weighting and witnessed similar results. This time Gradient Weighting did not produce a similar partition to Stiffness Matrix weighting and therefore didn't improve convergence.



(a) Unweighted partition

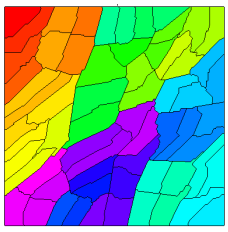


(b) DD convergence

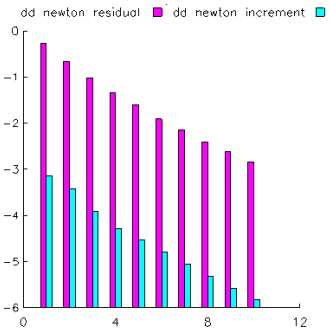
dd residual	--	dd increment
iteration:	1	-0.44 -3.06
iteration:	2	-0.73 -3.50
iteration:	3	-0.97 -3.92
iteration:	4	-1.17 -4.19
iteration:	5	-1.37 -4.42
iteration:	6	-1.54 -4.55
iteration:	7	-1.72 -4.78
iteration:	8	-1.86 -4.88
iteration:	9	-2.02 -5.10
iteration:	10	-2.13 -5.20
convergence factors:		0.65 0.58

(c) DD convergence

Figure 9.21: Unweighted partitioning scheme solving (9.18)-(9.19)



(a) Weighted partition



(b) DD convergence

dd residual	--	dd increment
iteration:	1	-0.28 -3.14
iteration:	2	-0.66 -3.42
iteration:	3	-1.02 -3.91
iteration:	4	-1.33 -4.28
iteration:	5	-1.60 -4.52
iteration:	6	-1.90 -4.79
iteration:	7	-2.14 -5.05
iteration:	8	-2.40 -5.32
iteration:	9	-2.61 -5.58
iteration:	10	-2.83 -5.82
convergence factors:		0.52 0.50

(c) DD convergence

Figure 9.22: Stiffness Matrix weighted partitioning scheme solving (9.18)-(9.19)

The values in figure parts (b) and (c) from Figures 9.21-9.22 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$ for the DD iteration. In both experiments, $||r_0|| = 119$ and $||u_0|| = 5.12 \times 10^{-3}$.

From these experiments, you can see that using parts that favor the diffusion direction improves convergence even when the direction isn't aligned with the x or y axis. The asymptotic convergence rates of the residual and u increment were 0.52 and 0.50 respectively when using Stiffness Matrix Weighting. Neither Convection Weighting nor Gradient Weighting produced similar results. Without weighting, the asymptotic convergence rates are 0.65 and 0.58 respectively. Using Stiffness

Matrix Weighting found the solution in 15 iterations while an unweighted scheme took 19 iterations. In both cases, the global mesh produced $||e_h|| = ||u - u_h|| = 5.25 \times 10^{-9}$ which means that both final global solutions have equal accuracy. Overall, using Stiffness Matrix weighting found the solution in less time.

9.2.9 Diffusion Strength

Experiment 9. *Do Edge Weighting schemes improve convergence for different strengths of anisotropic diffusion?*

We solved the same problem from Experiment 8 (section 9.2.8) with and without Stiffness Matrix weighting four more times using $\alpha = 1, 1.75, 2, 10$ and $a = \frac{1}{2}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$. Find $u \in H^2(\Omega)$ such $-\nabla \cdot a\nabla u - 1 = 0$ on Ω and $u = 0$ on $\partial\Omega$.

In these four new cases and in the original case of (9.18)-(9.19), a has the same eigenvectors. The difference is that $\lambda_1/\lambda_2 = 1, 1.75, 2, 10, 100$. This is the ratio of the diffusion in the direction $\frac{1}{\sqrt{2}}[1\ 1]^T$ versus the diffusion in the perpendicular direction. Table 9.8 shows the results.

Table 9.8: DD convergence rate versus diffusion strength

λ_1/λ_2	unweighted convergence $\|\|r_k\|\|/\|\|r_0\|\|$	weighted convergence $\|\|r_k\|\|/\|\|r_0\|\|$	unweighted convergence $\|\|\delta_k\|\|/\|\|u_0\|\|$	weighted convergence $\|\|\delta_k\|\|/\|\|u_0\|\|$	iteration reduction log/log
10^2	0.65	0.52	0.58	0.50	0.79
10^1	0.57	0.47	0.53	0.47	0.85
2	0.48	0.40	0.45	0.36	0.78
1.75	0.44	0.42	0.40	0.37	0.92
1	0.43	0.43	0.37	0.37	1.0

This shows that when the diffusion in one direction is 2 or more times more prominent than the diffusion in another, convergence rate improves by partitioning the domain with rectangle parts in the direction of the stronger diffusion. Stiffness Matrix Weighting does this automatically.

9.2.10 Diffusion Rectangle Aspect Ratio

Experiment 10. *What is the optimal rectangle aspect ratio when partitioning a mesh for an anisotropic diffusion PDE? Does the ideal aspect ratio depend on the strength of directionality? Does it depend on problem size and/or number of processors? Questions 1 and 2 are discussed in this section. Question 3 is discussed in Experiment 11 (section 9.2.11).*

We solved the same problem repeatedly with a combination of different aspect ratios, different diffusion strengths, different numbers of degrees of freedom, and different numbers of processors. We found $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} \alpha & 0 \\ 0 & 1 \end{bmatrix} \nabla u + 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.20}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.21}$$

The results from using 64 processors with each processor refined to 2.0×10^5 unknowns is displayed in Table 9.9 and Figure 9.23. (The results from other numbers of processors and problem sizes are reported in Section 9.2.5.) The results from 64 processors demonstrate that the ideal aspect ratio is dependent on diffusion directionality strength, $\alpha/1$. This data implies that the ideal aspect ratio is 2:1 when $\alpha = 2$ and 4:1 when $\alpha = 10$ or 100. When $\alpha = 1$ the diffusion is equal in all directions and the data shows that you should use squares and not rectangles (aspect ratio 1:1). The Stiffness Matrix Weighting scheme adjusts its aspect ratio to achieve the optimum each time. It produces 1:1,2:1, and 4:1 rectangles for each case as shown by the last column of Table 9.9.

Table 9.9: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on $||a||$ and rectangle aspect ratio.

$\|\|a\|\|$	1:1	2:1	3:1	4:1	5:1	6:1	16:1	25:1	stiffness
100	0.49	0.48	0.41	0.38	0.39	0.40	0.38	0.42	0.38
	1.0	0.97	0.80	0.74	0.76	0.78	0.76	0.82	0.74
10	0.48	0.40	0.37	0.35	0.36	0.39	0.46	0.57	0.36
	1.0	0.80	0.74	0.70	0.72	0.78	0.95	1.3	0.72
2	0.40	0.35	0.39	0.45	0.46	0.48	0.59	0.67	0.37
	1.0	0.87	0.97	1.15	1.18	1.25	1.74	2.29	0.92
1	0.37	0.39	-	-	0.51	-	0.69	-	0.37
	1.0	1.06	-	-	1.48	-	2.68	-	1.0

DD iteration reduction factor is calculated by $\log(r_1)/\log(r_k)$ where r_k is the convergence rate for aspect ratio k. Note that when $k = 1$, the rectangles have aspect ratio 1:1 and the resultant partition is the same as an unweighted partition. Therefore r_1 is the convergence rate of an unweighted partition.

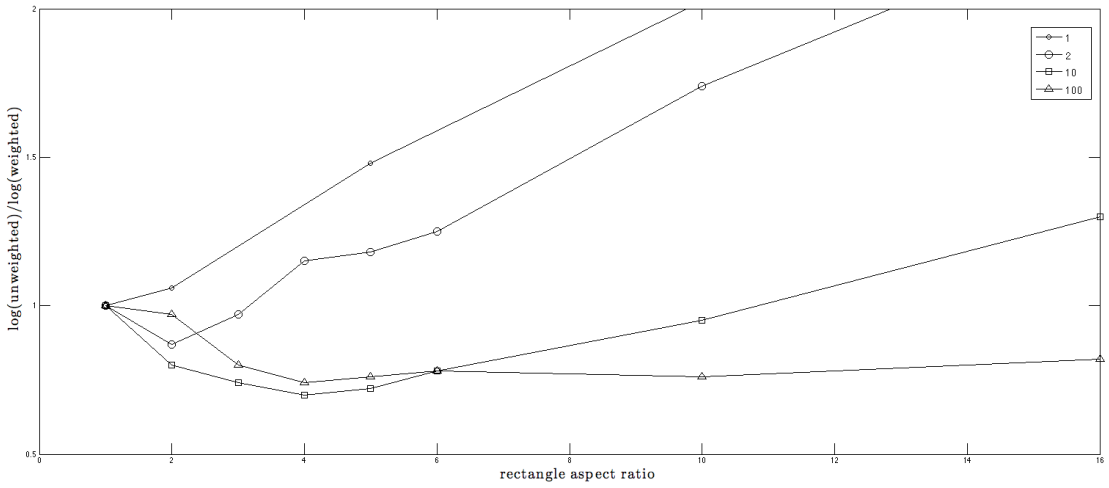
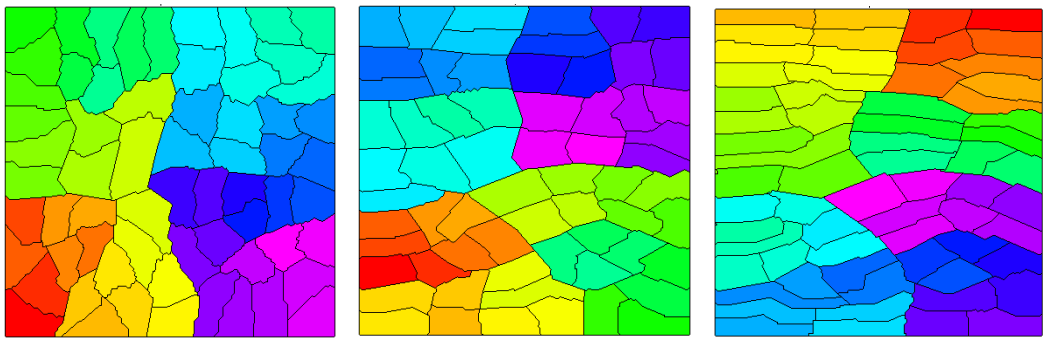


Figure 9.23: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different diffusion strength $||a||$



(a) $\alpha = 0$ (b) $\alpha = 2$ (c) $\alpha = 10, 100$

Figure 9.24: Stiffness Matrix weighting scheme adjusting based on u_{xx}/u_{yy} in PDE.

9.2.11 Diffusion Scalability

Experiment 11. *What is the optimal rectangle aspect ratio when partitioning a mesh for an anisotropic diffusion PDE? Does the ideal aspect ratio depend on the strength of directionality? Does it depend on problem size and/or number of processors? Questions 1 and 2 are discussed in Experiment 10 (section 9.2.10). Question 3 is discussed in this section.*

We solved the same problem repeatedly with a combination of different aspect ratios, different numbers of degrees of freedom, and different numbers of processors. We found $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \nabla u + 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.22}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.23}$$

The results are presented in Tables 9.10 and 9.11 and Figures 9.25 and 9.26. DD iteration reduction factor equals $\log(r_1)/\log(r_k)$ where r_k is the convergence rate for aspect ratio $k:1$. Note that 1:1 becomes an unweighted partition.

Table 9.10: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row) DD iterations reduction factor based on number of unknowns per processor and rectangle aspect ratio. All use 64 processors.

unknowns per proc	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	10:1 3.0	16:1 4.0	4:1 stiffness
2.0×10^5	0.48	0.40	0.37	0.35	0.36	0.39	0.46	0.57	0.36
	1.0	0.80	0.74	0.70	0.72	0.78	0.95	1.3	0.72
8.0×10^5	0.48	0.41	0.37	0.35	0.36	0.39	0.46	-	0.36
	1.0	0.82	0.74	0.70	0.72	0.78	0.95	-	0.72

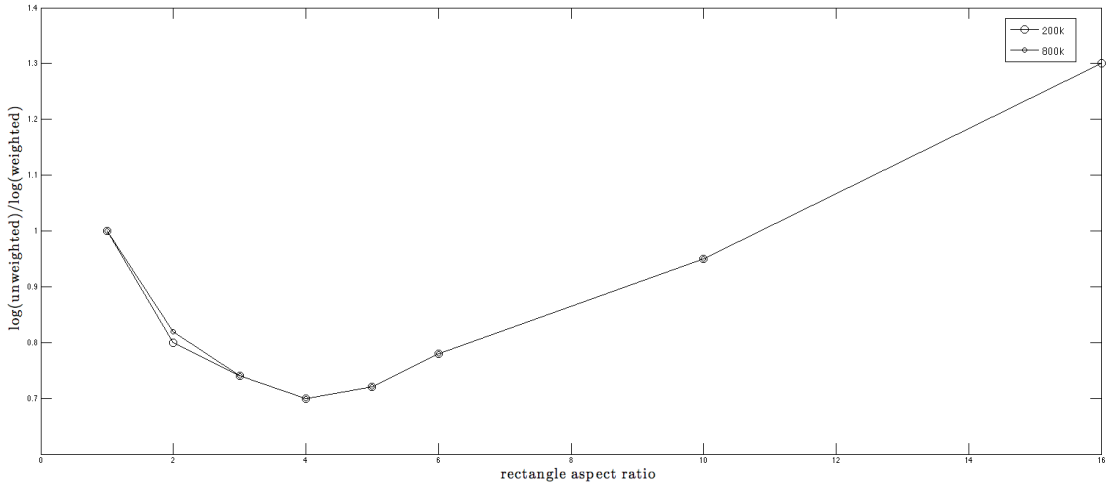


Figure 9.25: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different number of unknowns per processor.

Table 9.11: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on number of processors and rectangle aspect ratio. All use 2.0×10^5 unknowns per processor.

processors	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	10:1 3.0	16:1 4.0	4:1 stiffness
64	0.48	0.40	0.37	0.35	0.36	0.39	0.46	0.57	0.36
	1.0	0.80	0.74	0.70	0.72	0.78	0.95	1.3	0.72
512	0.56	0.47	0.43	0.42	0.42	0.43	0.53	-	0.42
	1.0	0.77	0.69	0.67	0.67	0.69	0.91	-	0.67

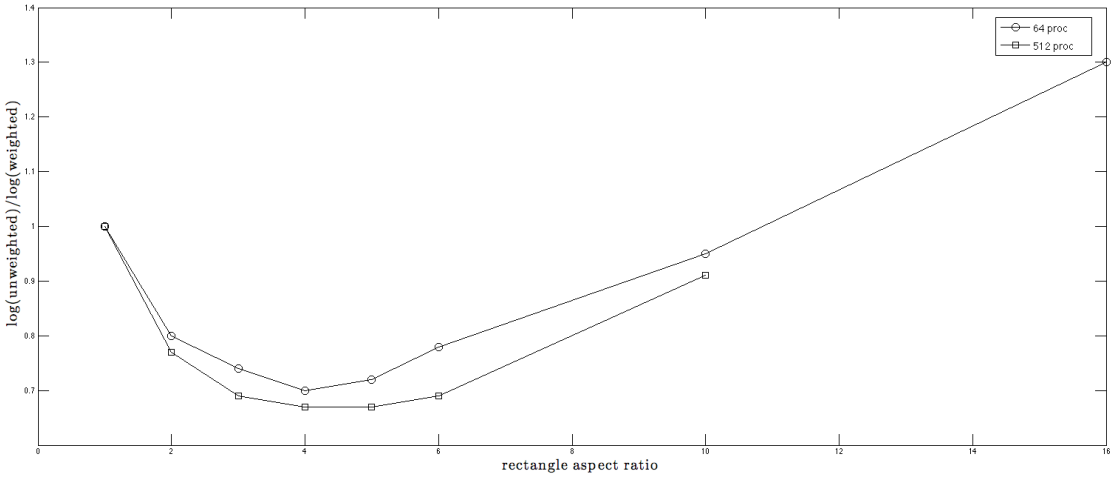


Figure 9.26: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different number of processors.

From these experiments, it appears that the optimal rectangle aspect ratio that should be used when partitioning for anisotropic diffusion is independent of the number of unknowns per processor and mostly independent (perhaps slightly dependent) of the number of processors.

9.2.12 Domain Shape

Experiment 12. *Does the shape of the domain affect the success of using Convection, Gradient, and Stiffness Matrix weighting?*

We solved a convection dominated PDE on the domain pictured in Figure 9.28a. Find $u \in H^2(\Omega)$ such that

$$-\triangle u + 10^4 \left[-\frac{y}{\sqrt{x^2+y^2}} \quad \frac{x}{\sqrt{x^2+y^2}} \right]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega \tag{9.24}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.25}$$

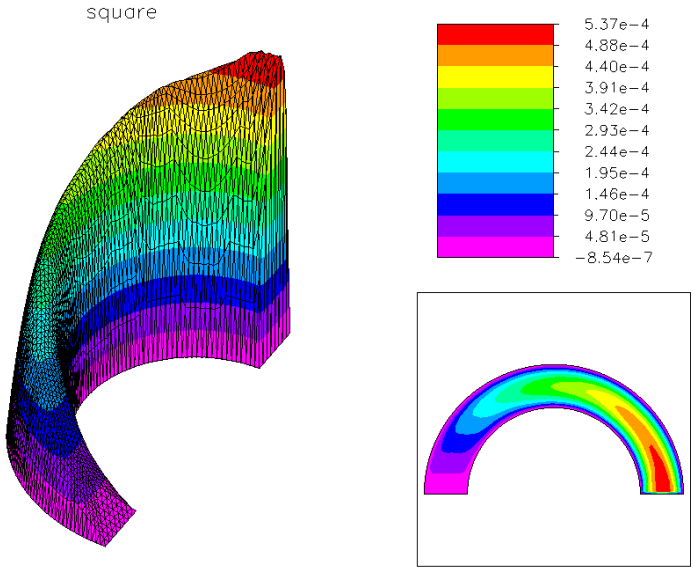
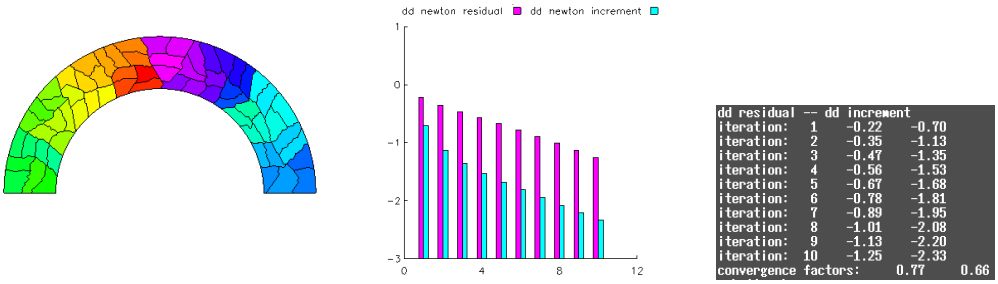
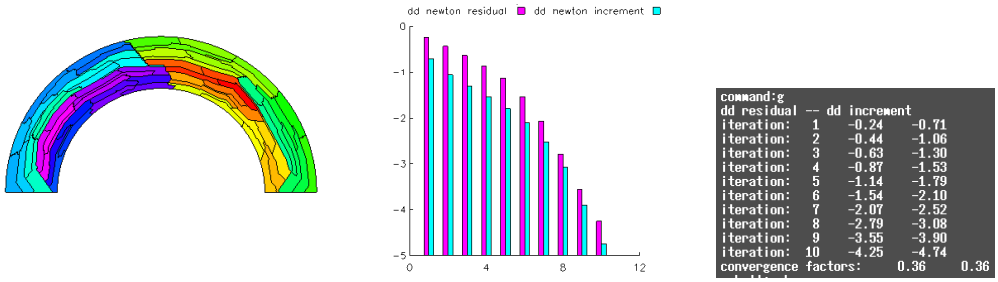


Figure 9.27: Solution to equation (9.24)-(9.25)



(a) Unweighted partition (b) DD convergence (c) DD convergence

Figure 9.28: Unweighted partitioning scheme solving (9.24)-(9.25)



(a) Weighted partition (b) DD convergence (c) DD convergence

Figure 9.29: Convection/ Stiffness Matrix weighted partitioning scheme solving (9.25)-(9.25)

The values in figure parts (b) and (c) from Figures 9.27-9.29 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. In both cases, $||r_0|| = 96.2$ and $||u_0|| = 2.44 \times 10^{-2}$.

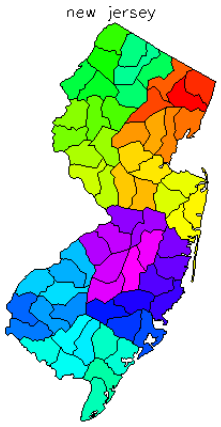
From this experiment, you can see that using parts that favor the convection direction improves convergence even when the domain isn't a square. The asymptotic convergence rates of the residual and u increment were 0.36 and 0.36 respectively when using Convection or Stiffness Matrix Weighting with a convection parameter of $s = 3$. Without weighting, the asymptotic convergence rates are 0.77 and 0.63 respectively. In both cases, the global mesh produced $||e_h|| = ||u - u_h|| = 4.18 \times 10^{-6}$ which means that both final global solutions have equal accuracy.

A reasonable DD Method iteration stopping criteria is when $||\delta u_k|| = ||u_k - u_{k-1}|| < \frac{1}{10}||e_h||$. According to this criteria, it took the unweighted scheme 25 iterations to converge while it took the weighted scheme 10 iterations. Overall, the weighted scheme found the same final solution in less time.

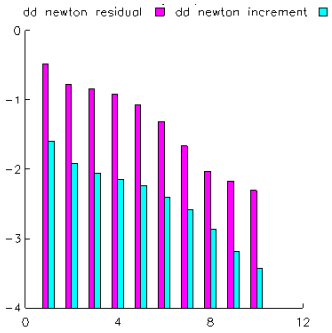
Next, we solved a convection dominated PDE on the domain pictured in Figure 9.30a. Find $u \in H^2(\Omega)$ such that

$$-\triangle u + 10^4 \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}^T \cdot \nabla u - 1 = 0 \text{ on } \Omega \tag{9.26}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.27}$$



(a) Unweighted partition

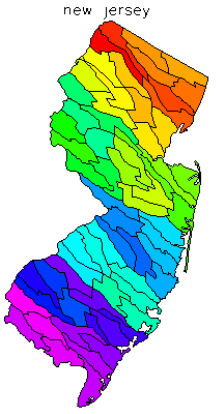


(b) DD convergence

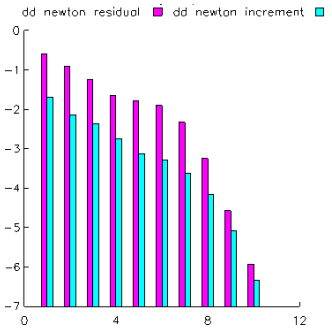
dd residual	--	dd increment
iteration: 1	-0.48	-1.60
iteration: 2	-0.77	-1.92
iteration: 3	-0.85	-2.06
iteration: 4	-0.92	-2.14
iteration: 5	-1.07	-2.23
iteration: 6	-1.31	-2.40
iteration: 7	-1.67	-2.59
iteration: 8	-2.03	-2.87
iteration: 9	-2.17	-3.19
iteration: 10	-2.31	-3.42
convergence factors:	0.63	0.63

(c) DD convergence

Figure 9.30: Unweighted partitioning scheme solving (9.26)-(9.27)



(a) Weighted partition



(b) DD convergence

dd residual	--	dd increment
iteration: 1	-0.60	-1.69
iteration: 2	-0.91	-2.13
iteration: 3	-1.25	-2.37
iteration: 4	-1.66	-2.76
iteration: 5	-1.78	-3.13
iteration: 6	-1.90	-3.29
iteration: 7	-2.33	-3.63
iteration: 8	-3.24	-4.17
iteration: 9	-4.57	-5.09
iteration: 10	-5.93	-6.34
convergence factors:	0.26	0.30

(c) DD convergence

Figure 9.31: Convection/ Stiffness Matrix weighted partitioning scheme solving (9.26)-(9.27)

The values in figure parts (b) and (c) from Figures 9.27-9.29 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. In both cases, $||r_0|| = 8.19$ and $||u_0|| = 2.06 \times 10^{-3}$.

From this experiment, you can see that using parts that favor the convection direction improves convergence even when the domain isn't a square. The asymp-

totic convergence rates of the residual and u increment were 0.26 and 0.30 respectively when using Convection or Stiffness Matrix Weighting. Without weighting, the asymptotic convergence rates are 0.63 and 0.63 respectively. In both cases, the global mesh produced $||e_h|| = ||u - u_h|| = 1.68 \times 10^{-8}$ which means that both final global solutions have equal accuracy.

A reasonable DD Method iteration stopping criteria is when $||\delta u_k|| = ||u_k - u_{k-1}|| < \frac{1}{10}||e_h||$. According to this criteria, it took the unweighted scheme 24 iterations to converge while it took the weighted scheme 10 iterations. Overall, the weighted scheme found the same final solution in less time.

9.2.13 Domain Scalability

Experiment 13. *When partitioning a mesh for convection dominated or anisotropic diffusion PDEs, does the optimal subdomain rectangle aspect ratio depend on the aspect ratio of the domain?*

We solved a convection dominated PDE repeatedly on a variety of rectangle domains with different aspect ratios. We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^5 \ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0, \kappa] \times [0, 1] \in \mathbb{R}^2 \tag{9.28}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.29}$$

We altered the domain by varying $\kappa = 0.25, 1, 2, 4, 8$. All solves use 2.0×10^5 unknowns per processor and 64 processors. The results are presented in Table 9.12 and Figure 9.32. DD iteration reduction factor equals $\log(r_0)/\log(r_k)$ where r_k is the convergence rate for $s = k$. Note that $s = 0$ has an aspect ratio of 1:1 and becomes an unweighted partition.

Table 9.12: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on domain and subdomain aspect ratio.

κ	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	10:1 3.0	16:1 4.0	25:1 5.0
0.25	0.18	0.13	0.08	0.10	0.09	-	0.09	-	-
	1.0	0.84	0.68	0.74	0.71	-	0.71	-	-
1	0.34	0.28	0.23	0.22	0.25	0.25	-	0.26	0.25
	1.0	0.85	0.73	0.71	0.78	0.78	-	0.80	0.78
2	0.52	0.33	0.31	0.29	0.27	0.28	0.21	0.22	0.24
	1.0	0.59	0.56	0.53	0.50	0.51	0.42	0.43	0.46
4	0.61	0.56	0.50	0.45	0.39	0.36	0.31	0.25	0.28
	1.0	0.85	0.71	0.62	0.52	0.48	0.42	0.36	0.39
8	0.63	-	0.59	-	0.57	0.51	0.43	0.42	-
	1.0	-	0.88	-	0.82	0.69	0.55	0.53	-

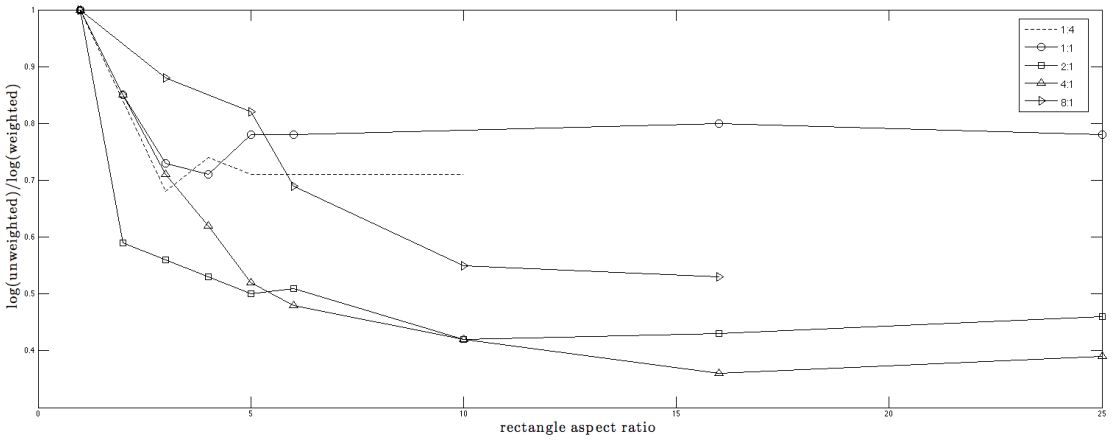


Figure 9.32: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents a different domain aspect ratio for convection dominated PDE.

Next, we solved an anisotropic diffusion problem repeatedly on a variety of rectangle domains with different aspect ratios. We found $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \nabla u + 1 = 0 \text{ on } \Omega = [0, \kappa] \times [0, 1] \in \mathbb{R}^2 \tag{9.30}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.31}$$

We altered the domain by varying $\kappa = 0.25, 1, 2, 4$. All solves use 2.0×10^5 unknowns per processor and 64 processors. The results are presented in Table 9.13 and Figure 9.33. DD iteration reduction factor equals $\log(r_0)/\log(r_k)$ where r_k is the convergence rate for $s = k$. Note that $s = 0$ has an aspect ratio of 1:1 and becomes an unweighted partition.

Table 9.13: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on domain and subdomain aspect ratio.

κ	1:1 s=0.0	2:1 1.0	3:1 1.56	4:1 1.83	5:1 2.0	6:1 2.23	10:1 3.0	16:1 4.0
0.25	0.37	0.31	0.29	0.30	0.29	-	-	0.36
	1.0	0.85	0.80	0.83	0.80	-	-	0.93
1	0.48	0.40	0.37	0.35	0.36	0.39	0.46	0.57
	1.0	0.80	0.74	0.70	0.72	0.78	0.95	1.3
4	0.47	0.41	-	0.35	0.34	-	-	0.53
	1.0	0.85	-	0.72	0.70	-	-	1.19

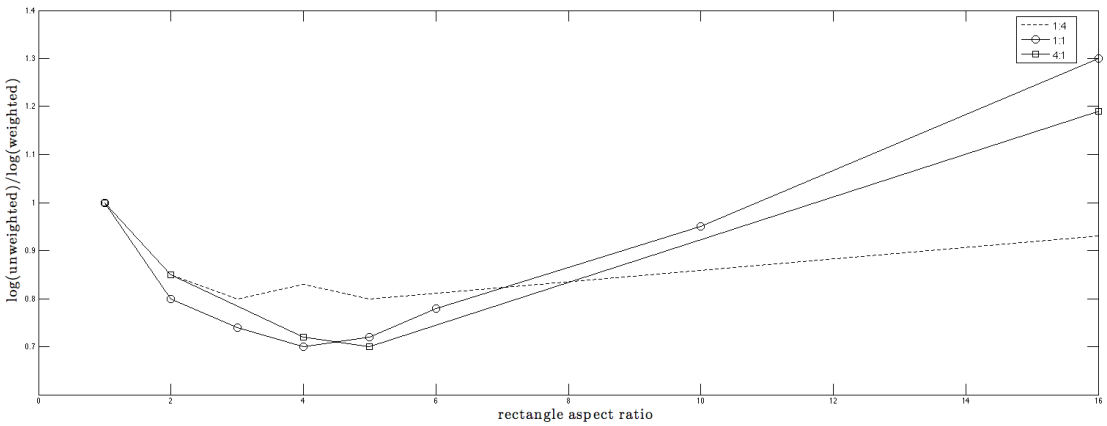


Figure 9.33: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents different domain aspect ratio for anisotropic diffusion PDE.

From these experiments, it appears that the optimal rectangle aspect ratio that should be used when partitioning for anisotropic diffusion is mostly independent (slightly dependent) of domain aspect ratio. And, the optimal rectangle aspect ratio that should be used when partitioning for convection dominated PDEs is notably dependent on domain aspect ratio.

If you keep the rectangle aspect constant at our previously determined optimum of 4:1, then the DD iteration reduction factor would approach 1 as you solve PDEs with increasingly elongated domains. These dependencies appear to be the same dependencies we saw in Experiment 7 and Experiment 11 (sections 9.2.7 and 9.2.11). If fact, Figure 9.32 looks very much like Figure 9.15 and Figure 9.33 looks very much like Figure 9.26.

For convection dominated PDEs, It appears that rectangle aspect ratio is dependent on the number of subdomains that must be traversed to move across the domain in the direction of convection. Both using more processors and/or having a domain with a large aspect ratio in the direction of convection increases this number and degrades the DD convergence rate. For the Edge Weighting schemes presented in this thesis to be most effective in improving the convergence rate, larger subdomain rectangle aspect ratios need to be used in these situations.

It also appears that Error and Flow Weighting do not suffer this dependence (to be seen in Experiment 18 in Section 9.3.5). This is probably because Error and Flow Weighting reduce the number of subdomains that need to be crossed logarithmically while Convection, Gradient, and Stiffness Matrix weighting only reduce the number linearly. This is explored more in Experiment 20 (section 9.5).

9.3 Vertex Weighting Experiments

The Vertex Weighting techniques explained in Chapter 7 include Error Weighting and Flow Weighting. In this Section, we describe the results of experiments conducted to test their performance on various PDEs, various domains, and various problem sizes. Assume all norms without subscripts to be L^2 or ℓ^2 norms.

9.3.1 Convection with Boundary Layer

Experiment 14. *Does using Error and Flow Weighting improve the convergence of Bank-Holst paradigm DD solver when solving convection dominated elliptic partial differential equations?*

For our first vertex weighting test, we solve the same first problem that we tested the Edge Weighting schemes with in Experiment 1 (section 9.2.1). Therefore you can compare these results to those. Find $u \in H^2(\Omega)$ such that

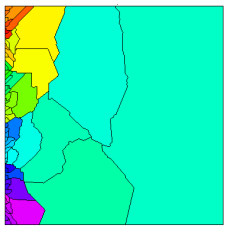
$$-\triangle u - [10^6\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.32}$$

$$u = 0 \text{ on } \partial\Omega_D \tag{9.33}$$

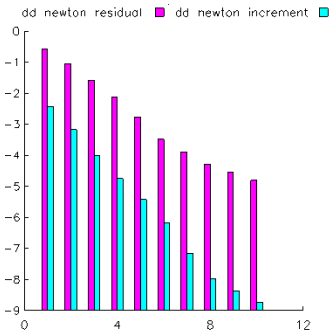
$$n \cdot \nabla u = 0 \text{ on } \partial\Omega_N \tag{9.34}$$

$\partial\Omega_D$ is the left and right side of the unit square ($x = 0, 1$) while $\partial\Omega_N$ is the top and bottom ($y = 0, 1$).

Because the solution has a boundary layer along the y axis, the finite element solution on a uniform mesh has much error located there in the form $||u - u_h||$. Therefore, when partitioning for this problem, both Error Weighting and Flow Weighting create the same partition when the Flow Weighting parameters are set to $z(x, y) = x + 10^{-3}$, $s = 2$, $\alpha = 1$, and $\beta = 0$. Each weighting scheme made small parts near the y axis and bigger parts as you move in the x direction. We partitioned this domain into 64 parts starting from a uniform mesh of 2.0×10^4 unknowns (4.0×10^4 triangles). Afterward, each part refined their mesh uniformly to 2.0×10^5 unknowns. The total problem size was around 10^7 degrees of freedom.



(a) Weighted partition

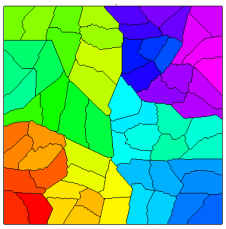


(b) DD convergence

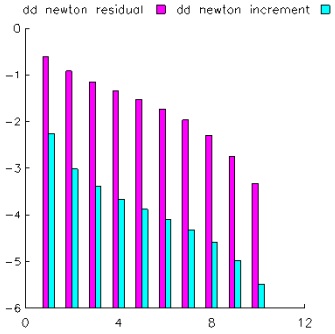
```
command:g
dd residual -- dd increment
iteration:   1    -0.59    -2.43
iteration:   2    -1.05    -3.17
iteration:   3    -1.59    -4.00
iteration:   4    -2.13    -4.75
iteration:   5    -2.77    -5.43
iteration:   6    -3.48    -6.19
iteration:   7    -3.89    -7.15
iteration:   8    -4.28    -7.98
iteration:   9    -4.53    -8.36
iteration:  10    -4.80    -8.74
convergence factors:     0.34     0.20
```

(c) DD convergence

Figure 9.34: Error and Flow weighted partitioning scheme solving (9.32)-(9.34)



(a) Unweighted partition



(b) DD convergence

dd residual -- dd increment			
iteration:	1	-0.61	-2.26
iteration:	2	-0.92	-3.01
iteration:	3	-1.14	-3.39
iteration:	4	-1.34	-3.67
iteration:	5	-1.52	-3.88
iteration:	6	-1.73	-4.10
iteration:	7	-1.97	-4.32
iteration:	8	-2.30	-4.59
iteration:	9	-2.75	-4.97
iteration:	10	-3.33	-5.48
convergence factors:		0.50	0.44

(c) DD convergence

Figure 9.35: Unweighted partitioning scheme solving (9.32)-(9.34)

The values in figure parts (b) and (c) from Figures 9.3-9.5 are logs of the relative residual, $\log_{10}(||r_k||/||r_0||)$, and logs of the relative u increment, $\log_{10}(||u_k - u_{k-1}||/||u_0||)$, for the DD iteration. For the unweighted partition, $||r_0|| = 0.66$ and $||u_0|| = 6.5 \times 10^{-5}$. For the weighted partition, $||r_0|| = 0.37$ and $||u_0|| = 1.5 \times 10^{-4}$

From these experiments, you can see that using parts that facilitate information travel in the convection direction improves convergence. The asymptotic convergence rates of the residual and u increment were 0.34 and 0.20 respectively when using Error or Flow Weighting. Using Error or Flow Weighting speeds up convergence in the same fashion as using Convection, Gradient, or Stiffness Matrix Weighting did. See Experiment 1 (section 9.2.1). This is what we expected.

Without weighting, the asymptotic convergence rates are 0.50 and 0.44 respectively. The global mesh from the unweighted partition produced $||e_h|| = ||u - u_h|| = 2.9 \times 10^{-9}$ and the global mesh from the weighted partition produced $||e_h|| = ||u - u_h|| = 1.17 \times 10^{-9}$. Since there was a boundary layer in the solution of this convection-diffusion equation, using Error or Flow Weighting reduced the error of the final solution by a factor of 2.5 compared to an unweighted scheme.

A reasonable DD Method iteration stopping criteria is when $||\delta u_k|| = ||u_k - u_{k-1}|| < \frac{1}{10}||e_h||$. According to this criteria, it took the unweighted scheme 10 iterations to converge while it took the weighted scheme 6 iterations. Overall, the weighted scheme found a solution in less time that had an error 2.5 times less also.

9.3.2 Convection without Boundary Layer

Experiment 15. *Does using Error and Flow Weighting improve the convergence of Bank-Holst paradigm DD solver when solving convection dominated elliptic partial differential equations whose solutions don't have a boundary layer?*

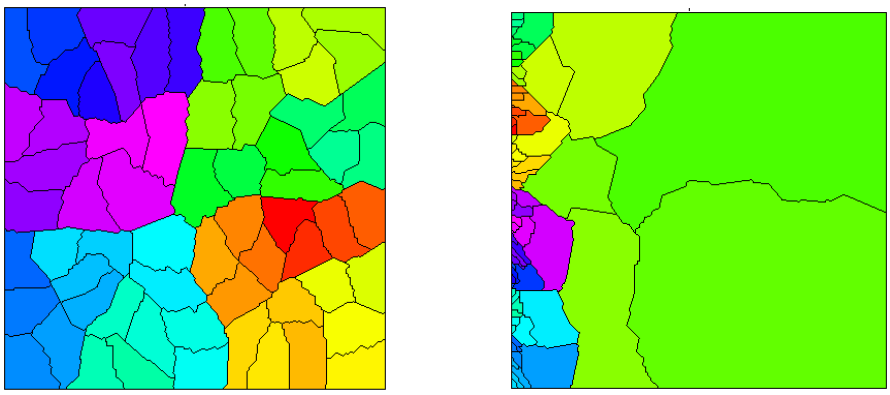
We solve the same problem that we tested in Experiment 14 (section 9.3.1) but add a forcing function that removes the boundary layer in the solution. We use the same flow function, $z(x, y) = x + 10^{-3}$ and parameters $s = 2$, $\alpha = 1$, and $\beta = 0$. Find $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^4 \ 0]^T \cdot \nabla u - f(x, y) = 0 \text{ on } \Omega = [0, 2\pi] \times [0, 2\pi] \in \mathbb{R}^2 \tag{9.35}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.36}$$

$$f(x, y) = -2\sin(x)\sin(y) + 10^4 \cos(x)\sin(y) \tag{9.37}$$

The exact solution is $u(x, y) = \sin(x)\sin(y)$ and is shown in Figure 9.12. The results of this experiment are very similar to those in Experiment 14 (section 9.3.1) with one difference. The Error Weighting scheme did not improve convergence compared to an unweighted scheme. This is expected because the final finite element solution has uniform error on a uniform mesh as a result of not having a boundary layer. Thus, the Error Weighting scheme behaves the same as the unweighted scheme.



(a) Error weighting partition (b) Flow weighting partition

Figure 9.36: Error versus Flow Weighting without boundary layer.

Using Flow Weighting improved convergence similar to Experiment 14 (section 9.3.1). Using the Flow Weighting scheme found the solution in 7 iterations while the unweighted scheme took 12 iterations. However, the accuracy of the final solution is less.

For both partitions, $||r_0|| = 6.5 \times 10^4$ and $||u_0|| = 4.5$. The asymptotic convergence rates of the residual and u increment were 0.20 and 0.14 respectively when using Flow Weighting. The Error Weighting or unweighted scheme had convergent rates of 0.48 and 0.41. The weighted partition's final solution had error $||e_h|| = ||u - u_h|| = 1.1 \times 10^{-4}$ while the unweighted partition's solution had error $||e_h|| = 8.0 \times 10^{-5}$. The Flow weighted scheme achieved 2.2× faster convergence but also suffered 33% more error.

9.3.3 Flow Function Placement

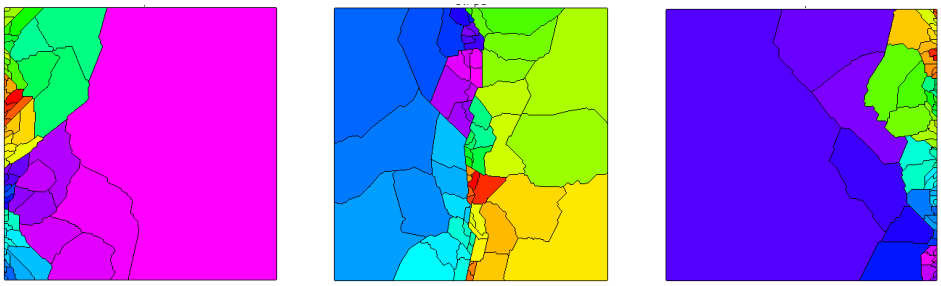
Experiment 16. *Do the Flow and Error Weighting schemes need to be aligned with convection and/or anisotropic diffusion to achieve improved convergence?*

From the previous experiments, we see that Error Weighting and Flow Weighting improve the convergence of convection dominated problems. We speculate that these methods do this by helping information travel similar to how rectangle parts helped in the Edge Weighting schemes because the improved convergence of the Vertex Weighting schemes are similar to those of the Edge Weighting schemes. But the improved convergence can also be a result of another reason such as the presence of small parts or the presence of many parts on the boundary of our domain or something we didn't think about.

To test this, we will move the Flow function around. We will solve (9.32)-(9.34) with various Flow functions. Each test uses parameters $s = 1$, $\alpha = 1$, $\beta = 0$, and $\epsilon = 10^{-3}$.

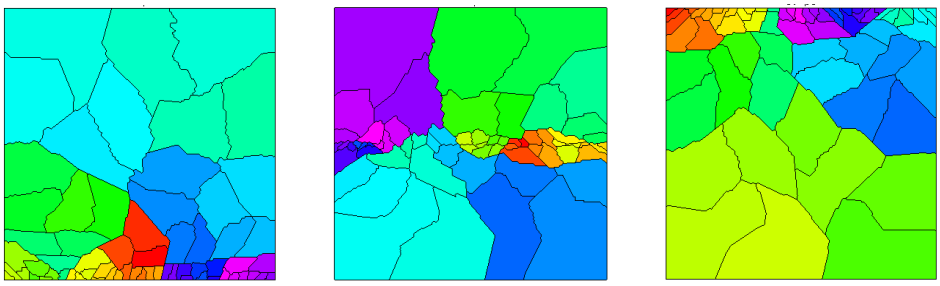
Table 9.14: DD convergence rates for different flow functions

flow function	convergence $\|\|r_k\|\|/\|\|r_0\|\|$	convergence $\|\|\delta_k\|\|/\|\|u_0\|\|$	figure
$z(x,y) = x + \epsilon$	0.36	0.21	9.37a
$z(x,y) = \|x - 0.5\| + \epsilon$	0.35	0.29	9.37b
$z(x,y) = \|x - 1\| + \epsilon$	0.42	0.26	9.37c
$z(x,y) = 1$	0.50	0.44	9.35
$z(x,y) = y + \epsilon$	0.70	0.51	9.38a
$z(x,y) = \|y - 0.5\| + \epsilon$	0.68	0.53	9.38b
$z(x,y) = \|y - 1\| + \epsilon$	0.70	0.51	9.38c



(a) $z(x,y) = x + \epsilon$ (b) $z(x,y) = |x - 0.5| + \epsilon$ (c) $z(x,y) = |x - 1| + \epsilon$

Figure 9.37: Flow Weighting with $z(x,y) = |x - c| + \epsilon$ solving (9.32)-(9.34)



(a) $z(x,y) = y + \epsilon$ (b) $z(x,y) = |y - 0.5| + \epsilon$ (c) $z(x,y) = |y - 1| + \epsilon$

Figure 9.38: Flow Weighting with $z(x,y) = |y - c| + \epsilon$ solving (9.32)-(9.34)

When you compare the results in Table 9.14 to the Edge Weighting schemes in Experiment 1 (section 9.2.1) solving the same problem, you notice that Flow Weighting with $z(x,y) = |x - c| + \epsilon$ is behaving like Convection Weighting (see Figure 9.4) and Flow Weighting with $z(x,y) = |y - c| + \epsilon$ is behaving like Perpendicular Convection Weighting (see Figure 9.5). This confirms that Flow Weighting

and Error Weighting have a direction in which they help information to travel. Additionally, Table 9.15 from the Experiment 17 (section 9.3.4), shows that Flow and Error Weighting don't improve convergence in the absence of both convection and anisotropic diffusion.

9.3.4 Flow Function Parameter

Experiment 17. *What is the optimal Flow Function parameter s in $z^{-s}(x,y)$ when using Flow Weighting? Does the optimal value depend on convection strength? Does it depend on problem size and/or number of processors? Does it depend on domain aspect ratio? Questions 1 and 2 are discussed in this section. Questions 3 and 4 are discussed in Experiment 18 (section 9.3.5).*

We solved the same problem repeatedly with a combination of different flow parameters, different convection strengths, different numbers of degrees of freedom, and different numbers of processors. We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.38}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.39}$$

We used flow function $z(x,y) = x + 10^{-3}$. The results from using 64 processors with each processor refined to 2.0×10^5 unknowns is displayed in Table 9.15 and Figure 9.39. (The results from other numbers of processors and problem sizes are reported in Section 9.3.5.) The results from 64 processors demonstrate that the ideal flow parameter is independent of convection strength.

From this data, I would say that the ideal flow parameter is between 1.5 and 2.0 inclusive. You want the smallest flow parameter that produces satisfactory convergence rates. When using flow weighting on a problem whose finite element solution has uniform error $||u - u_h||_{L^2(t_k)} \approx c\ \forall t_k$ on a uniform mesh, increasing the flow parameter decreases final solution accuracy. Theorem 7.2.2 describes this and Table 7.3 displays the ratio of error increase for common s values.

It is interesting to note that when $||b||h = 0$ corresponding to a pure diffusion PDE, using Flow Weighting does not degrade DD convergence. This differs from Convection Weighting which does degrade convergence when convection is absent.

Table 9.15: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on $||b||h$ and flow weighting parameter s.

$\|\|b\|\|h$	$\|\|b\|\|$	s=0.0	0.5	0.75	1.0	1.5	2.0	2.25	2.5	3.0
347	10^6	0.34	0.31	0.27	0.25	0.19	0.15	0.18	0.25	0.24
		1.0	0.92	0.82	0.78	0.65	0.57	0.63	0.78	0.76
34.7	10^5	0.34	0.31	0.28	0.20	0.09	0.09	0.08	0.12	0.08
		1.0	0.92	0.85	0.67	0.45	0.45	0.43	0.51	0.43
3.47	10^4	0.30	0.27	0.25	0.19	0.09	0.07	-	0.05	0.06
		1.0	0.92	0.87	0.72	0.5	0.45	-	0.4	0.43
0.347	10^3	0.14	-	-	0.12	0.12	0.13	-	-	-
		1.0	-	-	0.93	0.93	0.96	-	-	-
0.0	0.0	0.37	-	0.38	-	-	0.33	-	0.35	-
		1.0	-	1.03	-	-	0.90	-	0.95	-

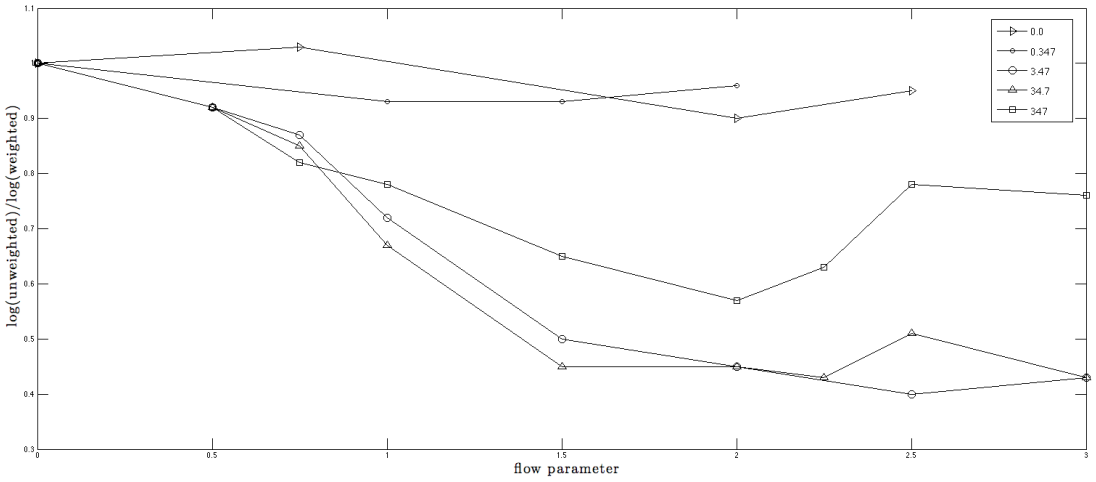
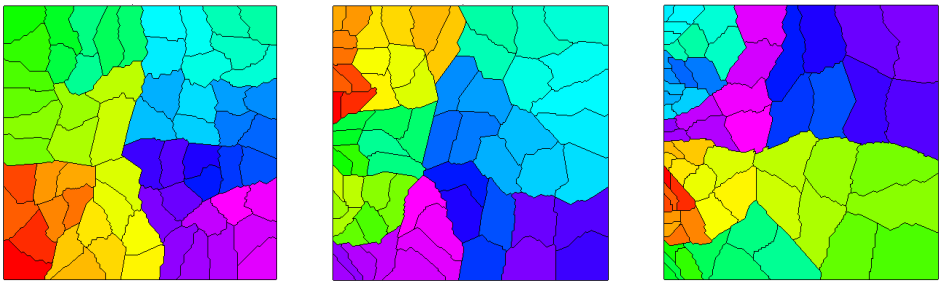
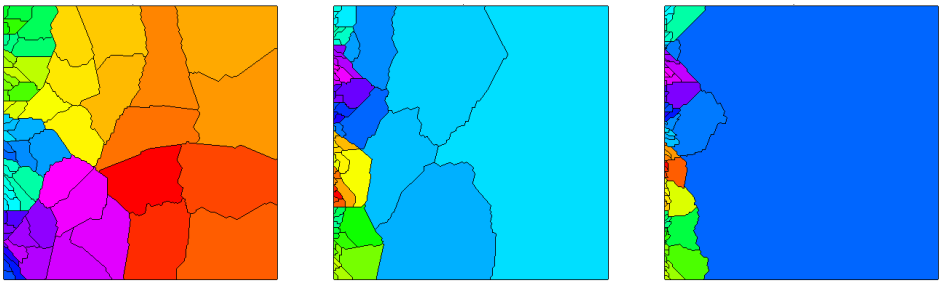


Figure 9.39: The factor of DD iteration reduction versus flow parameter. Each line represents using a different convection strength $||b||h$



(a) $s = 0.0$ (b) $s = 0.5$ (c) $s = 0.75$



(d) $s = 1.0$ (e) $s = 1.5$ (f) $s = 2.0$

Figure 9.40: Different flow parameter s values partitioning the unit square into 64 parts.

9.3.5 Vertex Weighting Scalability

Experiment 18. *What is the optimal flow function parameter s in $z^{-s}(x, y)$ when using Convection, Gradient, and Stiffness Matrix weighting? Does the ideal aspect ratio depend on convection strength? Does it depend on problem size and/or number of processors? Does it depend on domain aspect ratio? Questions 1 and 2 were discussed in section 9.3.4. Questions 3 and 4 are discussed in this section.*

We solved the same problem repeatedly with a combination of different flow parameters, different numbers of degrees of freedom, and different numbers of processors while keeping $||b||h = 34.2$ constant. For the base cases of 2.0×10^5 in Table 9.16 and 64 processors in Table 9.3.5 this is accomplished by letting $\beta = 10^5$. Then as you double the number of unknowns or double the number of processors, you increase β by a factor of $\sqrt{2}$. We used flow function $z(x, y) = x + 10^{-3}$

We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0, 1] \times [0, 1] \in \mathbb{R}^2 \tag{9.40}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.41}$$

Table 9.16: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on number of unknowns per processor and flow weighting parameter s. All use 64 processors.

unknowns per proc	s=0.0	0.5	0.75	1.0	1.5	2.0	2.5
2.0×10^5	0.34	0.31	0.28	0.20	0.09	0.09	0.12
	1.0	0.92	0.85	0.67	0.45	0.45	0.51
4.0×10^5	0.37	0.33	0.29	0.22	0.10	0.07	0.07
	1.0	0.90	0.80	0.66	0.43	0.37	0.37
8.0×10^5	0.38	0.36	-	0.23	0.10	0.07	0.07
	1.0	0.95	-	0.66	0.42	0.36	0.36

When $s = 0$, the partition is the same as using no weighting. Therefore the factor of DD iteration reduction equals $\log(r_0)/\log(r_k)$ where r_k is the convergence for $s = k$. These factors are displayed in Figure 9.41 and Table 9.16 beneath the convergence rates.

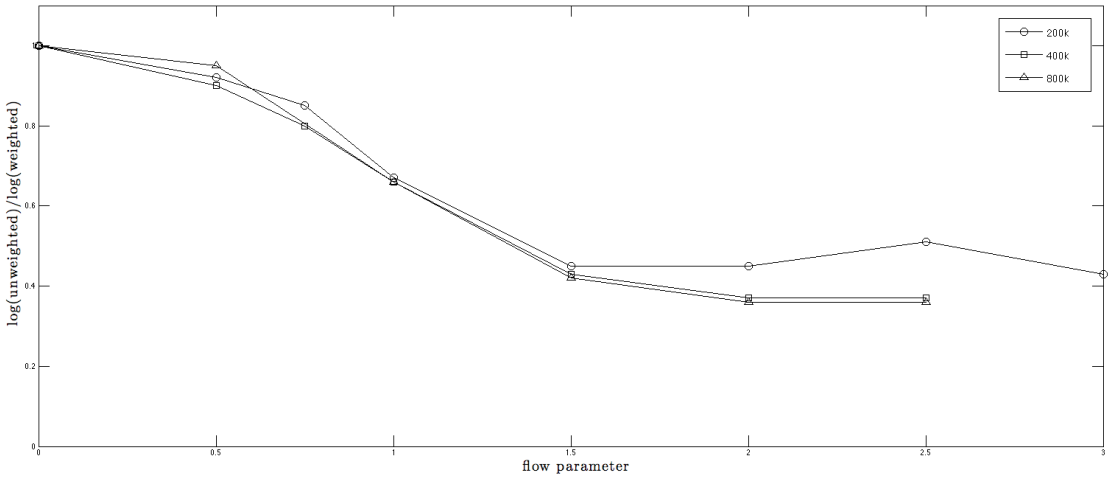


Figure 9.41: The factor of DD iteration reduction versus flow weighting parameter. Each line represents using a different number of unknowns per processor.

From this experiment, it appears that the number of degrees of freedom per processor does not affect the optimal flow parameter of $1.5 \leq s \leq 2.0$ that we concluded in Experiment 17 (section 9.3.4).

Table 9.17: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on number of processors and flow weighting parameter s. All use 2.0×10^5 unknowns per processor.

processors	s=0.0	0.5	0.75	1.0	1.5	2.0	2.5
64	0.34	0.31	0.28	0.20	0.09	0.09	0.12
	1.0	0.92	0.85	0.67	0.45	0.45	0.51
128	0.52	0.49	0.43	0.30	0.14	0.14	-
	1.0	0.92	0.77	0.54	0.33	0.38	-
256	0.64	0.64	0.60	0.51	0.20	0.24	-
	1.0	1.0	0.87	0.66	0.28	0.31	-
512	0.70	0.68	0.66	0.63	0.37	0.18	-
	1.0	0.92	0.86	0.77	0.36	0.21	-

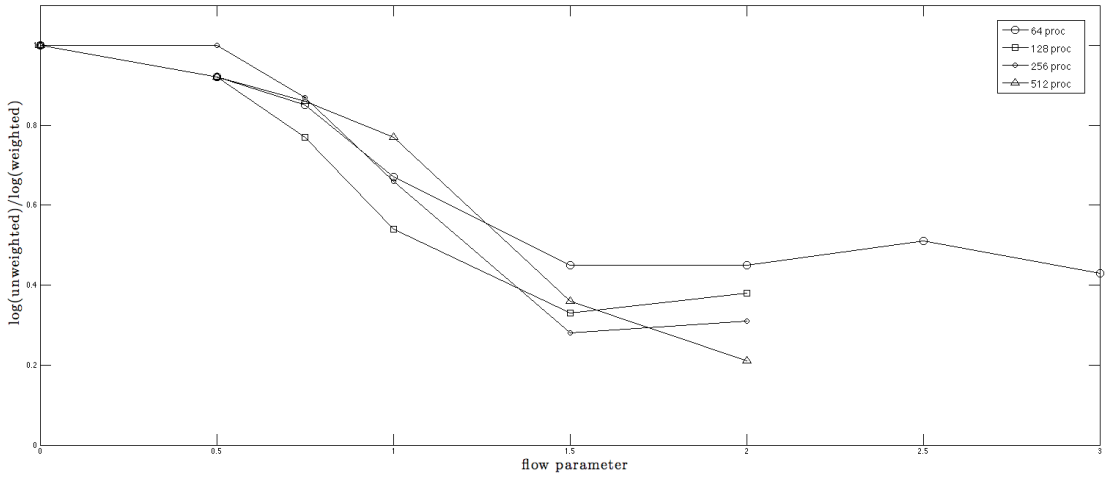
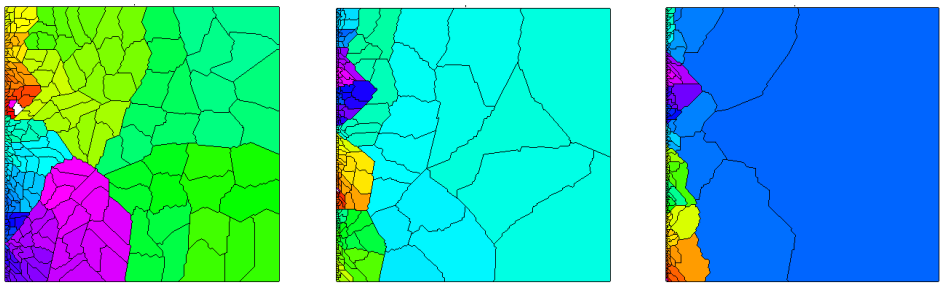


Figure 9.42: The factor of DD iteration reduction versus rectangle aspect ratio. Each line represents using a different number of processors.

From this experiment, it appears that the number of processors does not affect the optimal flow parameter of $1.5 \leq s \leq 2.0$ that we concluded in Experiment 17 (section 9.3.4). This differs from the Convection Weighting parameter which seemed to be dependent on number of processors. Figure 9.40 displays what different s partitions look like when dividing the unit square into 64 parts. Figure 9.43 pictures some s partitions into 256 parts.



(a) $s = 1$ (b) $s = 1.5$ (c) $s = 2$

Figure 9.43: Different s values partitioning the unit square into 256 parts.

Next, we solved a convection dominated PDE repeatedly on a variety of rectangle domains with different aspect ratios using Flow Weighting and flow function $z(x, y) = x + 10^{-3}$. We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^5\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0, \kappa] \times [0, 1] \in \mathbb{R}^2 \tag{9.42}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.43}$$

We altered the domain by varying $\kappa = 0.25, 1, 2, 4, 8$. All solves use 2.0×10^5 unknowns per processor and 64 processors. The results are presented in Table 9.18 and Figure 9.44. DD iteration reduction factor equals $\log(r_0)/\log(r_k)$ where r_k is the convergence rate for $s = k$. Note that $s = 0$ has an aspect ratio of 1:1 and becomes an unweighted partition.

Table 9.18: (1^{st} row:) DD convergence rate of $||\delta u_k||$ and (2^{nd} row:) DD iterations reduction factor based on domain and flow weighting parameter s.

κ	s=0.0	0.75	1.0	1.5	2.0	2.5
0.25	0.18	0.13	0.12	0.06	0.13	0.11
	1.0	0.84	0.81	0.61	0.84	0.78
1	0.34	0.28	0.20	0.09	0.09	0.12
	1.0	0.85	0.67	0.45	0.45	0.51
2	0.52	0.48	0.28	0.20	0.08	0.08
	1.0	0.89	0.51	0.41	0.26	0.26
4	0.61	0.59	0.43	0.17	0.10	0.10
	1.0	0.94	0.59	0.28	0.21	0.21
8	0.63	0.62	0.54	0.22	0.12	0.07
	1.0	0.97	0.75	0.31	0.22	0.17

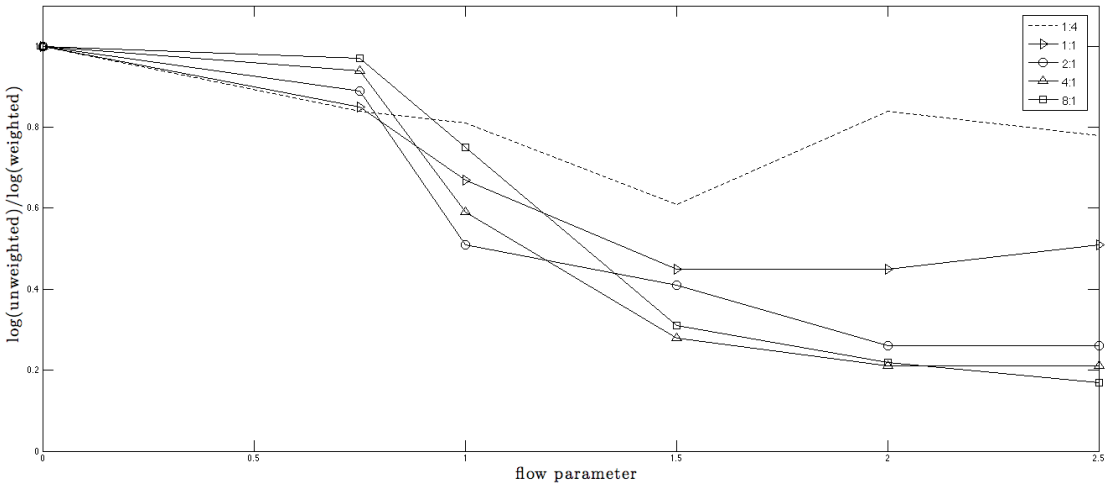


Figure 9.44: The factor of DD iteration reduction versus flow weighting parameter. Each line represents a different domain aspect ratio for convection dominated PDE.

From this experiment, it appears that the aspect ratio of the domain does not affect the optimal flow parameter of $1.5 \leq s \leq 2.0$ that we concluded in Experiment 17 (section 9.3.4). This differs from the Convection Weighting parameter which seemed to be dependent on domain aspect ratio (as seen in Experiment 13 in Section 9.2.13).

It is interesting to note that Figure 9.44 looks very similar to Figure 9.42 which suggests that increasing the number of processors used to solve a convection dominated PDE has a similar effect as increasing the domain aspect ratio in the direction of convection. We also saw this correspondence in Experiment 13 (section 9.2.13) when using Convection Weighting. This is explored further in Experiment 20 (section 9.5).

9.4 Mixed Weighting Experiments

Experiment 19. *What are the effects are using Edge Weighting and Vertex Weighting together? If we apply the best cases of both do we get a result better than each individually?*

We solved the same problem repeatedly using Convection Weighting (on the

graph edges) combined with Flow Weighting (on the graph vertices). We tested a combination of different convection parameters s_c combined with different flow parameters s_f which is like performing Experiments 3 and 17 simultaneously.

We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [10^5\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0,1] \times [0,1] \in \mathbb{R}^2 \tag{9.44}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.45}$$

Flow function $z(x,y) = x + 10^{-3}$ and 64 processors refined to 2.0×10^5 unknowns each were used. The results are displayed in Tables 9.19 and 9.20. The first column of data is the convergence rates of $||\delta u_k||$ using only convection weighting. These are the same rates that we saw in Experiment 3 (section 9.2.3) when $\beta = 10^5$ in Table 9.2. The first row of data is the convergence rates using only flow weighting. These are the same rates that we saw in Experiment 17 (section 9.3.4) when $\beta = 10^5$ in Table 9.15. The values across the top row are the flow weighting parameters and the values down the first column are the convection weighting parameters.

Table 9.19: DD convergence rates of $||\delta u_k||$ for combinations of convection weighting and flow weighting.

s_c	$s_f = 0.0$	0.5	0.75	1.0	1.5	2.0
0.0	0.34	0.31	0.28	0.20	0.09	0.09
1.0	0.28	0.25	0.22	0.19	0.12	0.13
1.56	0.23	0.21	0.16	0.16	0.13	0.11
1.83	0.22	0.21	0.17	0.15	0.15	0.12
2.0	0.25	0.21	0.18	0.17	0.15	0.13
2.23	0.25	0.21	0.18	0.15	0.18	0.15

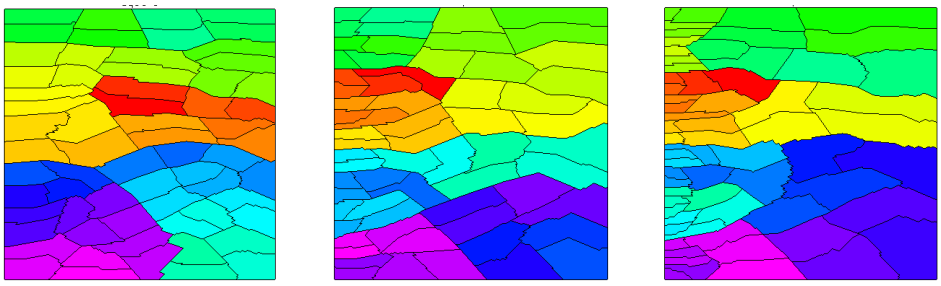
In Table 9.19, the convergence rate for the unweighted partition is the top left data entry 0.34. For each convergence rate in the table, $\log(0.34)/\log(r_k)$ equals the number of DD iterations reduction factor. These factors are displayed in Table 9.20.

Table 9.20: DD iteration reduction factors for combinations of convection weighting and flow weighting.

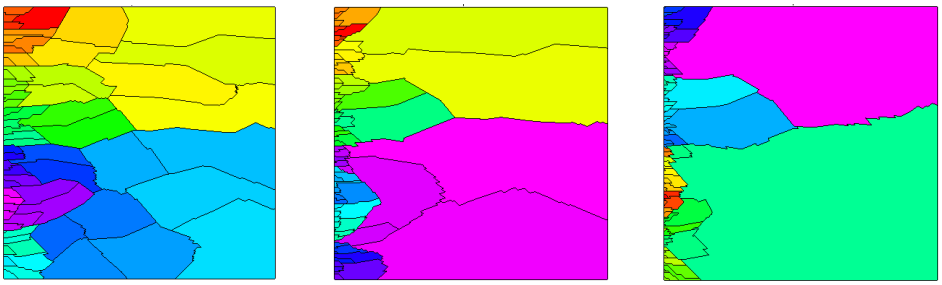
s_c	$s_f = 0.0$	0.5	0.75	1.0	1.5	2.0
0.0	1.0	0.92	0.85	0.67	0.45	0.45
1.0	0.85	0.78	0.71	0.65	0.51	0.53
1.56	0.73	0.69	0.59	0.59	0.53	0.49
1.83	0.71	0.69	0.61	0.57	0.57	0.51
2.0	0.78	0.69	0.63	0.61	0.57	0.53
2.23	0.78	0.69	0.63	0.57	0.63	0.57

How these two weighting schemes combine is interesting. If you restrict flow weighting to $s_f \leq 1$ and allow convection weighting all possibilities $0 \leq s_c \leq 4.0$, then using both methods together is better than using either individually. However, when flow weighting is applied with parameter $s_f > 1.0$, it is better by itself and adding convection weighting degrades its performance. The performance of convection weighting is always improved by adding flow weighting.

When you apply Error Weighting to $-\triangle u + [\beta\ 0]^T \cdot \nabla u - 1 = 0$, it creates a partition similar to Flow Weighting with $s_f = 2$ because of the boundary layer. Therefore if you are using Error Weighting on this problem, then adding Convection Weighting degrades the convergence.



(a) $s_f = 0.0$ (b) $s_f = 0.5$ (c) $s_f = 0.75$



(d) $s_f = 1.0$ (e) $s_f = 1.5$ (f) $s_f = 2.0$

Figure 9.45: Each partition has Convection parameter $s_c = 2.0$ and 64 parts. Flow parameter s_f varies.

9.5 DD Convergence Dependence

Experiment 20. *When solving convection dominated PDEs or anisotropic diffusion PDEs with an unweighted scheme, does the Bank-Holst paradigm DD solver convergence rate of $||\delta u_k||$ depend on the number of processors and/or domain aspect ratio?*

Using an unweighted scheme, we solved the same convection dominated problem repeatedly with different numbers of processors and different domain aspect ratios while keeping $||b||h = 34.2$ constant. For the base cases of 64 processors in Table 9.21 and domain aspect ratio 1:1 in Table 9.22, this is accomplished by letting $\beta = 10^5$. Then as you double the number of processors, you increase β by a factor of $\sqrt{2}$ and when you double the domain aspect ratio, you decrease β by a factor of $\sqrt{2}$. We found $u \in H^2(\Omega)$ such that

$$-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0 \text{ on } \Omega = [0, \kappa] \times [0, 1] \in \mathbb{R}^2 \tag{9.46}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.47}$$

The results are shown in Tables 9.21 and 9.22 and Figure 9.46.

Table 9.21: DD convergence rate of $||\delta u_k||$ based on the number of processors and $\kappa = 1$ for convection dominated PDE.

processors	9	16	25	36	49	64	128	256	512	900
hops $= \sqrt{\text{proc}}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	0.07	0.15	0.22	0.28	0.31	0.34	0.52	0.64	0.70	0.71

Table 9.22: DD convergence rate of $||\delta u_k||$ based on κ and the number of processors equal 64 for convection dominated PDE.

κ	7.11^{-1}	4^{-1}	2.56^{-1}	1.78^{-1}	1.31^{-1}	1	2	4	8	14
hops $= 8\sqrt{\kappa}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	0.13	0.18	0.19	0.23	0.29	0.34	0.52	0.61	0.63	0.65

Next, using an unweighted scheme, we solved the same anisotropic diffusion problem repeatedly with different numbers of processors and different domain aspect ratios. We found $u \in H^2(\Omega)$ such that

$$-\nabla \cdot \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \nabla u - 1 = 0 \text{ on } \Omega = [0, \kappa] \times [0, 1] \in \mathbb{R}^2 \tag{9.48}$$

$$u = 0 \text{ on } \partial\Omega \tag{9.49}$$

The results are shown in Tables 9.23 and 9.24 and Figure 9.46.

Table 9.23: DD convergence rate of $||\delta u_k||$ based on the number of processors and $\kappa = 1$ for anisotopic diffusion PDE.

processors	9	16	25	36	49	64	128	256	512	900
hops $= \sqrt{\text{proc}}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	-	0.41	0.43	0.43	0.47	0.48	0.52	0.54	0.56	0.57

Table 9.24: DD convergence rate of $||\delta u_k||$ based on κ and the number of processors equal 64 for anisotopic diffusion PDE.

κ	7.11^{-1}	4^{-1}	2.56^{-1}	1.78^{-1}	1.31^{-1}	1	2	4	8	14
hops $= 8\sqrt{\kappa}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	-	0.38	0.41	0.45	0.45	0.48	0.51	0.51	0.50	0.44

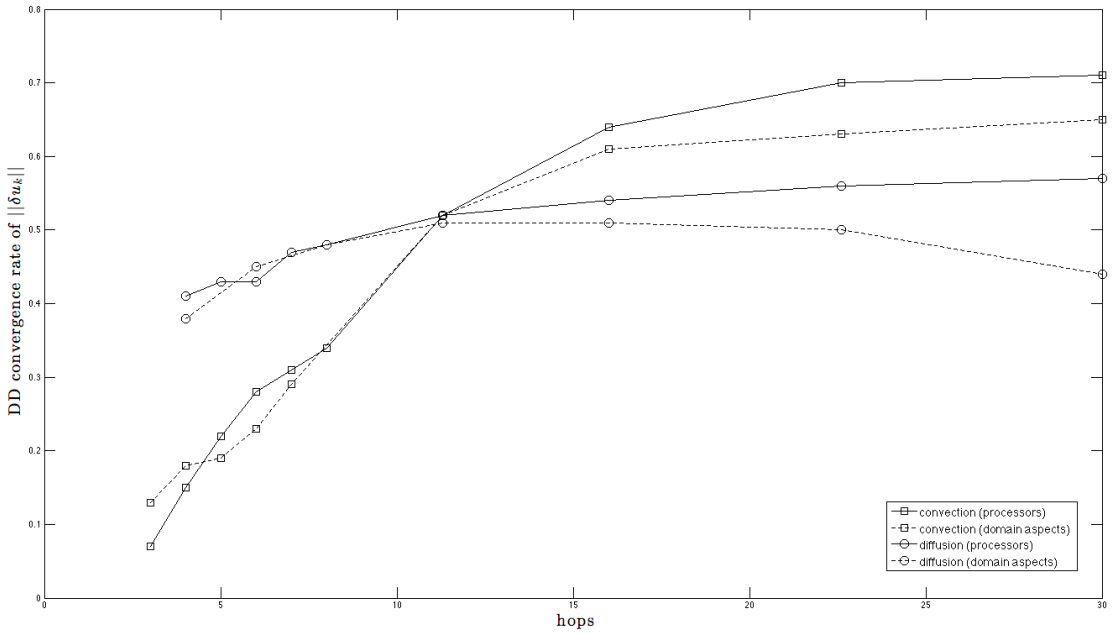


Figure 9.46: DD convergence rate of $||\delta u_k||$ based on number of processors or domain aspect ratio.

Regarding convection dominated PDEs, it appears that the DD convergence rate of $||\delta u_k||$ has a strong dependence on the number of subdomains that must be crossed when traversing the entire domain in the direction of convection (hereafter referred to as hops). Regarding anisotropic diffusion PDEs, there appears to be only a slight dependence on hops.

Each processor in the Bank-Holst paradigm maintains a fine mesh in its own subdomain and a coarse mesh of the entire domain. Information from all of these degrees of freedom is utilized in the Bank-Holst paradigm DD solver. This has been shown to reduce this method's dependence on the number of processors [15]. In all of the experiments in this dissertation including the previous ones in this subsection, each processor has had approximately $\frac{1}{4}$ of its degrees of freedom outside its own subdomain and $\frac{3}{4}$ of its degrees of freedom inside its subdomain. (An original coarse mesh of 2.0×10^4 unknowns gets partitioned and then each processor refines to 2.0×10^5.)

To test if the dependence that we have just observed is a result of each processor not placing enough of its degrees of freedom outside its own subdomain,

we repeated the experiments and had each processor place approximately $\frac{3}{5}$ of its degrees of freedom outside of its subdomain and $\frac{2}{5}$ inside. (An original coarse mesh of 1.0×10^5 unknowns gets partitioned and then each processor refines to 2.0×10^5.) The results are displayed in Tables 9.25 and 9.26 and Figure 9.47.

Table 9.25: DD convergence rate of $||\delta u_k||$ based on the number of processors and $\kappa = 1$ for convection dominated PDE and 100k/200k refinement.

processors	9	16	25	36	49	64	128	256	512	900
hops $= \sqrt{\text{proc}}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	0.05	0.09	0.11	0.25	0.21	0.23	0.45	0.59	0.65	0.65

Table 9.26: DD convergence rate of $||\delta u_k||$ based on the number of processors for anisotropic diffusion PDE and 100k/200k refinement.

processors	9	16	25	36	49	64	128	256	512	900
hops $= \sqrt{\text{proc}}$	3	4	5	6	7	8	11.3	16	22.6	30
convergence rate	0.28	0.31	0.34	0.39	0.41	0.41	0.45	0.49	0.50	0.50

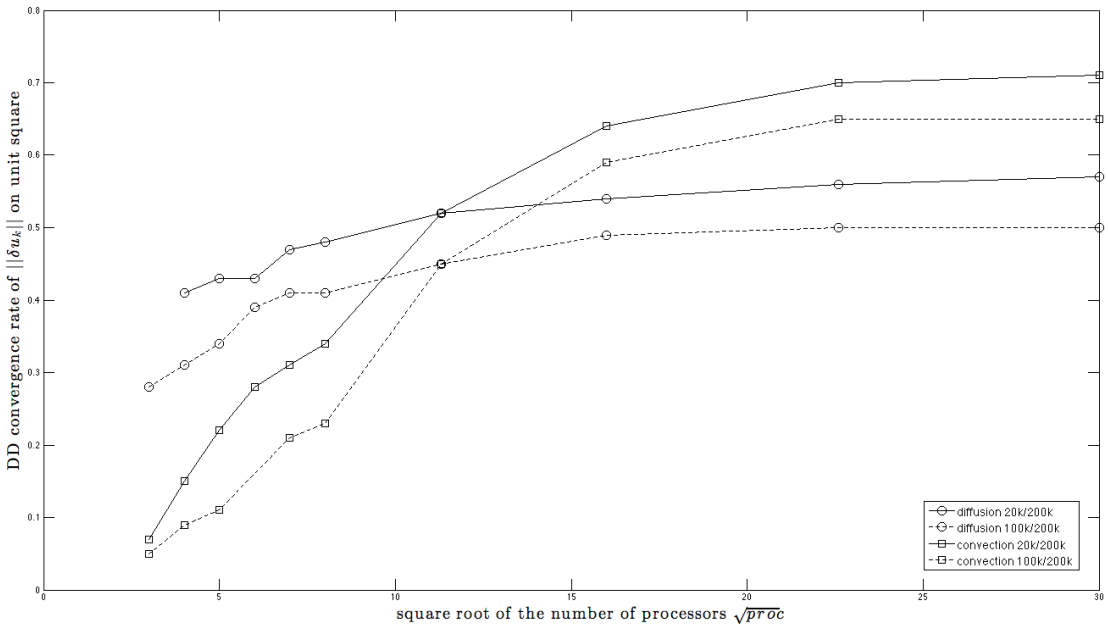


Figure 9.47: DD convergence rate of $||\delta u_k||$ based on the number of processors. The solid lines represent 20k/200k refinement and the dashed lines represent 100k/200k refinement.

This new experiment suggests that the dependencies we are seeing are a feature of the Bank-Holst paradigm DD solver and are not a result of each processor failing to put enough degrees of freedom outside its own subdomain.

The experiments in this subsection help us understand the dependancies we saw in Experiment 7 (section 9.2.7), Experiment 11 (section 9.2.11), Experiment 13 (section 9.2.13), and Experiment 18 (section 9.3.5). With convection dominated PDEs, the DD convergence rate of $||\delta u_k||$ appears to be roughly proportional to the log of the number of hops ($convergence \approx 0.3051 \ln(hops) - 0.2659$). This suggests that you need to reduce the number of hops in a logarithmic fashion in order to achieve a significant improvement in the DD convergence rate of $||\delta u_k||$. When holding the flow weighting parameter constant and varying the starting number of hops, Error Weighting and Flow Weighting do this but when holding the convection weighting parameter constant and varying the starting number of hops, Convection Weighting and Stiffness Matrix Weighting only reduce the number of hops in a linear fashion.

With anisotropic diffusion PDEs, the DD convergence rate of $||\delta u_k||$ appears to be roughly linearly proportional to the number of hops ($convergence \approx 0.0061(hops) - 0.4155$). Therefore both the Flow Weighting and Convection Weighting schemes achieve significant improvement in the DD convergence rate of $||\delta u_k||$ when their respective parameters are held constant and the starting number of hops is varied.

9.6 DD Communication Time

Experiment 21. *What is the ratio of communication time to computation time during one iteration of the Bank-Holst paradigm DD solver?*

Lemmas 5.0.11 and 5.0.12 discuss the relevance of the ratio of computation time to communication time. To summarize, it is advantageous to use a certain partitioning scheme even if it increases communication time per DD iteration as long as it decreases the total number of iterations sufficiently. The reduction necessary depends on α, the ratio of computation time to communication time.

In order to measure α, we added code to PLTMG that measures communication time and computation time. Then we solved a variety of different problems using PLTMG's normal partitioning scheme when our computer cluster BOOM was free from other jobs. Table 9.27 shows the results of solving $-\triangle u - 10^5 u_x - 1 = 0$ on the unit square with the dirichlet boundary condition $u = 0$. The times in Table 9.27 are in seconds and refer to one iteration of the Bank-Holst paradigm DD solver.

Table 9.27: Ratio of computation to communication time for convection-diffusion

number of processors	unknowns per processor	communication time	computation time	α
64	2.0×10^5	0.051	32.7	641
64	3.0×10^5	0.072	56.7	788
128	2.0×10^5	0.132	36.5	277
128	3.0×10^5	0.186	69.2	372
256	2.0×10^5	0.318	44.1	139
256	3.0×10^5	0.420	89.9	214

Table 9.28 shows the results of solving $-10u_{xx} - u_{yy} - 1 = 0$ on the unit square with the dirichlet boundary condition $u = 0$. The times in Table 9.28 are in seconds and refer to one iteration of the Bank-Holst paradigm DD solver.

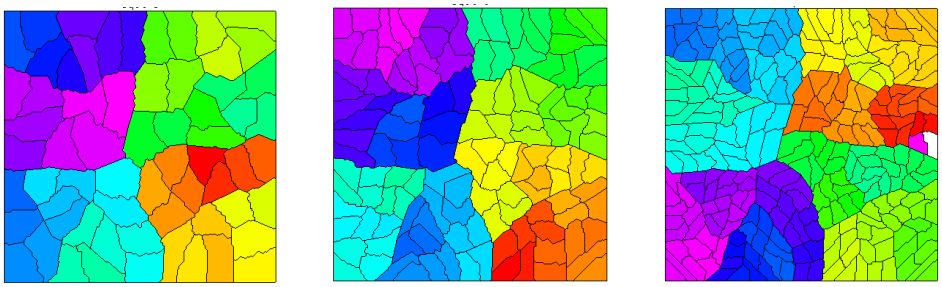
Table 9.28: Ratio of computation to communication to time for diffusion

number of processors	unknowns per processor	communication time	computation time	α
64	2.0×10^5	0.051	32.0	627
64	3.0×10^5	0.072	61.6	856
128	2.0×10^5	0.132	30.4	230
128	3.0×10^5	0.170	67.7	398
256	2.0×10^5	0.327	35.2	108
256	3.0×10^5	0.420	69.4	165

By Lemma 5.0.12, these large $\alpha > 100$ values together with the fact that our weighting schemes achieve DD iteration reduction factors $0.25 < r < 0.75$ imply that we could increase the interface length up to a factor of 34 which by Corollary 5.0.14 means that we could use rectangles with aspect ratios up to 4622! Our

preference of using rectangles with aspect ratios of 4:1 or 8:1 increase the interface length by a factor of only 1.2 or 1.6 respectively and surely are safe to use.

More important than the ratio of computation time to communication time is the ratio of the iteration time portion which is independent of interface length versus the iteration time portion which grows with interface length. Section 5.4 explains this. By analyzing all of our experiments, we found that communication time is indeed proportional to interface length. Each iteration of Bank-Holst paradigm DD solver makes 3 or more calls to MPI_ALLGATHERV to exchange boundary information. Communication time is the sum of all these calls and the associated overhead such as some interpolation procedures. We found that if you double the interface size, you double the communication time.



(a) 64 parts (b) 128 parts (c) 256 parts

Figure 9.48: Partitions of the unit square into 64, 128, and 256 processors.

Next we studied whether there was a correlation with slope $\neq 1$ between computation time and interface size. During each DD iteration, the Bank-Holst paradigm DD solver solves a system of equations as described in Algorithm 7 (section 3.3). Modifying the interface and varying the partition affects the matrix A^*. This in turn affects the time needed to compute the incomplete LU preconditioner for the Conjugate Gradient method which is used to solve $A^*\delta U = R^*$. Computing ILU is the majority of the computation time.

Table 9.29 lists the times in seconds to complete the computation of one iteration of Bank-Holst paradigm DD solver for experiments 1-12. The time it took the weighted and unweighted partition to solve the same problem are paired together. The pairs in the range $[20, 60] \times [20, 60]$ are plotted in Figure 9.49. The

best fit line through all 26 data points is $y = 1.03x - 2.34$. Since the slope is essentially equal to 1, we conclude that the computation time is independent of whether we use our weighted scheme or not.

Table 9.29: Time to complete the computation of one iteration of DD in seconds listed by experiment number.

	1	2a	2b	2c	2d	3a	3b	3c	4	5
unweighted	33.3	38.0	38.5	42.0	29.9	32.0	38.6	28.3	38.6	49.2
weighted	39.2	36.3	42.3	45.0	31.5	36.1	34.2	26.8	34.2	43.4
	6a	6b	6c	7a	7b	7c	7d	7e	7f	10a
unweighted	38.9	35.4	50.4	41.7	42.0	63.1	186	127	334	39.2
weighted	39.0	33.6	48.3	42.0	54.0	64.7	151	118	364	40.1
	10b	10c	11a	11b	12a	12b	-	-	-	-
unweighted	35.4	28.6	76.5	404	48.3	47.2	-	-	-	-
weighted	38.5	27.4	60.9	413	45.3	44.2	-	-	-	-

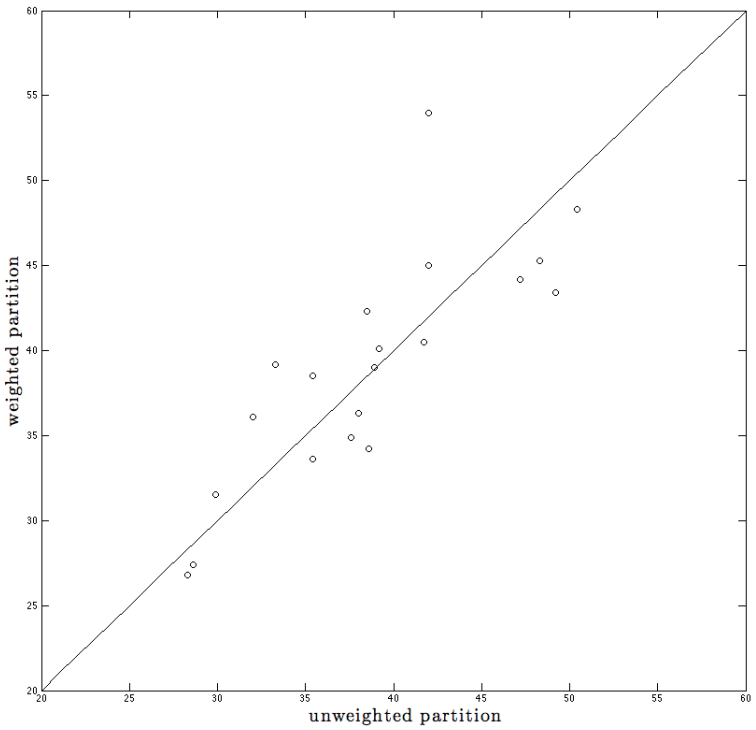


Figure 9.49: Time in seconds for an unweighted and weighted partition to complete the computation of one iteration of DD.

9.7 Summary and Conclusions

In conclusion, the theory of Chapter 8 and the numerical experiments of Chapter 9 demonstrate that domain decomposition methods converge faster if whenever the elliptic PDE being solved has strong convection or anisotropic diffusion, then this information is incorporated into the partitioning process. Regardless of boundary conditions, domain shape, force functions, or reaction terms, these directional PDEs experience a reduction factor in the number of DD iterations required between 0.25 and 0.75. Also, we solved problems with up to 900 processors and with up to 8.0×10^5 unknowns per processor and these problems experienced the same reduction factors between 0.25 and 0.75.

A specific and notable example was when 512 processors solved for $\frac{1}{4}$ billion unknowns. The PDE $-\triangle u - [10^{4.5}\ 0]^T \cdot \nabla u - 1 = 0$ was solved on the unit square with $u = 0$ Dirichlet boundary conditions. Without using any weighting schemes to help partition the domain, it took our 1 teraflop cluster 2.8 hours to find the solution with a $||\delta u_k||$ DD convergence rate of 0.68. When we used our stiffness matrix weighting scheme, it took only 1.3 hours! It had a $||\delta u_k||$ DD convergence rate of 0.40.

Five novel graph partitioning weighting schemes were presented in this dissertation; Convection weighting, Gradient weighting, Stiffness Matrix weighting, Error weighting, and Flow weighting. The first three use graph edge weighting to create rectangle shaped subdomains parallel to convection or anisotropic diffusion. The last two use vertex weighting to create increasingly sized subdomains that go from small to large when you move in the direction of convection or diffusion.

9.7.1 Edge Weighting schemes

With the correct choice of parameters, the first three schemes create essentially the same partition and therefore perform the same. One major difference is that Convection and Gradient weighting require the user to manually modify parameters to control the rectangle aspect ratios while the Stiffness Matrix weighting does this automatically. Another major difference is that Gradient weighting

needs the solution's gradient to be parallel to convection or anisotropic diffusion to work well. Note that Convection weighting can be used on anisotropic diffusion problems by using the prominent diffusion direction in the weighting formula instead of the usual convection direction vector.

When using Convection or Gradient weighting, the convection parameter s needs to be set to zero when both $||b||h/||a|| < 1$ and $\lambda_1(a)/\lambda_2(a) < 2$. When $||b||h/||a|| \geq 1$ and $\sqrt{rP} \leq 20$ where P is the number of processors and r is the domain aspect ratio in the direction of convection, then s must be set to create subdomain rectangles with aspect ratios of at least 4:1 (with METIS, use $s \geq 2$). When $||b||h/||a|| \geq 1$ and $\sqrt{rP} > 20$, then s must be set to create rectangles with aspect ratios of at least 8:1 (with METIS, use $s \geq 3$). Our experiments suggest that when $\sqrt{rP} > 30$, the rectangle aspect ratio needs to be increased again. Data suggests that the rectangle subdomain aspect ratio needed for optimal convergence is proportional to $\sqrt{rP}$, the square root of the product between the number of processors and the domain aspect ratio in the direction of convection. (See section 9.5 for more details.)

For anisotropic diffusion problems, when $4 \geq \lambda_1(a)/\lambda_2(a) \geq 1$, then s must be set to create rectangles with aspect ratio equal $\lambda_1(a)/\lambda_2(a) : 1$ and when $\lambda_1(a)/\lambda_2(a) > 4$, then s must be set to create rectangles with aspect ratio of 4:1.

The Stiffness Matrix Weighting scheme automatically adjusts itself in accordance with the description in the previous two paragraphs with one exception. It automatically does 4 of the 5 adjustments listed, but when solving convection dominated problems with $\sqrt{rP} \geq 20$, it doesn't automatically increase the rectangle aspect ratio. In these situations, Stiffness Matrix Weighting needs $\alpha = 100$ and $\beta = -15$ to create rectangles with aspect ratio 8:1.

The Edge Weighting schemes do not affect the accuracy of the final finite element solution. The final global mesh is the same whether you use an edge weighting scheme or not.

Overall, the Edge Weighting schemes enable DD to finish faster because they have a faster convergence rate than unweighted schemes and one iteration of

either weighted or unweighted takes the same time as shown in Section 9.6.

9.7.2 Vertex Weighting schemes

The vertex weighting schemes are powerful in that they can both improve DD convergence and affect the accuracy of the final finite element solution. Using Error Weighting will never degrade the accuracy and when it's possible to improve accuracy, it will accomplish that. Flow Weighting can both improve and degrade the final solution's accuracy. Therefore the user needs to be especially careful when using it.

For the most part, Flow Weighting is a theoretical idea to help us study the behavior of Error Weighting and shouldn't be employed. If a PDE has a finite element solution that has uniform error on a uniform mesh then Error Weighting performs exactly like unweighting and accomplishes nothing. But in this situation, Flow Weighting could be used to improve the convergence of the DD method if the PDE has strong convection or anisotropic diffusion. However, using Flow Weighting will also reduce the accuracy of the final solution as compared to using an unweighted scheme. Furthermore, one could use an unweighted scheme with fewer degrees of freedom to compute both a more accurate finite element solution and in less time than Flow Weighting. Therefore when Error Weighting cannot be used, Flow Weighting isn't desirable either.

When both vertex weighting and edge weighting can be applied, our data shows that vertex weighting achieves the fastest convergence. And, vertex weighting doesn't need to be adjusted as $\sqrt{rP}$ increases.

9.7.3 Application

From Experiment 19 (section 9.4), we see that edge weighting and vertex weighting don't combine well when they are both calibrated to perform their best. So, which scheme should be used when? The simple answer is; use Error Weighting when it works well and use Stiffness Matrix Weighting otherwise. And don't try to combine them.

When Error Weighting performs well, it outperforms the edge weighting schemes because it both achieves faster convergence and simultaneously improves the accuracy of the final solution. Error Weighting is successful when strong convection or anisotropic diffusion is present and the finite element solution has error $||\nabla(u-u_h)||_{L^2(t_k)}$ that gets larger as you move in the convection direction or prominent diffusion direction. From our experiments, we observed that this occurs in PDEs with strong convection whose solutions have a boundary layer. We did not observe this in convection problems without a boundary layer and we did not observe this in anisotropic diffusion problems.

For anisotropic diffusion problems and convection problems without a boundary layer, we observed that Stiffness Matrix weighting worked best.

Alternatively, one can use Flow Weighting in place of Error Weighting (in situations when Error Weighting would work well) to gain more control over the partition. And one could use Convection or Gradient Weighting in place of Stiffness Matrix Weighting to gain more control over the partition. Flow, Convection, and Gradient Weighting have parameters the user can change, however, our data shows that Error Weighting and Stiffness Matrix Weighting already perform like Flow Weighting and Convection Weighting with optimally set parameters.

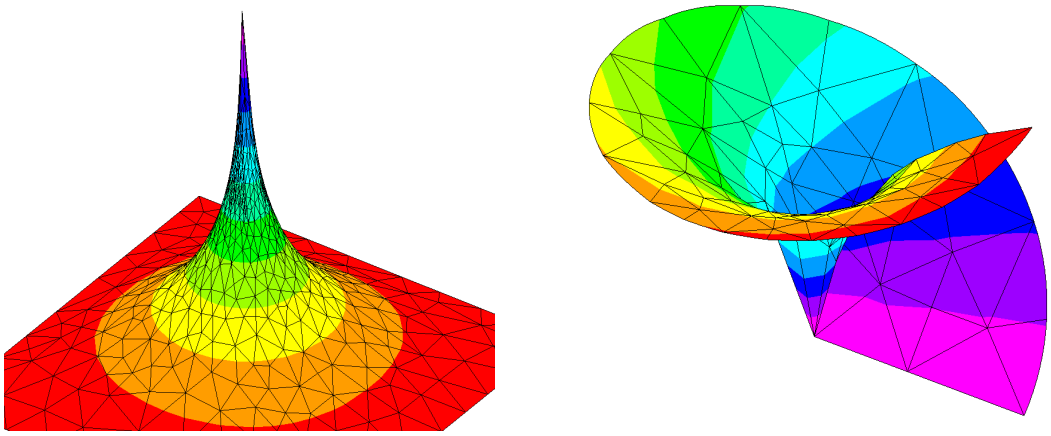
Chapter 10

Future Research

This section presents ideas for further research.

10.1 Singularities

When there are singularities involved in (1.1) - (1.3), how can the domain be partitioned differently to speed up convergence and/or improve the accuracy of the final solution? Singularities can be present in the stiffness matrix when singularities exist in coefficients a, b, and c or singularities can be present in the load vector when singularities are present in f, or singularities can be present in the boundary conditions affecting both the load vector and stiffness matrix. All three of these can create singularities in the final solution u or just make solving for u difficult.



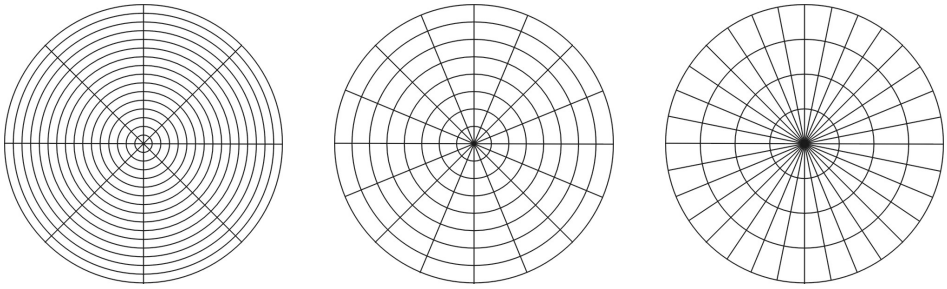
(a) Load vector singularity (b) Boundary singularity

Figure 10.1: Singularity examples

Figures 10.1 shows two examples of singularities present in (1.1) - (1.3). The PDE causing Figure 10.1a has a load vector which forces the function $(x^2 + y^2)^{-\frac{1}{2}}$ to show up in the solution creating a singularity at the origin. Figure 10.1b's underlying PDE has a domain of the unit circle with a crack on the x axis. There are dirichet boundary conditions on one side of the crack and neumann boundary conditions on the other. This creates a singularity at the origin.

During the research for this thesis, many experiments were conducted on singular problems with mixed results. One positive result was that numerical experiments indicated that the Error Weighting Scheme helps these problems to converge versus not converging. That scheme places many small parts near the singularity which helps resolve the solution there.

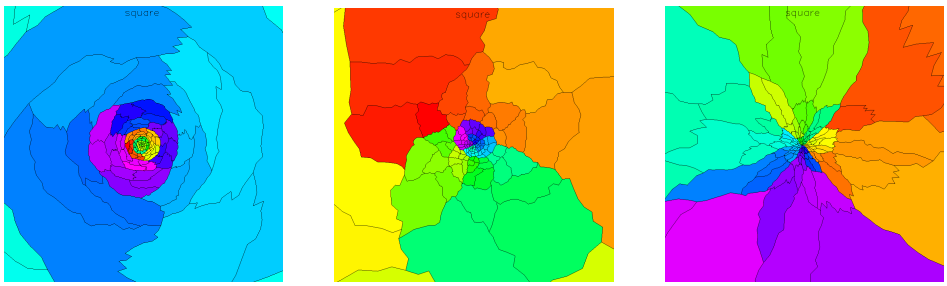
In the future, we would like to continue studying how weighting schemes can improve convergence and/or accuracy in the different types of singular problems mentioned above.



(a) Error and Contour (b) Error Weighting (c) Error and Gradient

Figure 10.2: Different partitions using different weighting schemes to address a singularity.

Gradient Weighting Schemes produced the three different partitions in Figure 10.2 and 10.3. Each partition has 128 parts and each domain has a singularity in the middle. Figure 10.2a and 10.3a encourage contour (perpendicular to gradient) cutting in addition to error weighting. Figure 10.2c and 10.3c encourage gradient cutting in addition to error weighting. Some numerical experiments were done to see which partition is better, but the results were inconclusive. Further research is needed to determine the effects.



(a) Error and Contour (b) Error Weighting (c) Error and Gradient

Figure 10.3: Different partitions using different weighting schemes to address a singularity with METIS.

A sixth type of weighting scheme that was not presented in Chapters 5 and 7 is Solution Weighting. If you increase the edge weights of the representative graph where the solution u approaches infinity, the partitioner will avoid cutting there. This will place the entire singularity in one subdomain as shown in Figure 10.4. Is it better for one processor to deal with the singularity and then communicate the

information to other processors during the DD Method, or is it better for multiple processors to each have a portion of the singularity in their subdomain? This is a question for future research.

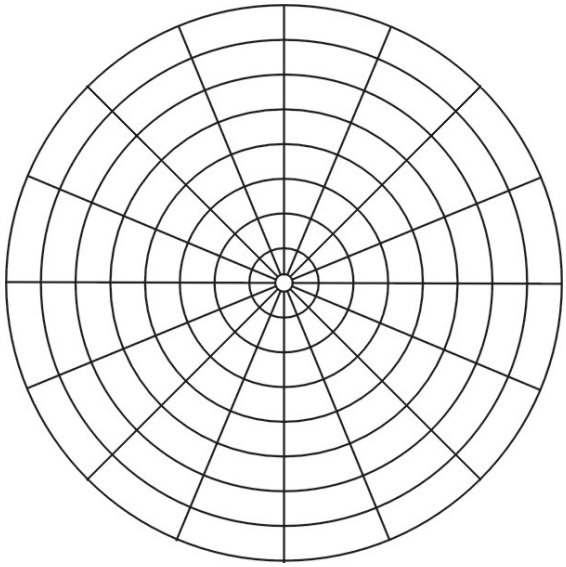


Figure 10.4: Singularity isolated to one processor

10.2 Adaptive Refinement

Most of the experiments and discussion in this thesis were done using uniform refinement. When individual processors refined their parts uniformly to different sized elements than other processors, we saw global meshes that were not overall uniform, but they were uniform in each partition part.

Using adaptive refinement before partitioning was discussed briefly in Section 7.4 and a few other features of adaptive refinement were discussed elsewhere. More work can be done on the effects of adaptive refinement in conjunction with the weighting schemes presented in this thesis. After using each of the different weighting schemes to partition a mesh, what is the difference between the final global meshes if each partition refines their part adaptively instead of uniformly during the DD Method iteration?

We investigated this during the research of this thesis but were met with inconclusive results in most cases. To be more specific, start with a uniform mesh

of the unit square and consider partitioning it two different ways. Partition into squares like Figure 5.2 and rectangles like Figure 5.3a. Rectangles are created by using an edge weighting scheme from Chapter 5. Call these partitions 1 and 2 respectively.

After partitioning, if you refine each part of each partition uniformly and then combine the parts, you will have two global meshes. Both global meshes will be identical. However, if after partitioning, you refine each part of each partition adaptively and then combine the parts, you will have two different global meshes. This will affect the accuracy of the final global finite element solution as discussed in Chapter 6. Which solution is more accurate? When we investigated this question, we had trouble removing the variance to make a conclusive determination.

Five weighting schemes were presented in this thesis with a sixth mentioned in the preceding section. Future research can study the effects of these schemes and combinations of these schemes on final solution accuracy when mixed with adaptive refinement.

Also when singularity problems are studied, adaptive refinement study should be mixed into that too. It may be the case, that there are no weighting schemes which will improve the convergence rate of singular problems. The only benefit may be that certain partitions improve the accuracy of the final finite element solution when used in conjunction with adaptive refinement.

10.3 Preconditioned Conjugate Gradient

Preconditioned Conjugate Gradient is a type of Krylov Subspace Method. In Section 8.1, preconditioned Richardson Iteration was introduced for solving $Ax = f$. Algorithm 9 showed that finding $x^{(k)}$ only used information about $x^{(k-1)}$ and none of the other previous iterates $\{x^{(k-2)}, ..., x^{(0)}\}$. Techniques that use information about more than one previous iterate are called *accelerator* techniques. Examples of accelerator techniques are the Chebyshev method, the preconditioned conjugate gradient method (PCG), GMRES, and BiCG-stab [43].

Algorithm 12 Preconditioned Conjugate Gradient

1: Initialize x_0 to an initial guess.
2: Calculate $r_0 = f - Ax_0$, $z_0 = Br_0$, $p_0 = z_0$
3: **for** $k = 0, 1, 2, ...$ until convergence **do**
4: $\quad \alpha_k = (r_k^T z_k)/(p_k^T A p_k)$
5: $\quad x_{k+1} = x_k + \alpha_k p_k$
6: $\quad r_{k+1} = r_k - \alpha_k A p_k$
7: $\quad z_{k+1} = B r_{k+1}$
8: $\quad p_{k+1} = z_{k+1} + \left((z_{k+1}^T r_{k+1})/(z_k^T r_k)\right) p_k$
9: **end for**

In Section 8.4, we saw that the Stiffness Matrix edge weighting scheme presented in this thesis made a good preconditioner for solving $Ax = f$ with Richardson's Method. The Stiffness Matrix Weighting Scheme creates a preconditioner for A using A. Therefore, this weighting scheme can be used for solving $Ax = f$ even without an associated PDE.

For future study, we would like to create and test the properties of preconditioners for Krylov Subspace Methods made from incorporating the Stiffness Matrix Weighting Scheme.

The Stiffness Matrix Weighting Scheme will partition the unknowns in the vector x from $Ax = f$ into many small groups of dependent degrees of freedom. Next you could use these groups to create a block Jacobi preconditioner or block Gauss Seidel preconditioner. Or you could create multilevel preconditioners by only dividing x into a small number of groups and then creating a block preconditioner by using standard preconditioners on the blocks of A corresponding to the groups of x.

10.4 More Domain Decomposition

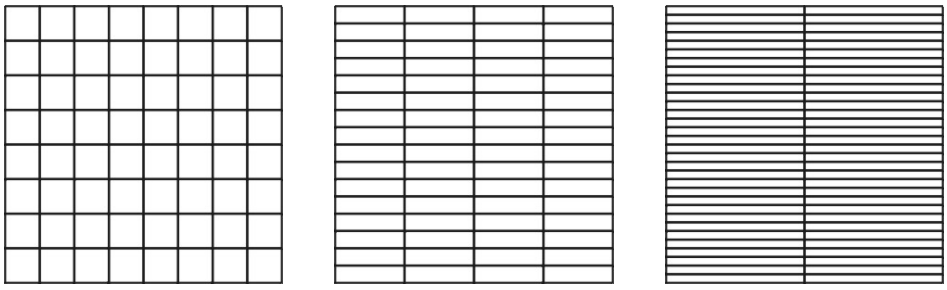
In this work we analyzed the effect of new partitioning techniques using the Schwarz Domain Decomposition Methods and the Bank-Holst paradigm DD solver. One would suspect that the same results hold true with other Domain

Decomposition Methods. In the future, we would like to test these techniques on other Domain Decomposition Methods. In addition to confirming that the techniques presented here benefit more DD schemes, what we find may also help us understand and improve our methods.

10.5 Mathematical Model

Experiment 20 (section 9.5) is the beginning of a model that could explain how the weighting schemes presented in this research achieve faster DD convergence on convection dominated PDEs. Successfully discovering a model could help us improve the schemes presented here and/or formulate new techniques.

Assume we have a PDE with strong convection in the x direction. Experiment 20 discusses the relationship between x hops and the DD convergence rate of $||\delta u_k||$. The next step in developing a model of DD convergence is to analyze the y direction. When using Convection Weighting, as we increase the elongation of the rectangle subdomains, you reduce the number of x direction hops and improve convergence. But at some point, reducing the hops further does not improve convergence. Most likely this is because as you elongate the rectangles, you increase the number of y hops which slows down convergence of the diffusion in the y direction.



(a) aspect 1:1 (b) aspect 4:1 (c) aspect 16:1

Figure 10.5: When increasing the aspect ratio of subdomain rectangles, x hops decrease and y hops increase.

Even if the original convection dominated PDE doesn't have much diffusion

in the y direction, when you solve $-\triangle u - [\beta\ 0]^T \cdot \nabla u - 1 = 0$ with Finite Elements, it requires the application of upwinding for stability. Scharfetter Gummel Upwinding adds artificial diffusion in both the x and y directions when using a triangle mesh. And, the magnitude of the added diffusion in the y direction is roughly one half the magnitude of the added diffusion in the x direction. So, the diffusion in the y direction becomes significant.

The fact that using x direction rectangle subdomains of any aspect ratio slows down the convergence of $-\triangle u - 1 = 0$ versus using square subdomains supports the idea that increasing y hops degrades the DD convergence rate. (Experiment 3 section 9.2.3 showed this.)

Another interesting observation that needs to be incorporated into the model is that Flow Weighting with flow function $z(x, y) = x + 10^{-3}$, $s = 2$, and $\Omega = [0, 1] \times [0, 1]$ does not slow down the convergence of $-\triangle u - 1 = 0$. (Experiement 17 section 9.3.4 showed this.) So although both Flow Weighting and Convection Weighting facilitate information travel in the x direction, only Flow Weighting simultaneously preserves the y direction convergence.

A complete model may also explain why Edge Weighting and Vertex Weighting did not work well together (as seen in Experiment 19 section 9.4) and perhaps a model could propose a way to combine Edge Weighting and Vertex Weighting successfully.

Bibliography

[1] A.K. Aziz and I. Babuska. *The mathematical foundations of the finite element method with applications to partial differential equations.* Academic Press, New York, 1972.

[2] R. E. Bank. A domain decomposition solver for a parallel adaptive meshing paradigm. *Domain Decomposition Methods in Science and Engineering*, XVI:3–14, 2006.

[3] R. E. Bank, J. F. Burgler, W. Fichtner, and R. K. Smith. Some upwinding techniques for finite element approximations of convection-diffusion equations. *Numerical Math*, 58:158–202, 1990.

[4] R. E. Bank, W. M. Coughran, and L. C. Cowsar. Analysis of the finite volume scharfetter-gummel method for steady convection diffusion equations. *Computing and Visualization in Science*, 1:123–136, 1998.

[5] R. E. Bank and M. J. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM J. on Scientific Computing*, 22:1411–1443, 2000.

[6] R. E. Bank and M. J. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Review*, 45:292–323, 2003.

[7] R. E. Bank and P. K. Jimack. A new parallel domain decomposition method for the adaptive finite element solution of elliptic partial differential equations. *Concurrency and Computation: Practice and Experience*, 13:327–350, 2001.

[8] R. E. Bank, P.K. Jimack, S. A. Nadeem, and S. V. Nepomnyaschikh. A weakly overlapping domain decomposition preconditioner for the finite element solution of elliptic partial differential equations. *SIAM J. on Scientific Computing*, 23:1817–1841, 2002.

[9] R. E. Bank and J. Lu. Asymptotically exact a posteriori error estimators, part i: Grids with superconvergence. *SIAM J. Numerical Analysis*, 41:2294–2312, 2003.

[10] R. E. Bank and S. Lu. A domain decomposition solver for parallel adaptive meshing paradigm. *SIAM J. on Scientific Computing*, 45:292–323, 2003.

[11] R. E. Bank and H. Nguyen. Domain decomposition and hp-adaptive finite elements. *Lecture Notes in Computational Science and Engineering*, 78:3–13, 2011.

[12] R. E. Bank and H. Nguyen. hp adaptive finite elements based on derivative recovery and superconvergence. *Computing and Visualization in Science*, submitted.

[13] R. E. Bank and J. S. Ovall. Dual functions for a parallel adaptive method. *SIAM J. on Scientific Computing*, 29:1511–1524, 2007.

[14] R. E. Bank and D. J. Rose. Marching algorithms for elliptic boundary value problems. i: The constant coefficient case. *SIAM J. Numerical Analysis*, 14(5):792–829, 1977.

[15] R. E. Bank and P. S. Vassilevski. Convergence analysis of a domain decomposition paradigm. *Computing and Visualization in Science*, 11:333–350, 2008.

[16] R. E. Bank and J. Xu. Asymptotically exact a posteriori error estimators, part ii: General unstructer grids. *SIAM J. Numerical Analysis*, 41:2313–2332, 2003.

[17] R. E. Bank, J. Xu, and B. Zheng. Superconvergent derivative recovery for lagrange triangular elements of degree p on unstructured grids. *SIAM J. Numerical Analysis*, 45:2032–2046, 2007.

[18] Randolph E. Bank. *PLTMG: A software Package for Solving Elliptic Partial Differential Equations Users' Guide 11.0*. 2012.

[19] S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency and Computation: Practice and Experience*, 6:101–117, 1994.

[20] Charles-Edmond Bichot and Patrick Siarry. *Graph Partitioning*. Wiley, 2011.

[21] Dietrich Braess. *Finite Elements: Theory, fast solvers, and applications in solid mechanics*. University Press, Cambridge, second edition, 2001.

[22] A Brandt. Multilevel adaptive solutions to boundary value problems. *Math. Comp.*, 31:333–390, 1977.

[23] M Fiedler. Algebraic connectivity of graphs. *Czech. Math. J*, 23:298–305, 1973.

[24] M Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czech. Math. J*, 25:619–633, 1975.

[25] C. M. Da Fonseca and J. Petronilho. Explicit inverse of a tridiagonal k-toeplitz matrix. 2002.

[26] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The John Hopkins University Press, third edition, 1996.

[27] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to parallel computing*. Pearson Education Limited, second edition, 2007.

[28] W. HackBusch. *Multigrid methods and applications*. Springer-Verlag, 1985.

[29] B. Hendrickson and R Leland. The chaco user's guide. *Technical Report*, 1993.

[30] M. J. Holst. Sg user's guide. *http://www.fetk.org/codes/sg/index.html*, 2014.

[31] Thomas J.R. Hughes. *The Finite Element Method: Linear static and dynamic Finite Element analysis*. Dover Publications, Inc. Mineola, NY, 2000.

[32] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. *Proceedings of Supercomputing*, 1998.

[33] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of parallel and distributed computing*, 48:96–129, 1998.

[34] George Karypis. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reduced Orderings of Sparse Matrices Version 5.1.0*. 2013.

[35] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, 20(1):359–392, 1999.

[36] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(1):291–307, 1970.

[37] Stig Larsson and Vidar Thomee. *Partial Differential Equations with Numerical Methods*. Springer, 2009.

[38] S. Lu. Parallel adaptive multigrid algorithms. *PhD thesis, Dept of Math, UCSD*, 2004.

[39] Tarek P.A. Mathew. *Domain decomposition methods for the numerical solution of partial differential equations*. Springer, 2008.

[40] A. Pothen. *Graph partioning algorithms with applications to scientific computing*. Kluwer Academic Press, 1996.

[41] Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for patial differential equations*. Oxford University Press, 1999.

[42] H. D. Simon. Partitioning of unstructured problems for parallel processing. *Comp. Sys. Engrg*, 2(2/3):135–148, 1991.

[43] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 1996.

[44] A. Toselli and O. Widlund. *Domain Decomposition Methods*. Springer, 2005.