Robust Preconditioning for Indefinite and Ill-Conditioned Sparse Linear Systems

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Daniel Osei-Kuffuor

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Yousef Saad

December, 2011

Acknowledgements

I consider myself very fortunate to have benefited from the advice and support of many individuals, throughout the course of my education. It is only fitting that I give due credit to the people who have supported me over the years, and without whose help this thesis would not have been possible. This work is sponsored in part by the U.S. Department of Energy under grants DE-FG02-03ER25585 and DE-FG-08ER25841, and by the Minnesota Supercomputing Institute.

I reserve the bulk of my gratitude to my advisor, Yousef Saad. Six years ago when I was admitted into the scientific computing program at the University of Minnesota, I contacted Yousef about the possibility of working with him as a research assistant and having him as my academic advisor. I will forever be grateful for his immediate response to offer me a position in the scientific computing (scicomp) group. I have had a very rewarding graduate education experience, in which he served as an outstanding mentor and colleague. His expert advice and profound knowledge of the subject matter reinforced my interest in, and improved my understanding of the fields of numerical analysis and numerical linear algebra. The weekly group meetings and discussions exposed me to other areas of research that may complement my area of interest. Often times I left these discussions motivated and armed with new ideas, many of which contributed to the contents of this thesis. I am very grateful to Yousef for allowing me to do research with a flexible schedule, which enabled me to take classes, attend conferences and workshops, and have some vacation and family time when needed. Through his guidance and encouragement, I have had the opportunity to collaborate with many different scientists and mathematicians, which has led to various publications and the development of scientific software. I am honored to have worked with Prof. Jeff Derby, Andrew Yeckel, Xie Hui, Lingzhu Kong, Pierre Carrier, Jok Tang, Scott

MacLachlan, Masha Sosonkina, Zongzhe Li, and Ruipeng Li. Yousef encouraged me to use the summer months to pursue external opportunities that benefit my research. This led to five very productive summer internships at the Lawrence Livermore National Laboratory, where I had the opportunity to network with other students and renowned scientists, and work on various projects that complemented my research interests. I am thankful to Yousef for taking time in his busy schedule to review this thesis and make recommendations that improved its overall quality. Yousef has always been available, and ready to answer my many questions. He is always willing to exchange ideas and give advice based on his immense experience. Through his patience and guidance, I have become a better scientist, and for this I am sincerely grateful.

In the past five summers, I have had the opportunity to work as a graduate intern at the renowned Lawrence Livermore National Laboratory. I would like to thank Carol Woodward for offering me the first of these opportunities, and many more afterwards. She has been an amazing mentor, whose insightful comments, remarks, and attention to detail have helped shape the way I do scientific research. Through her guidance and network of collaborators, I have had the opportunity to work with some very talented and prominent scientists. I am honored to have worked with Reed Maxwell, Andy Tompson, Dan Reynolds, Ravi Samtaney, and Glenn Hammond on various projects. Carol has been an ideal mentor in many ways, and has been very influential in my career development. I would also like to give credit to other friends and scientists at the lab who helped make my time there worthwhile. I would like to thank Rob Falgout and the HYPRE team for taking time to answer my questions about multigrid, and for helping me to resolve any solver issues I encountered during my work. I am very grateful to Steve Smith for being an awesome mentor and friend. Steve taught me a lot about the dos and don'ts of developing an efficient code. I owe thanks also to Milo Dorr for taking time to explain to me the concept of discretization on mapped-grids, and for sharing with me some clever ideas about the finite volume discretization scheme. I would also like to acknowledge the lunch-time group of friends with whom I had lots of interesting and entertaining discussions about sports, the weather, technology, movies, the food in the cafeteria, and many more. Many thanks to Rob, Steve, Geoff, Andy, Tzanio, and Vesselin for making lunch time fun.

My time at Livermore has been very rewarding, as I have had the opportunity to

be involved in some very interesting research. I would like to thank Sam Schofield, Al Nichols, and Phil Sterne for the opportunity to work with them, and for their confidence in my abilities. It was a rewarding experience. My sincere gratitude also goes to the entire ISCR staff for their tireless effort in ensuring that the summer program is top notch. I have found it to be very engaging, stimulating, and at the same time, fun. I would like to thank especially Tony Baylis, Linda Becker, Erica Dannenberg, Jan Winfield and Tami Campbell for ensuring that my time at the Lab was productive and enjoyable.

Being a part of the scientific computing group has been very beneficial, as I have had the chance to work and interact with some very talented and inspiring people. My sincere gratitude goes to Scott MacLachlan, who has been a good friend and mentor from the very beginning of my graduate education. Scott gave me my first introduction to multigrid methods, and the many discussions I had with him greatly improved my understanding of multilevel methods and their theory, or lack thereof. I am fortunate to have had the opportunity to work on a research project with Scott, in collaboration with Yousef Saad. This work represents a significant contribution to Chapter 3 of this thesis, and for this I am very grateful. I owe a lot of thanks also to Suzanne Shontz, for her friendship, mentorship and insightful advice on issues pertaining to graduate education and career development. Suzanne was the driving force behind my initial interest to intern at a national laboratory, and consequently influenced the first of such opportunities. I am very grateful to Suzanne for the genuine interest she has shown in my progress as a graduate student, and in my professional development. I would like to acknowledge Yunkai Zhou for his help during the early stages of my graduate education and for serving as a valuable resource whenever I had questions about numerical linear algebra. I would also like to thank Zhongze Li for his part in developing the initial version of the pARMS code, and consequently for his contribution to the current version of the code. I am thankful for his contribution to my knowledge and understanding of parallel solvers, which formed an important part of my graduate research, and is the subject of Chapter 6 of this thesis. My thanks also goes to other current and past members of the scicomp group, in particular, Jok Tang, Pierre Carrier, Da Gao, Haw-Ren Fang, Jie Chen, Thanh Ngo, Ruipeng Li, and Eugene Vecharynski. Their friendship and support has been very inspiring. I am grateful to them for taking time to sit through

my practice talks, and for their feedback and constructive criticisms that helped me to improve the final versions. I especially want to thank Jok Tang and Pierre Carrier, with whom I have had the opportunity to do some very interesting work on the application of numerical linear algebra techniques, in the area of dynamical mean field theory (DMFT). Sincere thanks also goes to Ruipeng Li for the collaborative effort, which contributed to Chapter 4 of this thesis.

I wish to thank all my instructors who have inspired me, and have played an important role in my success. In particular, I would like to thank Ms. Angela Addy, my middle school teacher and one of the most caring teachers I have ever had. I would also like to thank my undergraduate professors, especially Prof. Sandra Monteferrante, Prof. Fred Rispoli, Prof. John Vargas, Prof. Raymond Grinnell, Prof. Herbert Bernstein, and Prof. Paul Abramson. I am grateful to them for their training and support, which equipped me with the basic tools I needed to be successful in graduate school. My sincere gratitude also goes to faculty of the computational science program in the School of Computing and Mathematical Sciences at the University of Greenwich in London, U.K. In particular, I wish to thank Dr. Mayur Patel, Prof. Koulis Pericleous, Prof. Chris Bailey, and Prof. Choi-Hong Lai for their insightful instruction during my MSc. education. They motivated me to learn more about computational science and engineering, which ultimately led to my interest in scientific computing.

I am very grateful to all my friends and acquaintances for their support and friendship during my graduate studies. I wish to thank my good friend Getiria Onsongo, who has been a positive source of motivation and encouragement. I wish to also thank Daniel Achina for his role in ensuring a smooth transition when I first moved to Minnesota. He showed me the ropes and introduced me to a few new friends. My gratitude also goes to Evans Ampomah and Adam Jundt for their kind support and encouragement over the past few years.

Finally, I wish to offer my sincere gratitude to my family for their unwavering love and support over the years. I would like to especially thank my mother Rose Sarfo, and my father Kofi Osei, who have been a constant source of encouragement and inspiration from my childhood on. Their love, prayers, and faith in me, kept me focused to complete my PhD, and for this I will be forever grateful. I wish to thank my brothers for their support, words of encouragement, and for always believing in me. I am also very

grateful to Ed and Liz McCarthy for their love, care and support. I really appreciate their help and confidence in me, and for treating me like a son. Last but certainly not least, I wish to say a very big thank you to my beloved wife Becca, for her patience and unwavering support over the last five years. She has been my biggest fan, encouraging me during the hard times, and celebrating with me every success in the progress of my doctoral degree. I am very grateful to her for inspiring and motivating me during my doctoral research, and especially, for her patience and understanding while this thesis was being written.

Daniel Osei-Kuffuor
University of Minnesota, 2011

Dedication

To my loving parents, my mother Rose Gyamfua Sarfo, and my father Kofi Afram Osei.

Abstract

Linear systems originating from certain applications in the physical sciences can be challenging to solve by iterative methods. In the past, direct methods like Gaussian elimination have been often used to solve these systems, due to their robust nature. However, as problems are now often formulated in 3-D geometries, the use of direct solvers is becoming prohibitive. Moreover, unlike iterative solvers, direct solvers cannot be easily parallelized. Two classes of methods have attracted the interest of researchers in recent years. First, is the class of multigrid methods, which have been shown to be very efficient for elliptic-type problems, and for which various adaptations have been brought. Second, is the class of preconditioned Krylov subspace methods. Here, work has focused mainly on improving the preconditioning. The underlying challenge is to develop or identify techniques that are robust and efficient in terms of accuracy, stability, and scalability. As problems in computational science continue to evolve, so do preconditioning techniques, as new ideas are incorporated to develop solver technologies that are well adapted to solve novel and challenging problems. To further contribute to this endeavor, my research uses ideas from numerical linear algebra to address algorithmic and computational issues in the numerical solution of application problems. The ideas presented in this thesis will focus on improving the stability and effectiveness of ILU-based preconditioners to better handle challenging problems characterized by indefinite and poorly conditioned linear systems.

First, we present various strategies designed to improve the quality of the ILU factorization in order to safeguard stability without compromising the accuracy of the resulting factors. These strategies introduce new contributions to the areas of shifted ILU methods, modified ILU and compensation-based techniques, and reordering techniques for ILU. The resulting factors yield good and more robust preconditioners that are effective on highly indefinite and ill-conditioned linear systems.

Next, we demonstrate the effectiveness of combining ILU factorizations with multilevel methods. Multilevel ILU methods have become a popular area of research, as researchers seek to take advantage of the superior robustness of ILU, coupled with the

efficiency and scalability of multilevel methods. We discuss issues related to constructing the next-level matrices in the multilevel hierarchy and present ideas from multilevel graph coarsening and reordering strategies to construct an algebraic multilevel ILU-based preconditioner.

Finally, we discuss key concepts necessary for an efficient adaptation of the ILU-based preconditioner into a parallel framework. We discuss the parallel implementation within the structure of the parallel algebraic multilevel solver (pARMS) framework. We demonstrate how different preconditioning techniques may be incorporated into this framework, and discuss issues relating to scalability.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The numerical solution of many problems in science and engineering requires solving large sparse linear systems of equations. The basic problem is to solve the linear system

$$Ax = b \tag{1.1}$$

where A is a square matrix of size n with real or complex-valued entries, b is the $n \times 1$ right hand side vector, and x is the $n \times 1$ solution vector. For many applications, such as wave propagation phenomena, the linear system can be indefinite, which is characterized by the matrix A having eigenvalues that lie on both sides of the imaginary axis. The system may also be ill-conditioned and have eigenvalues clustered close to the origin. The result is that the numerical solution of the problem will be highly sensitive to small changes (noise) in the right hand side of the linear system. In the past, iterative methods have not been too successful with such problems and for this reason, direct methods have often been used. In recent years, the need for a more accurate representation of the physical problem has led to the development of large scale 3D models. The resulting linear systems are much larger than before, and are intrinsically harder to solve. Furthermore, the efficient solution of these large scale models requires solvers that scale well. For example, in molecular dynamics, the study of liquid/solid transitions in metals requires a large scale model of the problem in order to obtain an accurate simulation that is independent of size [1]. Direct methods, by

their nature, are not very scalable, which makes them rather inefficient for large scale simulations. Furthermore, they cannot be easily parallelized, which makes them less favorable candidates for the parallel solution of linear systems. Iterative methods have thus become a more attractive choice for large scale simulations.

Multigrid methods [2–6] are one class of iterative solvers that have attracted interest in recent years. They are known to be ideal for solving problems originating from elliptic partial differential equations (PDEs). Multigrid theory is based on the use of a hierarchy of grids or discretizations, to solve a coarse problem, in order to obtain a global correction to the solution. A simple two-level scheme is shown in Figure 1.1.

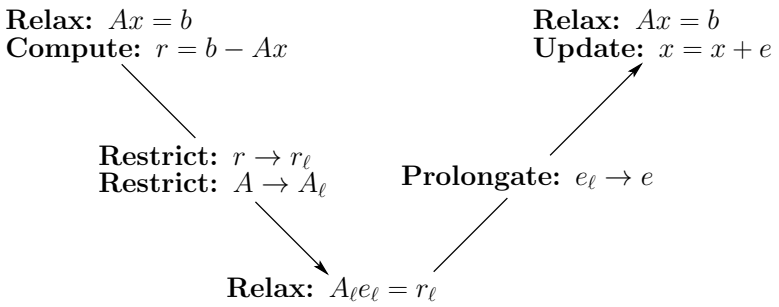


Figure 1.1: A simple schematic of a 2-level multigrid process.

The residual equation at the coarse grid can be solved by exploiting recursion. Each coarse system is approximately solved by a few relaxation steps, known as smoothing. The resulting residual is then matched onto the next coarser grid by a process known as restriction. The new coarse grid system is then approximately solved, and the process continues until the final coarse system is obtained and solved to get a correction to the solution at the coarsest level. This correction is then prolongated through the hierarchy of coarse grids until the correction to the original (fine grid) problem is obtained, and the solution updated. This recursive procedure to update the solution is known as a v-cycle. The updated solution may be used as an initial guess for the same problem, and the entire process repeated again to obtain a more refined solution.

Geometric multigrid methods [2–5] are well known for their computational efficiency. They rely on the geometric structure to fix the choice of the coarse grid operators, and select an appropriate smoother to obtain a fast convergence for the numerical solution.

The convergence theory for the geometric multigrid algorithm shows convergence with an $O(N)$ computational complexity. This result implies that the geometric multigrid algorithm is highly scalable, and hence is an attractive choice for large scale simulations. However, this theory only holds for the solution of elliptic-type problems on regular grids. For linear systems derived from more general (non-elliptic) applications, and systems derived from problems with irregular or unstructured grids, the geometric multigrid method can be quite ineffective as a solver.

Several adaptations have been brought to make multigrid methods amenable to a wider class of problems. The algebraic multigrid method (AMG) [5–9] is a generalization of the multigrid idea to handle problems whose solution with geometric multigrid is unfeasible, for example discretized problems on unstructured grids. Unlike geometric multigrid, AMG does not assume any underlying geometry. The smoother is fixed, and an appropriate strategy is used to select the coarse grid operators that constitute the hierarchy of grids. The strategies for selecting the coarse grid operator are based solely on the entries in the coefficient matrix of the linear system under consideration. This algebraic way of defining the grid hierarchy implies that large jumps and anisotropies in the coefficient matrix are better captured by the AMG algorithm. Thus, in general, AMG is a more robust solver than geometric multigrid.

One major drawback of the algebraic multigrid method is that its convergence is not fully understood, as it does not obey to the same convergence theory as geometric multigrid. Furthermore, like geometric multigrid, it is most effective on the class of elliptic partial differential equations. As a result, AMG is only considered advantageous where the solution by geometric multigrid is ineffective [5].

While multigrid methods are not ideal as general purpose solvers for indefinite or general sparse linear systems of equations, they can be used as preconditioners in conjunction with Krylov accelerators, to yield efficient iterative solvers.

Krylov subspace methods [10] belong to a class of iterative solvers called projection methods. They extract approximate solutions to the linear system 1.1 from the subspace

$$span\{r_0, Ar_0, ..., A^{m-1}r_0\},$$

where $r_0 = b - Ax_0$ is the initial residual for an initial guess x_0 . The approximate

solution lies in the affine space

$$x_0 + span\{r_0, Ar_0, ..., A^{m-1}r_0\}.$$

This essentially means that the solution is approximated by a polynomial of degree $m-1$, where m is the Krylov subspace dimension. Krylov methods do a fairly good job at resolving the instabilities of the linear system when only a few eigenvalues are close to the origin. However, when there is a significant cluster of eigenvalues close to the origin, the convergence of the Krylov method significantly deteriorates.

1.2 Preconditioned Krylov Solvers

The generalized minimal residual (GMRES) algorithm of Saad and Schultz [11] is a commonly used Krylov method designed for general (non-symmetric) linear systems. The algorithm uses the Arnoldi method to find the vector of minimal residual in the Krylov subspace, and approximates the solution to the linear system by this vector. A restarted version, GMRES(m) [12], can be invoked to reduce the cost per iteration, by fixing the size of the Krylov dimension. Once the iteration count reaches the subspace dimension m, the algorithm is restarted using the current solution x_m as the initial guess. Variants of this method include deflated GMRES (DGMRES) [13, 14], which uses eigenvector information from the previous restart, to accelerate convergence; and flexible GMRES (FGMRES) [15], which allows for a different preconditioner to be used at every step in the iteration. This means that any other iterative method may be used as a preconditioner within the FGMRES framework. The work of this thesis will rely on flexible GMRES as the main accelerator.

Like other Krylov solvers, (F)GMRES benefits when the eigenvalues of the linear system are clustered enough away from the origin. Otherwise a preconditioner may be necessary to speed-up its convergence. Much of the recent effort on preconditioned Krylov solvers has focused mainly on improving the preconditioning. A preconditioner, M, transforms the linear system in 1.1 into the form:

$$M^{-1}Ax = M^{-1}b,$$

for left preconditioning,

$$AM^{-1}y = b, \text{ where } y = Mx$$

for right preconditioning, or

$$M_L^{-1} A M_R^{-1} y = M_L^{-1} b, \text{ where } y = M_R x$$

for centered preconditioning with $M = M_L M_R$. Here, the requirement for M is that it must be inexpensive and easy to construct, and the operation $M^{-1}v$, where v is some column vector, must be cheap to perform. Consequently, from the Gershgorin theorem [16–18], an obvious choice for the preconditioner is to simply extract M from A, so that $M^{-1}A$ is approximately the identity matrix. This means that the eigenvalues of the preconditioned matrix, $M^{-1}A$, should approach 1.0. The better the approximation M is to A, the closer the spectrum of $M^{-1}A$ is to 1.0. However, for indefinite matrices, the approximation to A can cause some eigenvalues of the preconditioned matrix, $M^{-1}A$, to be trapped close to zero, resulting in a slow convergence for the Krylov method [19,20].

The need to construct efficient preconditioned solver strategies, that are also effective on indefinite problems, is therefore an active area of research. Some of the most general purpose preconditioning techniques available are based on extracting M from A by performing some form of incomplete factorizations on A. These include the incomplete LU factorization (ILU) (or incomplete Cholesky (IC) for the symmetric version) [10], the approximate inverse [7,21], and Vaidya's [22,23] preconditioners. Vaidya's preconditioners require that the system matrix ,A, be symmetric and diagonally dominant. This makes them not very suitable for indefinite problems. Approximate inverse methods are also known to have limited success, particularly for indefinite problems [24]. However, incomplete LU factorization (ILU) techniques can be quite robust as preconditioners for iterative solvers.

1.3 Incomplete LU Factorization Preconditioners

Standard ILU techniques are a well-known class of preconditioners used to accelerate the convergence of Krylov methods for the iterative solution of linear systems of equations. ILU factorization methods were initially developed in the 1960s and early 1970s with the initial works of Buleev [25], Oliphant [26], Varga [27], Dupont, Kendall, and Rachford [28], Axelsson [29] and others. The 1977 paper by Meijerink and Van der Vorst [30] marked a turning point. This paper proved some results for M-matrices and showed a

few extensions of the method. However, it contributed even more by publicizing the method as a possible general-purpose technique for solving sparse linear systems.

When Gaussian elimination is applied to a sparse matrix, A, a large number of nonzero elements in the factors, L and U, may appear in locations occupied by zero elements in A. These 'fill-ins' often have small values and, therefore, they can be dropped to obtain a sparse approximate LU factorization, referred to as an incomplete LU (ILU) factorization. The simplest of these procedures, ILU(0), is obtained by performing the standard LU factorization of A and dropping all fill-in elements generated during the process. Thus, the factors, L and U, have the same pattern as the lower and upper triangular parts of A (respectively).

In the early work on ILU preconditioners, it was understood that ILU(0) could be ineffective and that more accurate factorizations could be needed. Such factorizations, denoted by ILU(k) and IC(k) (for incomplete Cholesky in the symmetric case), were initially derived by adopting a strategy to drop fill-ins according to their so-called 'level-of-fill' parameter, k, first defined in the reservoir simulation literature [31]. This level-of-fill concept associates each fill-in element with the 'level' at which it was formed. Initially, all non-zero entries of the matrix are assigned level zero. Then level k fill-ins are generated by products of fill-in elements at levels less than k. ILU(k), then, arises by keeping all fill-ins that have level k or less, and dropping any fill-in whose level is higher.

1.3.1 Threshold-based ILU Factorizations

Since the level-of-fill concept was founded on properties of M-matrices, alternative techniques were soon after developed for general sparse matrices. One of the first contributions along these lines is by Munksgaard [32], who defined an ILU factorization that uses a drop tolerance. Another method in the same class is ILU with Threshold (ILUT) [33]. ILUT is a procedure based on a form of Gaussian elimination in which the rows of L and U are generated one by one. This row-wise algorithm is based on the so-called IKJ (or delayed update) Gaussian elimination process, whereby the i-th step computes the i-th rows of L and U:

Algorithm 1: IKJ-ordered Gaussian Elimination.

0. For $i = 1 : n$, do:
1. $\mathbf{w} = A_{i,1:n}$
2. For $k = 1 : i - 1$, do:
3. $w_k := w_k / u_{k,k}$
4. $\mathbf{w}_{k+1:n} := \mathbf{w}_{k+1:n} - w_k \cdot U_{k,k+1:n}$
5. Enddo
6. For $j = 1, \ldots, i - 1$, $l_{i,j} = w_j$ $(l_{i,i} = 1)$
7. For $j = i, \ldots, n$, $u_{i,j} = w_j$
8. Enddo

Here, and in all following discussion, $a_{i,k}$, $l_{i,k}$, and $u_{i,k}$ represent the scalar entries at the i-th row and k-th column of the matrices A, L, and U, respectively, $A_{i,1:n}$ denotes the complete i-th row of A (transposed as a column vector) while $A_{1:n,j}$ denotes the j-th column of A, $\mathbf{w}_{k+1:n}$ denotes the last $n-k$ entries in the vector $\mathbf{w}$, $U_{k,k+1:n}$ denotes the last $n - k$ entries in the k-th row of U (transposed as a column vector), $L_{i,1:i-1}$ denotes the first $i - 1$ entries in the i-th row of L (transposed as a column vector), and so forth. Of note in Algorithm 1 is that at the i-th step, the i-th row of A is modified by previously computed rows of U, while the later rows of A and U are not accessed. The incomplete version of this algorithm is based on exploiting sparsity in the elimination and dropping small values according to a certain 'dropping rule'.

ILUT: Row-based Implementation

The dropping strategy for the ILUT algorithm proposed in [33] uses two parameters. The first parameter is a drop tolerance, τ, which is used mainly to avoid doing an elimination if the pivot is too small. The second parameter is an integer, p, which controls the number of entries that are kept in the i-th rows of L and U. Details can be found in [10, 33]. An illustration of the elimination process is shown in Figure 1.3.1, and a sketch of the general structure of the algorithm is given next as Algorithm 2.

Algorithm 2: IKJ-ordered ILUT.

0. For $i = 1 : n$, do:
1. $\mathbf{w} = A_{i,1:n}$
2. For $k = 1 : i - 1$ and if $w_k \neq 0$, do:

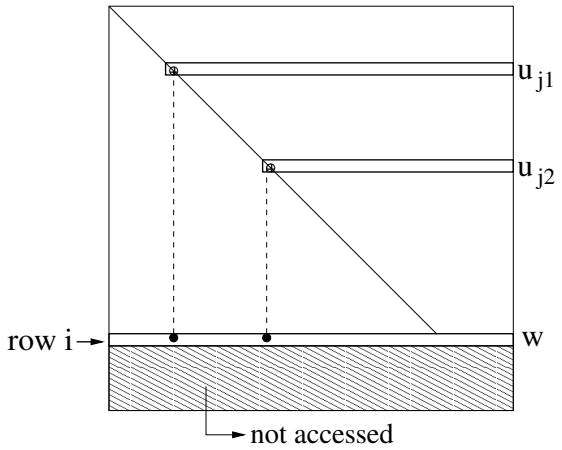


Figure 1.2: Illustration of the row-based ILUT algorithm

3. $w_k = w_k/u_{k,k}$
4. Apply first dropping rule to w_k
5. If w_k is not dropped, $\mathbf{w}_{k+1:n} = \mathbf{w}_{k+1:n} - w_k \cdot U_{k,k+1:n}$
6. Enddo
7. For $j = 1, \ldots, i-1$, $l_{i,j} = w_j$ $(l_{i,i} = 1)$
8. Apply second dropping rule to $L_{i,1:i-1}$
9. For $j = i, \ldots, n$, $u_{i,j} = w_j$
10. Apply second dropping rule to $U_{i,i+1:n}$
11. Enddo

As shown in Figure 1.3.1, and in Lines 3 and 4 of Algorithm 2, the pivot, w_k, is computed and compared with the dropping parameter, τ, and dropped if it is smaller relative to some scaling parameter. Otherwise, operations of the form $\mathbf{w} = \mathbf{w} - w_k \mathbf{u}_k$ are performed to eliminate the entry associated with the pivot entry w_k. Here, $\mathbf{u}_k$ is used to denote the k-th row of U, and $\mathbf{w}$ is a working vector initially set to the i-th row of A. In Lines 8 and 10, the threshold, τ, is invoked again to drop small terms, then the largest p entries in the resulting i-th rows of L and U are kept, and the others are dropped.

ILUT: Column-based Implementation

The above algorithm is row-based; for column-oriented storage schemes (such as within Matlab), however, a column-based approach is more efficient. Furthermore, the triangular solves involving the L and U factors can be efficiently performed using a column-oriented data structure. For the column version of ILUT, at a given step j, the initial j-th column of A, $\mathbf{a}_j$, is transformed by zeroing out entries above the diagonal element. As in the row version, at the start of step j a working column, $\mathbf{w}$, is created, which is set to $\mathbf{a}_j$. Then operations of the form $\mathbf{w} := \mathbf{w} - w_k \mathbf{l}_k$ are performed to eliminate entries of $\mathbf{w}$ from top to bottom, until all entries strictly above the diagonal are zeroed out. In the incomplete LU case, only a few of these eliminations are performed. An illustration is shown in Figure 1.3.1, and the complete algorithm is given in Algorithm 3. Each elimination begins by seeking the top element of $\mathbf{w}$, which is, say, in position i_1. This entry is then either dropped (by the first dropping rule), or the column $w_{i_1}\mathbf{l}_{i_1}$ is subtracted from $\mathbf{w}$ to eliminate the entry w_{i_1}. We then seek the next entry in $\mathbf{w}$ to eliminate, and this process continues until all entries above the diagonal are zero. We then apply a second dropping rule to obtain a pruned version of $\mathbf{w}$. Note that in Algorithms 2 and 3, the elimination of the entry in $\mathbf{w}$ is only implied, as the entry is not actually zeroed out (see Lines 3 and 5 in Algorithm 2 or Lines 1 and 4 in Algorithm 3). The eliminated positions will contain the pivot entries which become entries in L for the row version (see Line 7 of Algorithm 2), or U for the column version (see Line 8 of Algorithm 3). The elimination steps can be written in equation form as

$$\mathbf{a}_j - w_1\mathbf{l}_1 - w_2\mathbf{l}_2 \cdots - w_{j-1}\mathbf{l}_{j-1} = \hat{\mathbf{w}} + \epsilon_U + \epsilon_L, \tag{1.2}$$

where $\mathbf{l}_k$ is a column of L with $k < j$, and w_k is the coefficient used for the elimination of pivot a_{kj}. In order to avoid confusion in the notation, we introduce $\hat{\mathbf{w}}$ as the version of the transformed $\mathbf{w}$ with zero entries in positions $1, \cdots, j-1$. Furthermore, for the sake of simplifying notation, we write Equation (1.2) for dense columns, so the scalars w_k are understood to be zero except for a few. The column ϵ_U contains the terms, w_k, which were not eliminated (dropped by first dropping rule). The column ϵ_L contains the entries dropped by the post-dropping on $\mathbf{w}$, which is achieved by the second dropping rule. The resulting column, $\hat{\mathbf{w}}$, which now has zeros above position j, is divided by its diagonal entry and becomes the j-th column of L, while the scalars w_k, representing

the eliminated pivot a_{kj}, will constitute the j-th column of U. Dropping in Lines 3, 7, and 9 of Algorithm 3 may be handled in the same way as the row variant described previously (in Algorithm 2).

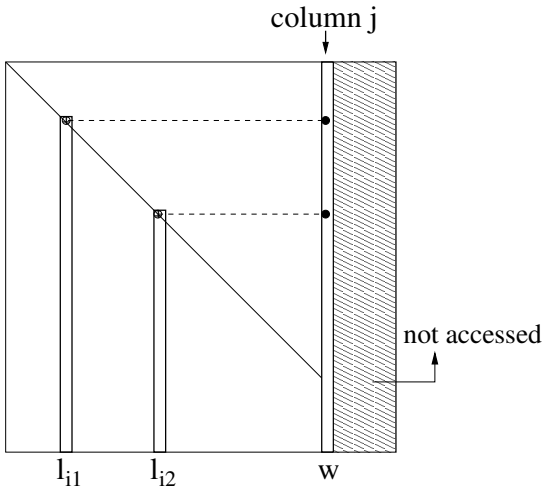


Figure 1.3: Illustration of the column-based ILUT algorithm

Algorithm 3: Left-looking or JKI ordered ILUT.

0. For $j = 1 : n$, do:
1. $\mathbf{w} = A_{1:n,j}$
2. For $k = 1 : j - 1$ and if $w_k \neq 0$, do:
3. Apply first dropping rule to w_k
4. If w_k is not dropped, $\mathbf{w}_{k+1:n} = \mathbf{w}_{k+1:n} - w_k \cdot L_{k+1:n,k}$
5. Enddo
6. For $i = j + 1, \ldots, n$, $l_{i,j} = w_i / w_j$ $(l_{j,j} = 1)$
7. Apply second dropping rule to $L_{j+1:n,j}$
8. For $i = 1, \ldots, j$, $u_{i,j} = w_i$
9. Apply second dropping rule to $U_{1:j-1,j}$
10. Enddo

On the practical side, it has often been observed that while the second parameter, p, does help in reducing the storage requirement of the algorithm, it sometimes leads to

poorer performance. In other words, it seems better in practice to take a large limiting parameter, p, and rely more on the parameter, τ, to control dropping. This phenomenon is based only on observation and does not appear to have been analyzed.

1.3.2 Accuracy and Stability of the ILU Factorization

Two important considerations must be addressed when constructing an incomplete factorization (or, indeed, any other) preconditioner. Of primary importance from a theoretical point of view is the accuracy of the preconditioner. This is typically expressed in terms of the spectral equivalence of the preconditioner, M, to the system matrix, A, expressed by conditions such as

$$\alpha \mathbf{x}^T M \mathbf{x} \leq \mathbf{x}^T A \mathbf{x} \leq \beta \mathbf{x}^T M \mathbf{x} \quad \forall \mathbf{x}, \tag{1.3}$$

when A and M are both assumed to be symmetric and positive definite. Here, the performance of the Krylov accelerator may be bounded in terms of the spectral equivalence bound, $\frac{\beta}{\alpha}$, and this bound may be sharp, if the spectrum of $M^{-1}A$ is roughly evenly distributed between α and β. If, however, the spectrum is significantly clustered, this bound may be insufficient, since only an upper bound is guaranteed by the theory. On the other hand, of significant importance from a practical (or computational) point of view is the stability of the preconditioner. Standard ILU factorizations can be prone to instabilities, characterized by two main forms. The first is the result of unstable pivots during the factorization, which can lead to extremely large (off-diagonal) entries in the L and U factors. This results in a factorization that is only a poor approximation to the original matrix (poor spectral equivalence), and hence will be ineffective as a preconditioner. In the past, column (or row) pivoting techniques have been typically employed to ensure stability during the factorization. These techniques permute the columns (or rows) of the matrix during the factorization, so that columns (or rows) that are likely to yield unstable pivots are moved to the end of the factorization. Since the matrix is sparse, columns (or rows) near the end of the factorization are less likely to contribute to the elimination of subsequent columns (or rows) during the factorization. Thus, moving the columns (or rows) with unstable pivots to the end will have no effect on the stability of the factorization (up to this point), since these pivots may not be used in the factorization. Diagonal compensation strategies have also been proposed in the literature to

help eliminate unstable pivots. This concept will be addressed later. The second form of instability occurs when solving with the resulting L and U factors lead to recurrences that grow exponentially. This was first observed by Elman [34], who showed that disastrous situations can arise wherein the norm of $U^{-1}L^{-1}$ can be huge, even though A, and the factors L and U, are (relatively) well-behaved. This results in unstable triangular solves when computing the solution to the system $LUx = y$ [35]. For problems of current interest, this appears to be more common with ILUT than with the level-based ILU(k). Some recent work has been devoted to correcting this problem. A paper by Bollhöfer [36] proposed more rigorous ways of dropping small entries in ILUT-type factorizations by precisely monitoring the growth of the inverse factors. This method consists of using condition-number estimation techniques and defining dropping so that the resulting estimated condition number will not be too large. In [37], it was argued that this approach is well-suited to the Crout version of ILU.

The argument behind these inverse-based ILUs is based on writing the matrix, A, as a sum of the preconditioner plus an error term, $A = M + E$, and, then, considering the resulting preconditioned matrix, $M^{-1}A = I + M^{-1}E$. When an iterative method is applied to this preconditioned system, what matters is how well $I + M^{-1}E$ approximates the identity matrix. In exact arithmetic, all that matters is the spectral equivalence condition in Equation (1.3); however, for practical computation, the boundedness in norm of $M^{-1}A$ (or, equivalently, of $M^{-1}E$) is also important. The stability of a preconditioner can be measured directly by $\|M^{-1}E\|$ in the L^1 or L^∞ norm. For many practical cases, this norm can be much larger than the spectral radius of the preconditioned system, and this mismatch leads to significant differences between the theoretical and actual performance of the preconditioned Krylov iteration. Indeed, ILUT is quite prone to this type of instability for indefinite problems; it is possible to construct examples where $\|M^{-1}\|_1$ is arbitrarily large for simple matrices, A. Later, we shall discuss some new ideas based on modification and compensation strategies, aimed at improving or maintaining the stability of the ILUT factors.

1.3.3 Reordering and Multilevel ILU Factorizations

For poorly structured matrices, the ILU factorization could generate significant fill-ins, making the resulting L and U factors dense and hence inefficient for preconditioning.

Reordering the matrix by permuting the columns and rows based on the underlying grid or sparsity pattern have been shown to help correct this problem [38]. The reverse Cuthill-McKee (RCM) algorithm [39] is a common technique used to reorder sparse symmetric matrices. The algorithm yields a permutation that reduces the bandwidth of the matrix, which can lead to fewer fill-ins during the Gaussian elimination process. The minimum degree algorithm [40–43] is another algorithm used to permute the rows and columns of a sparse symmetric matrix. This algorithm is specifically designed to reduce the number of non-zeros in the resulting L, U (or Cholesky for Cholesky decomposition) factors. Improved variants of the algorithm, such as multiple minimum degree(MMD) and approximate minimum degree (AMD) [43], have been developed with the goal of reducing the computational cost of the original algorithm. Nested dissection techniques [38, 44] have also been successfully used to reduce fill-ins during the LU decomposition. The algorithm is based on a divide-and-conquer heuristic, which can be used to reorder the matrix to generate independent blocks of submatrices that can be eliminated simultaneously. Most of the fill-ins generated during the Gaussian elimination process is contained within the submatrices, with very few occurring between the blocks of submatrices.

The above algorithms have been typically designed for the solution of sparse symmetric linear systems by direct methods. Nonetheless, while they may be applicable to incomplete factorizations, they may not be the optimal approach. For incomplete LU factorizations, obtaining a stable factorization is one of the most important considerations. The occurrence of unstable pivots during the ILU factorization can result in a very poor preconditioner. Column pivoting strategies have been used to safeguard the stability of the factorization during the elimination process [10,45–47]. In the context of threshold-based incomplete factorizations, ILUT with pivoting (ILUTP) [10], is known to be somewhat more robust than ILUT. However, ILUTP is less memory efficient as pivoting with ILU tends to generate more fill-ins.

As problems become larger and more complex much of the recent research effort on solving sparse linear systems by iterative techniques has been devoted to the development of effective, robust, and scalable preconditioners. Multigrid techniques [9, 48] and their algebraic counterparts (AMG) [6, 49] are known to be effective and optimally efficient for solving discretized elliptic partial differential equations. Nonetheless, they

may become ineffective when faced with more general types of sparse linear systems, such as systems that are highly indefinite. Many authors have advocated multilevel methods that combine the robust and general-purpose qualities of ILU preconditioners, with the multilevel framework of multigrid [50–60]. A key ingredient in any multilevel scheme is to find an independent set ordering P, which permutes the matrix A into the form:

$$PAP^T = \begin{bmatrix} B & F \\ E & C \end{bmatrix}, \tag{1.4}$$

where B is a block diagonal matrix corresponding to an independent set. A block factorization of the permuted matrix is then obtained as:

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} B & 0 \\ E & I \end{bmatrix} \begin{bmatrix} I & B^{-1}F \\ 0 & S \end{bmatrix}, \tag{1.5}$$

and the process is continued recursively on the Schur complement matrix ,$S = C - EB^{-1}F$. There are several considerations for choosing the permutation, P, see for instance [36, 53, 60–64]. The general idea is to choose P so that the approximation of the B-block, by an incomplete factorization, is stable. This is usually achieved by ensuring that the B-block is diagonally dominant. This allows for a sparse approximation of B, which benefits memory efficiency and reduces computational costs during the preconditioning operation. Another important aspect of the multilevel scheme is the formation of the next level matrix, that is, the Schur complement matrix S. Computing S in its exact form can be quite expensive, and so, in the context of preconditioning, an approximation is used instead [7, 36, 53, 60, 63, 64].

1.3.4 Adaptation to Parallel

[Parallel implementation and examples with pARMS ...]

1.4 Application Problems

The following describes the application areas used in this thesis.

1.4.1 Acoustic Wave Diffraction

The Helmholtz equation is a partial differential equation of the form

$$(\Delta + k^2)\Phi = f, \tag{1.6}$$

which governs the propagation of waves in media. In the above equation, f represents a harmonic source, and k represents the wave number. The numerical solution to the Helmholtz equation at high wave numbers has been the subject of extensive research. At high wave numbers, the system matrix tends to be very indefinite, causing problems for many numerical methods.

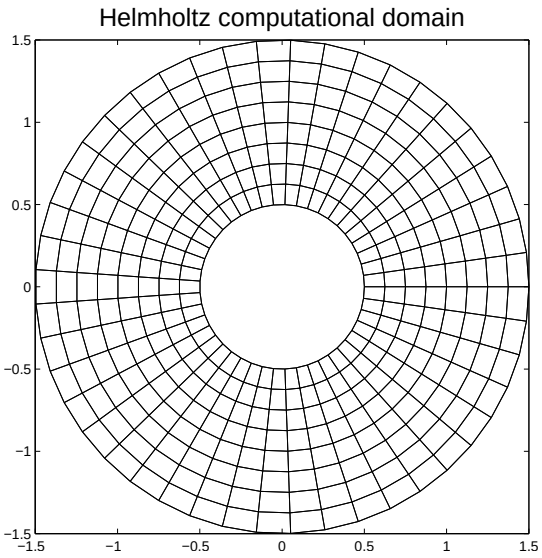


Figure 1.4: An example of the discretized domain mesh for the Helmholtz problem.

Our application problem is based on the simulation of the diffraction of an acoustic wave originating from infinity through an open medium, and incident on a bounded obstacle with boundary $\Gamma = \partial\Omega$, of circular shape. Here, we assume the the plane wave is propagating along the x-axis, and the radius of the bounded obstacle is 0.5m.

The corresponding boundary value problem (BVP) is as follows:

$$\begin{cases} \Delta u + k^2 u &= f, \text{ in } \Omega \in \mathbb{R}^2 \\ u &= \delta, \text{ on } \Gamma \\ \lim\limits_{r\to\infty} \sqrt{r}\left(\dfrac{\partial u}{\partial n} - iku\right) &= 0 \end{cases} \tag{1.7}$$

where f represents a harmonic source, k is the wave number and δ is determined by the use of Bessel functions. The last equation in the above BVP is referred to as the Sommerfeld radiation condition and it models the non-reflecting condition at the boundary, thereby guaranteeing a unique and physically meaningful solution to the above problem.

This BVP is however not suitable for solution via the finite element method, primarily because the condition that the incident wave originates from infinity prescribes the problem in an infinite domain. The problem is therefore reformulated to introduce an artificial boundary, by enforcing, for example, the so-called Dirichlet-to-Neumann (DtN) technique. Thus, the radiation condition at infinity is replaced by a boundary condition on the artificial boundary, Γ_{art}. The resulting problem becomes:

$$\begin{cases} \Delta u + k^2 u &= f, \text{ in } \Omega \in \mathbb{R}^2 \\ u &= \delta, \text{ on } \Gamma \\ \frac{\partial u}{\partial n} &= -Bu \text{ on } \Gamma_{art} \end{cases} \tag{1.8}$$

where B denotes the DtN operator. For the examples considered in this thesis, the artificial boundary condition is imposed at a distance of 1.5m from the obstacle. The resulting boundary value problem is discretized by the Galerkin least-squares finite-element method, using an isoparametric discretization over quadrilateral elements. The discretized system is of the form

$$([K] - k^2[M] + \imath k[C])\{u\} = \{f\} = \{0\},$$

where [K] is the stiffness matrix, [M] is the mass matrix, and C is the boundary matrix responsible for the introduction of complex terms in the discretization. The coefficient matrix is complex, symmetric but not Hermitian, and generally not diagonally dominant. It is also very indefinite for the higher values of the wave number k. Details of the reformulation of the BVP and the finite element discretization of the result may be found in the paper by Kechroud at al. [65] and references therein.

1.4.2 Crystal Simulation

The physical problem considered is a 3D buoyancy-driven flow in an enclosed cylinder containing molten cadmium zinc telluride. This material has generated a great deal of interest, due to its unusual thermo-physical properties. See for instance [66] and references therein. The flow within the cylinder is incompressible, and energy transfer within the domain is assumed to be controlled by conductive and convective forces only.

The underlying equations for the flow are the steady state Navier-Stokes equations including a Boussinesq approximation of the buoyancy force, which can be written in dimensionless forms as

$$\mathrm{Pr}^{-1}\mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v}-\boldsymbol{\nabla}\cdot\mathsf{T}-\mathrm{Ra}(T-1)\mathbf{e}_z=\mathbf{0}, \tag{1.9}$$

$$\boldsymbol{\nabla}\cdot\mathbf{v}=0, \tag{1.10}$$

where $\mathsf{T}=-p\,\mathsf{I}+(\boldsymbol{\nabla}\mathbf{v}+\boldsymbol{\nabla}\mathbf{v}^{\mathrm{T}})$ is the stress tensor, $\mathbf{v}$ is the velocity, p is the pressure, and T is the temperature. The Prandtl number Pr is set to unity, and the magnitude of the Rayleigh number, Ra, scales the intensity of buoyant thermal convection. Here, the vector $\mathbf{e}_z$ is the unit vector in the axial direction. The boundary conditions for Equations (1.9)-(1.10) are no-slip conditions on the cylinder walls:

$$\mathbf{v}=\mathbf{0}. \tag{1.11}$$

Heat transfer in the system is described using a steady state energy transfer equation, which is given by

$$\mathbf{v}\cdot\boldsymbol{\nabla}T-\boldsymbol{\nabla}^2T=0. \tag{1.12}$$

In this system, flow is driven by thermal gradients arising from the spatially-varying temperature gradients at the cylinder walls. The boundary condition for Equation (1.12) is a flux boundary condition that represents flux of energy conducted and convected from an outer environment (e.g., a furnace) to the cylinder walls. The boundary condition uses a parabolic temperature profile, which introduces a thermally destabilizing regime to the flow, and thus provides a more challenging test for the solver.

The equations are discretized by the Galerkin finite element method, using 15-noded tetrahedral elements. The numerical solution of the discretized nonlinear equations is

achieved by the use of Newton's iterations. The Jacobian matrix, which is associated with the resulting linear system that has to be solved at each Newton iteration, is generally non-symmetric and indefinite. It is also poorly conditioned, as a result of the incompressibility constraint in the velocity-pressure formulation. Furthermore, for high values of the Rayleigh number, a more refined mesh is needed for the discretization to adequately approximate the solution. This leads to a larger and more complicated system to solve. The entire numerical simulation of the problem is done with the Cats3D code from the Jeff Derby group[1] . Further details about the problem formulation and discretization may be found in the code documentation.

1.4.3 General Applications

This thesis will also make use of other general application models to test and illustrate the effect of various strategies discussed herein.

- Shifted Laplace: The base problem is a 2D finite-difference discretization of the Laplace operator, $-\Delta$, on a uniform grid using centered differences. The discretization assumes a scaling based on the mesh size parameter, h, so that the resulting matrix initially has 4 on the diagonal, and four off-diagonal entries of -1. This is then shifted by a small negative term, ϱ, so that the resulting matrix is indefinite.

- Normal Matrices: The normal matrix is defined as $A = L^T L$, where L is the matrix obtained from the shifted Laplace operator. Although the matrix A is symmetric and positive definite by definition, it is generally poorly conditioned (compared to L), which makes linear systems involving A challenging to solve.

1.5 Outline of Thesis

The outline of the thesis is as follows.

Chapter 2: Shifted ILU Methods: This chapter begins with a brief introduction about the motivation for using diagonal perturbations within the ILU factorization. A

[1] http://www.msi.umn.edu/∼derbyj/

successful shifted ILU scheme requires the careful selection of the shift. An analysis with shifted ILU(0) is given to further motivate the idea and a criterion for selecting the shift is given. A more general approach for selecting the shift is then presented to construct an effective shifted ILUT preconditioner for complex, symmetric, and indefinite linear systems. An illustration of the performance of the resulting technique is presented using numerical experiments from applications in acoustic wave scattering.

Chapter 3: Modified ILU and Compensation-based Techniques: Modified ILU methods are known to yield preconditioners that are more robust than standard ILU methods. This chapter first introduces the idea and motivation for modifying the ILU factorization. An extension to the classical modified ILU technique is then presented, based on an optimal modification criterion, in the context of the threshold-based ILU factorization (ILUT). This is further extended based on purely imaginary compensation strategies for complex linear systems. Numerical results are presented to show the performance of the new modified ILU scheme, compared to standard ILU methods.

Chapter 4: Reordering for ILU: ILU factorizations have been shown to benefit from some form of reordering. The primary considerations when reordering for ILU are to avoid excessive fill-ins, and to maintain or improve the stability of the factorization. This chapter presents ideas that combine graph coarsening techniques and multigrid methods to develop reordering strategies for ILU factorizations. Three main strategies are presented that utilize a multilevel graph coarsening approach to define a reordering for the ILU factorization. Numerical results are presented to show the effectiveness of this reordering, compared to standard reordering strategies like reverse Cuthill-McKee and nested dissection, on applications from acoustic wave scattering.

Chapter 5: Multilevel ILU: This chapter is devoted to a discussion of the use of ILU factorizations within a multilevel framework. Details about the multilevel scheme are presented, together with a discussion about its key features. Numerical examples are used to illustrate the performance of multilevel ILU methods on indefinite systems, compared to single-level ILU methods. A discussion about the possible contributions

of modification and compensation-based techniques and reordering for ILU, within the multilevel approach, is also presented.

Chapter 6: Adaptation to parallel: This chapter discusses a general approach to adapt the ILU factorization into a parallel framework. This follows a domain decomposition strategy for parallel preconditioning. The parallel ILU solver, pARMS[2] , is briefly introduced, and numerical examples are presented on applications from crystal simulation. Various challenges to obtaining an efficient parallel solver are discussed, and some ideas are proposed to improve the quality of the parallel solver.

Chapter 7: Conclusion: A summary of the main ideas of this thesis, together with some suggestions for future work, is presented in Chapter 7.

[2] http://www-users.cs.umn.edu/~saad/software/pARMS/

Chapter 2

Shifted ILU Methods

Shifted ILU methods belong to the class of methods that use diagonal perturbation techniques to improve the quality of the incomplete LU factorization. Diagonal perturbation was first used by Kershaw [67] to remove unstable pivots during an incomplete Cholesky factorization. It has since become an important ingredient in the construction of effective preconditioning strategies to handle indefinite problems, such as problems originating from electromagnetics and quantum physics. The idea is to obtain new matrix, B, as a diagonally shifted approximation to the original matrix A, so that the ILU factorization on B yields factors that are more stable than that of A. In real arithmetic, the matrix B is formally defined as:

$$B = A + \alpha I_n,$$

where I_n is the $n \times n$ identity matrix, and $\alpha \in \mathbb{R}$ is known as the shift. In [68], Manteuffel proved some results on the existence of an ILU factorization for the matrix B, where A is symmetric and positive definite (SPD), and $\alpha > 0$. It is clear that while the ILU factorization of a SPD matrix is not is not guaranteed to exist, with an appropriate diagonal shift, the factorization exists [69].

2.1 Shifted ILU(k) for Structured Problems

Standard level of fill-based incomplete factorizations, such as ILU(k) (or IC(k) for the symmetric case), can be attractive for preconditioning matrices that exhibit M-matrix

properties. However, their usage for general sparse linear systems is often limited, partly due to the lack of a guaranteed existence of the factorization, even for some symmetric positive definite (SPD) matrices. Their shifted variants, however, have been successfully used to precondition systems arising from the Normal Equations. It is easy to see that the quality of the resulting factorization, and hence the overall performance of the preconditioner, varies with the shift α [10, 70]. This variation suggests that there is an optimal value for α, for which the preconditioner achieves the best performance. Unfortunately, finding this optimal choice for α is a non-trivial problem, and different authors have proposed different techniques that satisfy some design criterion. Here, we present a strategy for defining the shift for ILU(k) on problems defined on grids with a regular stencil structure.

2.1.1 Shifted ILU(0) for structured matrices

Let A be some symmetric matrix with a regular 5-point stencil structure defined as

$$A = D_0 - E - F,$$

where $-E$ is the strict lower part of A, $-F$ is the strict upper part of A, and D_0 is the diagonal of A. Then the ILU(0) factorization may be written in the form

$$LU = (D - E)D^{-1}(D - F),$$

where D is some diagonal matrix, which is generally different from D_0. In this case, finding the factorization reduces to finding the entries of D. This can be obtained via a simple recursive formula that corresponds to an ILU elimination process to update the diagonal. See Figure 2.1 for an illustration.

At the i-th step of the recursion, the diagonal entry, d_i, is computed as follows:

$$d_i = d_{0_i} - \mathbf{e}_i^T D_{i-1}^{-1} \mathbf{f}_i, \tag{2.1}$$

where D_{i-1} is an $(i-1) \times (i-1)$ matrix corresponding to the diagonal of U computed thus far (during the factorization), $\mathbf{e}_i$ is the i-th row of E, transposed as a column vector, $\mathbf{f}_i$ is the i-th column of F, and d_{0_i} represents the i-th entry of the diagonal of the original matrix A. Equation 2.1 may be written more succinctly by expanding the term $\mathbf{e}_i^T D_{i-1}^{-1} \mathbf{f}_i$. To see this, consider the stencil of a standard 5-point difference

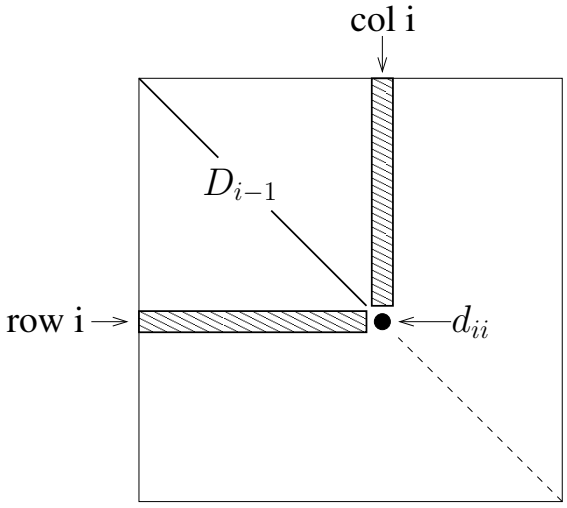


Figure 2.1: The i-th step of the recursion.

approximation to a second-order partial differential operator, discretized on a structured grid with homogeneous Dirichlet boundary conditions. This is given in Figure 2.2. After the ILU(0) factorization, the non-zero entries in the L and U factors lie on the stencil points corresponding to the upper and lower parts of the original stencil, as shown in Figures 2.3(a) and 2.3(b) respectively.

Multiplying the stencils corresponding to L and U together gives the stencil shown in Figure 2.4. Note that the diagonal entries of the product LU are the same as the diagonal entries of the original matrix. This leads to the following expression for a diagonal entry in D:

$$d_i = d_{0_i} - l_{i,i-1}u_{i-1,i} - l_{i,i-m}u_{i-m,i}. \tag{2.2}$$

Equating the terms in the stencil of LU to those of the original matrix, we have that

$$l_{i,i-1} = \frac{e_{i,i-1}}{d_{i-1}} \quad \text{and} \quad l_{i,i-m} = \frac{e_{i,i-m}}{d_{i-m}},$$

and

$$u_{i-1,i} = f_{i-1,i} \quad \text{and} \quad u_{i-m,i} = f_{i-m,i}.$$

Substituting these into Equation 2.2 gives the following recurrence relation:

$$d_i = d_{0_i} - \frac{e_{i,i-1}f_{i-1,i}}{d_{i-1}} - \frac{e_{i,i-m}f_{i-m,i}}{d_{i-m}}. \tag{2.3}$$

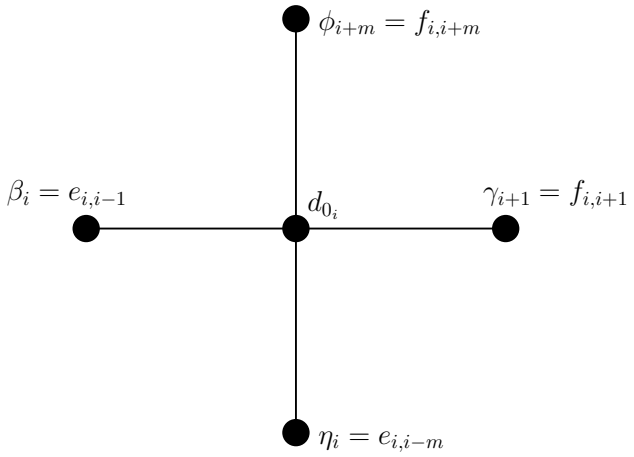
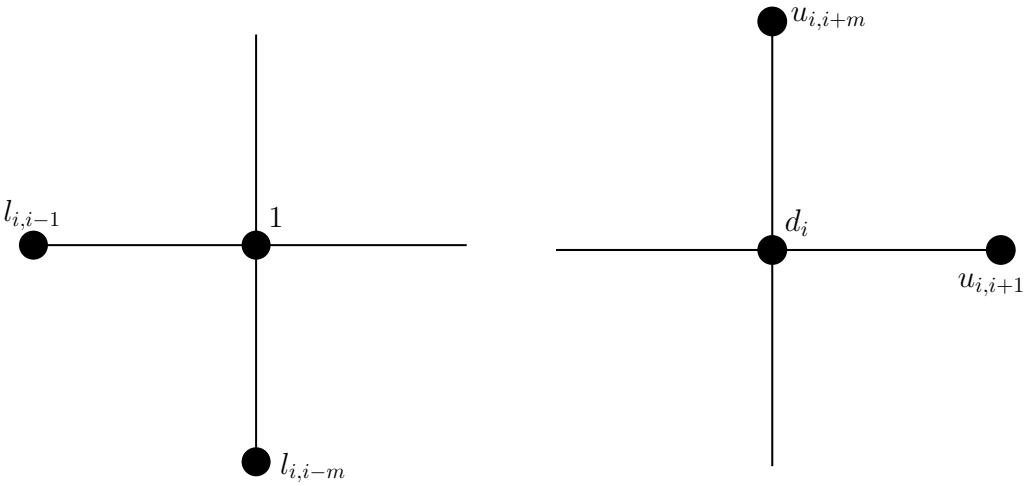


Figure 2.2: Stencil of the original matrix.



(a) Stencil of L

(b) Stencil of U

Figure 2.3: Stencil of the L and U factors from the ILU(0) factorization

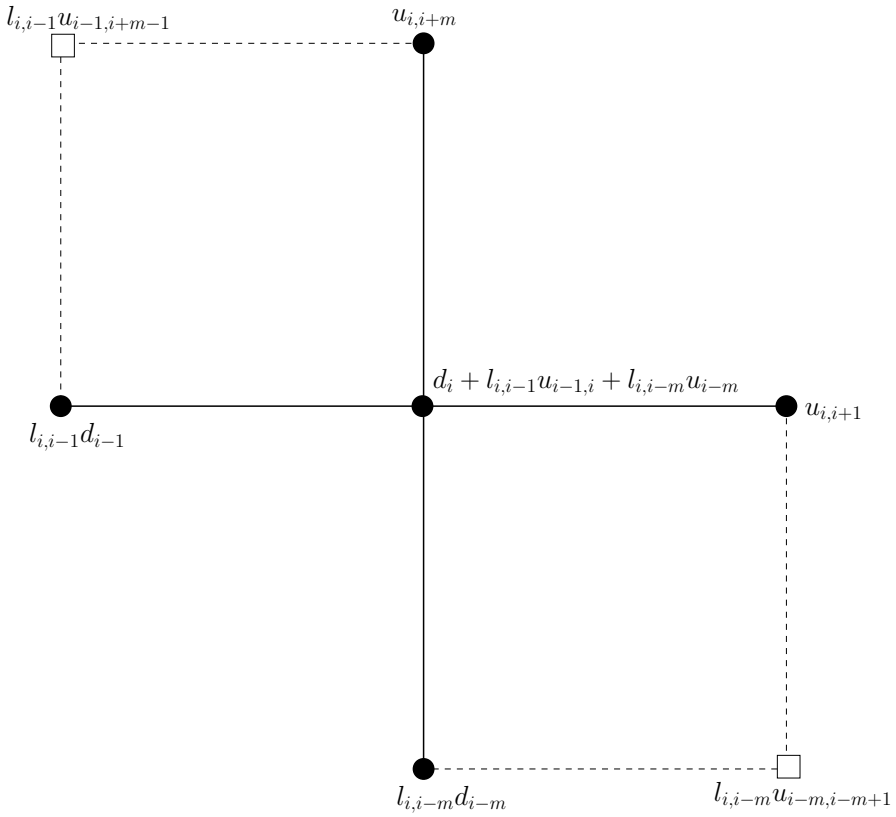


Figure 2.4: Stencil of the product LU

The error in the product LU as an approximation to A, is characterized by two extra off-diagonal bands that correspond to stencil indices that do not coincide with the stencil of the original matrix. In other words, we have that $R = LU - A$, where R is the error matrix with two off-diagonal (one upper and one lower) bands of entries. For each row, the sum of the error entries may be expressed as

$$s_i = \frac{e_{i,i-1}f_{i-1,i+m-1}}{d_{i-1}} + \frac{e_{i,i-m}f_{i-m,i-m+1}}{d_{i-m}}. \tag{2.4}$$

The above details, and more on the recurrence, may be found in [10, 69].

Note that when $i < m$, the term $\frac{e_{i,i-m}f_{i-m,i}}{d_{i-m}}$ drops out of the recurrence relation. This means there is indeed more than one sequence appearing in the recurrence relation, as a result of influences from the boundary nodes on the interior nodes. To simplify the analysis, let us ignore the boundary effects by assuming we have an infinite grid for the

domain. As such, for each node, the diagonal entry in D can be expressed exactly as in Equation 2.3. This recurrence relation has a limit, which can be obtained by considering the following limit expression for Equation 2.3:

$$\begin{aligned} d^* &= d_{0_i} - \frac{(e_{i,i-1}f_{i-1,i} + e_{i,i-m}f_{i-m,i})}{d^*} \\ &= d_{0_i} - \frac{\mathbf{e}_i^T \mathbf{f}_i}{d^*} \\ &= d_{0_i} - \frac{\Sigma}{d^*}, \end{aligned} \tag{2.5}$$

where $\Sigma = \mathbf{e}_i^T \mathbf{f}_i$ is the dot product of the i-th row of E, and the i-th column of F. We thus obtain the limit d^* as:

$$d^* = \frac{d_{0_i} + \sqrt{d_{0_i}^2 - 4\Sigma}}{2}. \tag{2.6}$$

From Equation 2.6, it is clear that this limit is well-defined when $d_{0_i}^2 - 4\Sigma \geq 0$. In other words, the recurrence relation converges if $d_{0_i} \geq 2\sqrt{\Sigma}$. Note that this is under the assumption that d_{0_i} is the same for all the nodes (at least in the interior) of the domain.

Now, assume that the original stencil is perturbed by some $\alpha > 0$, prior to performing the ILU factorization. Then Proposition 2.1.1 suggests a lower bound for α, necessary to obtain a stable factorization.

Proposition 2.1.1. *The recurrence relation for the perturbed system converges to a well-defined limit if*

$$\alpha \geq 2\sqrt{\Sigma} - d_{0_i}. \tag{2.7}$$

Proof. The perturbed form of Equation 2.3 takes the form:

$$d_i = (d_{0_i} + \alpha) - \frac{e_{i,i-1}f_{i-1,i}}{d_{i-1}} - \frac{e_{i,i-m}f_{i-m,i}}{d_{i-m}}, \tag{2.8}$$

and as in Equation 2.5, we obtain the limit d^* as

$$d^* = \frac{(d_{0_i} + \alpha) + \sqrt{(d_{0_i} + \alpha)^2 - 4\Sigma}}{2}. \tag{2.9}$$

This limit is well-defined when $(d_{0_i} + \alpha)^2 - 4\Sigma \geq 0$, and the result follows by solving for α. □

The error matrix R, of the perturbed system now has additional entries on the main diagonal corresponding to α. The resulting sum of the error for each row takes the form:

$$s_i = \alpha + \frac{e_{i,i-1}f_{i-1,i+m-1}}{d_{i-1}} + \frac{e_{i,i-m}f_{i-m,i-m+1}}{d_{i-m}},$$

which in the limit, gives

$$\begin{aligned} s_i &= \alpha + \frac{(e_{i,i-1}f_{i-1,i+m-1} + e_{i,i-m}f_{i-m,i-m+1})}{d^*} \\ &= \alpha + \frac{\hat{\Sigma}}{d^*}, \end{aligned} \tag{2.10}$$

where $\hat{\Sigma} = \sum_{k \neq i} \mathbf{e}_i^T \mathbf{f}_k$ is the sum of the dot product of the i-th row of E, and the k-th column of F, so that the entry $a_{i,k} = 0$ in the original matrix A. In practice, since the original matrix is sparse, only a few of these dot products need to be computed. For structured matrices, such as ones under consideration here, the location of the errors are known (as shown in Figure 2.4), so $\hat{\Sigma}$ can be computed in a simplified way. Substituting d^* into the above equation gives:

$$s_i = \alpha + \frac{2\hat{\Sigma}}{(d_{0_i} + \alpha) + \sqrt{(d_{0_i} + \alpha)^2 - 4\Sigma}}. \tag{2.11}$$

To this point, we have assumed that the components of the sum of the error s_i, are all positive as a result of the problem under consideration. In this case, an important observation is that at the limit, $s_i = \|A - LU\|_\infty$. Thus, it is easy to see from Equation 2.11 that for small α, the infinity norm of the error is dominated by the fraction on the right hand side of the equation, whereas for large α, the infinity norm of the error grows like α.

Rationalizing s_i to get rid of the square root in the denominator and minimizing the result by taking the derivative with respect to α gives:

$$\begin{aligned} \alpha &= 2\left[\left(\frac{2\Sigma + \hat{\Sigma}}{\hat{\Sigma}}\right)^2 - 1\right]^{-\frac{1}{2}} \left[\frac{2\Sigma + \hat{\Sigma}}{\hat{\Sigma}}\right]\sqrt{\Sigma} - d_{0_i} \\ &= \frac{2H\sqrt{\Sigma}}{\sqrt{H^2 - 1}} - d_{0_i} = \alpha^*, \text{ with } H = \frac{2\Sigma + \hat{\Sigma}}{\hat{\Sigma}}. \end{aligned} \tag{2.12}$$

Since this value for α is bigger than the lower bound (in Equation 2.12), it is an appropriate choice as it also minimizes the infinity norm of the error. More generally, we

could take α^* as an upper bound, so that we obtain a range of values for α as follows:

$$2\sqrt{\Sigma} - d_{0_i} \leq \alpha \leq \frac{2H\sqrt{\Sigma}}{\sqrt{H^2-1}} - d_{0_i}. \tag{2.13}$$

Note that here, the lower bound is what needs to be satisfied, since it directly impacts the stability of the factorization. Defining the upper bound with respect to some norm of the error in the factorization seems appropriate, since that helps control the accuracy of the factorization by constraining the preconditioner $M = LU$, to be close enough to the original matrix A. Here, we have used the infinity norm of the error to satisfy this criterion, as it comes up naturally in the error sum, and makes for a simpler analysis.

Note that if $\alpha^* < 0$, then no shifting is necessary since the upper bound is satisfied when α is negative. Recall that we have assumed α to be positive, since a negative value of α will make the perturbed matrix less diagonally dominant, which can affect the stability of the factorization. It makes sense, therefore, to choose α to be zero in this case. This helps to safeguard the accuracy of the factorization by ensuring that the perturbed system matrix is not too far from the original matrix. This is rather unfortunate, as it suggests that we cannot do better with a real shift. However, defining the shift as $\sqrt{\alpha} = i\sqrt{|\alpha|}$ provides an interesting alternative.

More generally, we can select $|\alpha|$ in the range

$$\left| \frac{2H\sqrt{\Sigma}}{\sqrt{H^2-1}} - d_{0_i} \right| \leq |\alpha| \leq |2\sqrt{\Sigma} - d_{0_i}|, \tag{2.14}$$

and define $B = A \pm i\sqrt{|\alpha|}I$ as the perturbed operator to be used to construct the ILU preconditioner. This way of selecting the shift means we now have to precondition a real-valued system with a complex-valued matrix. While the spectrum of the resulting preconditioned matrix can be appealing (see section on imaginary perturbations below), the convergence of GMRES can be slow as a result of the complex arithmetic and the presence of imaginary terms in the residual. One way to get around this issue is to extract a real preconditioner from the resulting *complex-valued* preconditioner.

Suppose that we define a preconditioner based on the ILU factorization of A, so that $M_A = LU = A + R$, where R is the residual from the ILU factorization, and L and U are the incomplete LU factors of the original *real* matrix A. Now consider a similar

decomposition for $B = A + i\sqrt{|\alpha|}I$ in complex arithmetic. We have:

$$\begin{aligned} M_B = \tilde{L}\tilde{U} &= B + \tilde{R} \\ &= A + i\sqrt{|\alpha|}I + \tilde{R}, \end{aligned} \tag{2.15}$$

where $\tilde{L}$, $\tilde{U}$ and $\tilde{R}$ are the incomplete LU factors of B and the corresponding residual respectively. We wish to extract a preconditioner M, from M_B, so that M is close enough to A. A simple solution is to choose the preconditioner M so that $M^{-1} = Re[M_B^{-1}] = Re[(\tilde{L}\tilde{U})^{-1}]$, where $Re(Q)$ means taking the real part of Q. This is equivalent to taking the real part of the result of the preconditioner solve involving M_B^{-1}, during the solution of the linear system with the preconditioned GMRES solver. However, this approach does not work very well in practice, as the imaginary contributions perturb the real part of the inverse, and ignoring them can result in a poor solution. Another option is to take $M = Re(\tilde{L}\tilde{U})$. This is a reasonable option, since in exact arithmetic, extracting the real part of B should give back A. Note however, that we are only interested in an approximation to A. From Equation 2.16, we have that

$$Re(\tilde{L}\tilde{U}) = A + Re(\tilde{R}).$$

However, replacing α with $i\alpha$ in Equation 2.11 reveals that $\|Re(\tilde{R})\| < \|R\|$. This implies that $M = Re(\tilde{L}\tilde{U})$ should be a more accurate preconditioner than M_A.

Expanding M_B as

$$M_B = \tilde{L}\tilde{U} = \tilde{L}_r\tilde{U}_r - \tilde{L}_i\tilde{U}_i + i(\tilde{L}_r\tilde{U}_i + \tilde{L}_i\tilde{U}_r),$$

where the subscripts r and i represent the real and imaginary parts (of $\tilde{L}$ and $\tilde{U}$) respectively, we obtain $Re(\tilde{L}\tilde{U}) = \tilde{L}_r\tilde{U}_r - \tilde{L}_i\tilde{U}_i$. Although the real and imaginary parts of $\tilde{L}$ and $\tilde{U}$ are available, their products are not. Explicitly forming the matrix $\tilde{L}_r\tilde{U}_r - \tilde{L}_i\tilde{U}_i$ requires two sparse matrix-matrix multiplies. Once this is done, the preconditioner solve can be performed with a few iterations of a Krylov method, such as GMRES, preconditioned with $\tilde{L}_r\tilde{U}_r$. Note here that in order to avoid doing the matrix-matrix products, one would have to somehow approximate the matrix $\tilde{L}_r\tilde{U}_r - \tilde{L}_i\tilde{U}_i$. For instance, one could take M to be $\tilde{L}_r\tilde{U}_r$. However, the resulting preconditioner, M, will be a poorer approximation to A. Alternatively, one could approximate M^{-1} by writing $M^{-1} = [I - (\tilde{L}_r\tilde{U}_r)^{-1}\tilde{L}_i\tilde{U}_i]^{-1}(\tilde{L}_r\tilde{U}_r)^{-1}$, and approximating $[I - (\tilde{L}_r\tilde{U}_r)^{-1}\tilde{L}_i\tilde{U}_i]^{-1}$ by a

Maclaurin series expansion. This is promising, as the terms involved in the expansion are all available from $\tilde{L}$ and $\tilde{U}$. However, it can be quite expensive as it involves performing a series of matrix-matrix multiplies. Furthermore, it requires that the spectral radius of $(\tilde{L}_r\tilde{U}_r)^{-1}\tilde{L}_i\tilde{U}_i$ be less than 1.0 in order to guarantee a good approximation, and this is hard to enforce.

For a symmetric real matrix A, we can explore another option that does not require extracting a real preconditioner from M_B. This approach is based on solving the normal equations $A^TAx = A^Tb$, instead of the original problem $Ax = b$. The preconditioner M, can then be obtained from the incomplete factorization of the shifted matrix

$$\begin{aligned} G &= A^TA + |\alpha|I \\ &= (A^T - i\sqrt{|\alpha|}I)(A + i\sqrt{|\alpha|}I) \\ &= B^HB, \end{aligned} \tag{2.16}$$

where the superscript H represents the Hermitian transpose. Notice here that now the preconditioner is real, and a real, positive shift is applied. Once again, we can either choose $|\alpha| = |\alpha^*|$, or choose α as in Equation 2.13. Here, we have assumed a fixed (or constant) α for all the rows of the matrix. However, based on the definition of α, it is clear that it can be chosen locally with respect to each row.

2.1.2 Numerical Results

In what follows, we present some numerical results for shifted ILU(0) on some examples from the shifted Laplace problem described earlier in Section 1.4.3. These examples are run in matlab, and for each test case, we invoke matlab's `LUINC` function with zero fill-in elements, to generate the ILU(0) factorization. We use restarted (F)GMRES with a restart dimension of 100, and allow a maximum of 500 iterations. We assume convergence of the iterative procedure whenever the initial residual is reduced by a factor of 1.0×10^{-6}. The right hand side for the linear system is artificially constructed by assuming a solution of all ones.

In the first example, we present results for the shifted Laplace problem with a shift of $\varrho = -0.25$, discretized on a grid of size 50×50, 100×100, and 200×200. The resulting system matrix is real, symmetric, and indefinite. For this example, the bounds for the

range of α in Equation 2.13 can be easily evaluated, giving

$$\begin{aligned} 2\sqrt{2} - 3.75 \leq \quad \alpha \quad &\leq \frac{6\sqrt{2}}{\sqrt{8}} - 3.75 \\ -0.9216 \leq \quad \alpha \quad &\leq -0.75 \end{aligned} \tag{2.17}$$

The choice of α is negative, and the matrix is diagonally dominant. We compare the performance of standard (unshifted) ILU(0) against that of shifted ILU(0) using an imaginary shift on the original matrix A, and that of shifted ILU(0), using a real shift on $A^T A$. For the test with the imaginary shift, we compute the ILU(0) factorization of the shifted matrix, and approximate the real preconditioner by performing 5 iterations of GMRES on the matrix $\tilde{L}_r\tilde{U}_r - \tilde{L}_i\tilde{U}_i$, preconditioned by $\tilde{L}_r\tilde{U}_r$ as discussed above. We use a global shift by assuming a constant shift for all the rows of the matrix. For each of the shifted schemes, we choose the shift corresponding to the upper bound of their range for selecting the shift. In other words, for the imaginary shifted preconditioner, we select the shift as $i\sqrt{|\alpha|} = i\sqrt{0.9216}$ (from Equation 2.14) and for the real shifted approach (using $A^T A$), we select the shift as $|\alpha| = |\alpha^*| = 0.75$ (from Equation 2.13). Note that for the imaginary shift, choosing the shift as $\pm i\sqrt{0.9216}$ gives the same result. Tables 2.1, 2.2, and 2.3 give the results. For the larger problems, using the unshifted preconditioner fails to converge within the allowable number of iterations. This is not surprising, as the problem gets harder with the increase in size. Using the imaginary shift approach appears to perform relatively well for the small problem. However, it also fails to converge when the problem gets harder with increase in size, although the residual after 500 iterations is smaller than that of the unshifted approach. This failure to converge is also understandable, since by extracting an approximate real preconditioner from the complex preconditioner, the resulting preconditioning operation is equivalent to preconditioning with a *twice* approximated preconditioner, which may not be accurate enough. Using more GMRES iterations for the preconditioner solve can improve the overall convergence. However, this can make the overall scheme expensive. Furthermore, the result of the preconditioning operation depends on the preconditioner $\tilde{L}_r\tilde{U}_r$, used to precondition the operation. Transforming the linear system into the normal equations form, and using a preconditioner derived from a real shift to the normal matrix, successfully solves all the problems.

In the next example, we illustrate the effect of shifting on ILU(0) by visualizing the

ILU(0) method	iterations	time
Unshifted	170	1.77s
Imaginary shift on A	112	2.45s
Real shift on A^TA	100	1.82s

Table 2.1: Results for unshifted and shifted ILU(0) on a shifted Laplace problem on a 50×50 grid

ILU(0) method	iterations	time
Unshifted	$*500*$	–
Imaginary shift on A	$*500*$	–
Real shift on A^TA	158	16.47s

Table 2.2: Results for unshifted and shifted ILU(0) on a shifted Laplace problem on a 100×100 grid

ILU(0) method	iterations	time
Unshifted	$*500*$	–
Imaginary shift on A	$*500*$	–
Real shift on A^TA	198	219.86s

Table 2.3: Results for unshifted and shifted ILU(0) on a shifted Laplace problem on a 200×200 grid

result of the recurrence relation. The problem is the shifted Laplace example, shifted by $\varrho = -0.5$ and discretized on a 50×50 grid. While this choice of ϱ makes the problem quite challenging, the range of α is still negative for this problem. Hence, we look instead at the result of the recurrence relation for the normal equations formulation, as it allows for the selection of a real α. Figures 2.5 and 2.6 show the resulting diagonal entries from the recurrence relation for the unshifted and the shifted ILU(0) factorizations respectively. The figures show a section of the interior grid points on the horizontal axis, and the values of the resulting diagonal entries associated with these grid points, on the vertical axis. Figure 2.5 shows large fluctuations in the diagonal entries of the resulting factorization. This yields a preconditioner that is very unstable with $||(LU)^{-1}\mathbf{1}||_2 \approx 2.69 \times 10^{18}$, rendering it useless as a preconditioner for any iterative method. Shifting the matrix by $|\alpha| = 0.65$, say, which lies in the preferred range, yields the result in Figure 2.6. Here, we observe that the diagonal entries in the interior converge to a limiting value ≈ 9.1. There are no extreme fluctuations in the diagonal entries as observed for the unshifted scheme, and the values are close to each other. The resulting factorization is stable, with $||(LU)^{-1}\mathbf{1}||_2 \approx 416.70$, and the solution to the linear system converges after 136 iterations.

While the ideas discussed in this section may be generalized to the ILU(k) factorization for regularly structured matrices, it is not generally applicable to the threshold-based ILUT factorization. This is because the dropping strategy for the ILUT factorization is algebraic in nature, and not based on the structure of the matrix. As such, it is not feasible to define α in the way described in this section. The next section presents new ideas for a shifted ILUT factorization, using imaginary perturbations for complex-valued linear systems. More details on this work may be found in [71].

2.2 Imaginary Perturbations

2.2.1 Motivation

Indefinite linear systems generally require a more accurate factorization or complicated preconditioning approach, such as those in [63, 72, 73]. However, simply improving the accuracy of the factorization (by decreasing the drop tolerance) can yield factors that are even more unstable than the original system matrix, making the factorization ineffective

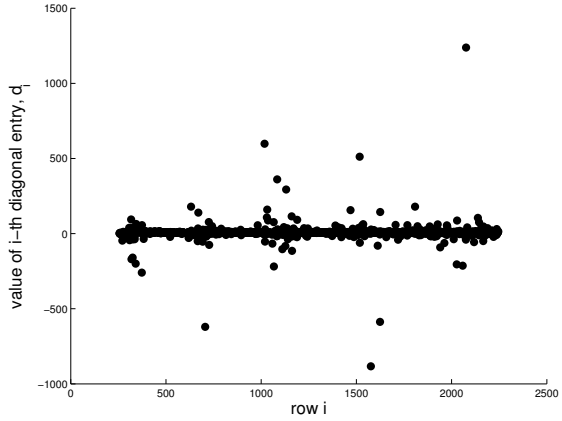


Figure 2.5: Diagonal of unshifted ILU(0). Convergence profile of diagonal entries with respect to the (interior) grid points.

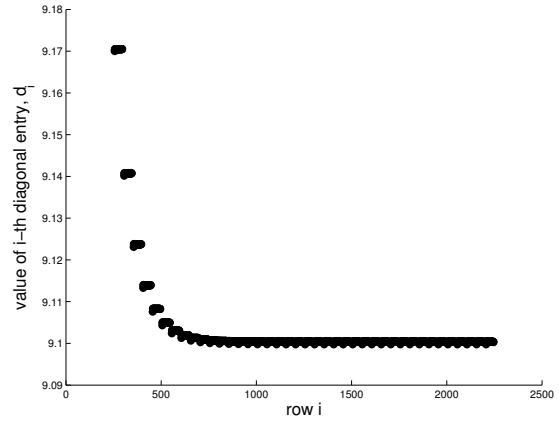


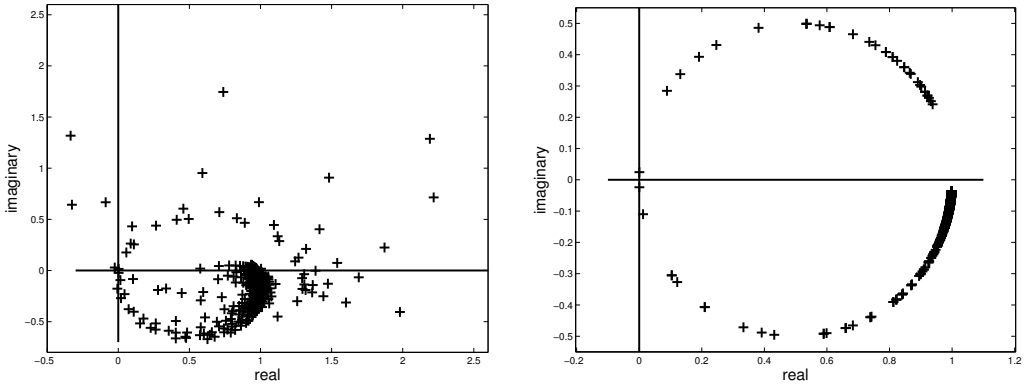
Figure 2.6: Diagonal of shifted ILU(0). Convergence profile of diagonal entries with respect to the (interior) grid points.

as a preconditioner. While the above strategy in Section 2.1 offers an improvement on standard ILU(k), even for some very poorly conditioned systems, performance can be poor when the original system matrix is very indefinite. For indefinite systems, real-valued perturbations to the diagonal tend to be ineffective as they only shift the spectrum along the x-axis, and could possibly move eigenvalues close to zero. As such, a few of the successful methods that use diagonal perturbation strategies on indefinite systems, have relied on the use of complex (or imaginary) perturbations instead [19, 63, 71–75].

To demonstrate the effect of complex shifts, we consider a simple example programmed in matlab. The example follows the model problem described in Section 1.4.3. We take the finite-difference discretization of the Laplace operator, $-\Delta$, on a 25×25 grid. The discretization assumes a scaling based on the mesh size parameter, h, so that the resulting matrix initially has 4 on the diagonal, and four off-diagonal entries of -1 each. The matrix is then shifted by adding a negative shift of -1.0 to the diagonal, to make it indefinite. Note that (because of the scaling by h^2), this shift is quite severe and makes it a challenge to iterative methods to solve a linear system involving this matrix. The resulting matrix A, has size $n = 625$, with 49 negative eigenvalues (smallest is $\lambda_1 = -0.9708$ and largest is $\lambda_{625} = 6.9708$). First, we perform an ILU factorization of A, calling the `LUINC` matlab function without pivoting and using a drop tolerance of 0.1. This yielded a preconditioner that is unstable. The condition number of the matrix LU was calculated to be $\kappa(LU) \approx 4.8e + 19$. This preconditioner is useless for any iterative method.

Next, we perturb the matrix by adding the shift $\alpha i \equiv 0.25i$ to the diagonal of A, and then compute the resulting ILU preconditioner of the shifted matrix $B = A + \alpha i I_n$, under the same conditions as above. As expected the condition number of LU is now better, and we find that $\kappa(LU) \approx 1.1e + 03$. The resulting approximate L, U factors of this matrix B are then used to precondition *the original matrix* A, so we plot the eigenvalues of $L^{-1}AU^{-1}$ where L, U are the incomplete LU factors of B. The result is shown in Figure 2.7(a). The new preconditioner now appears remarkably good. It yields a good clustering around one and few eigenvalues with negative real parts and none close to zero. The interesting observation here is that by making the problem complex, somewhat artificially, we avoid the very serious instability we initially encountered with

the ILU factorization on the original (real) matrix.



(a) Incomplete LU factors of B (b) Exact LU factors of B

Figure 2.7: Eigenvalues of $L^{-1}AU^{-1}$, L and U factors of $B = A + 0.25iI_n$

More generally, let A be a Hermitian matrix, and define

$$B = A + \theta I_n,$$

where $\theta = \nu + i\alpha$ for some real numbers $\nu, \alpha \geq 0$. Suppose that we use the exact LU factorization of B as the preconditioner, $M = LU(B)$. Then, the eigenvalues of the preconditioned matrix $M^{-1}A$ satisfy

$$\mu_j = \frac{\lambda_j}{(\lambda_j + \nu) + i\alpha} = \frac{\lambda_j(\lambda_j + \nu)}{(\lambda_j + \nu)^2 + \alpha^2} - i\frac{\lambda_j \alpha}{(\lambda_j + \nu)^2 + \alpha^2}, \tag{2.18}$$

where the λ_j's are the eigenvalues of A, and the μ_j's are the eigenvalues of the preconditioned matrix, $M^{-1}A$. From the above equation and the identity $\lambda_j = \frac{1}{2}(\lambda_j + \nu + i\alpha + \lambda_j - \nu - i\alpha)$, we obtain

$$\left|\mu_j - \frac{1}{2}\right| = \frac{1}{2}\frac{|(\lambda_j - \nu) - i\alpha)|}{|(\lambda_j + \nu) + i\alpha)|}. \tag{2.19}$$

First, we observe from Equation (2.18) that a (real) positive eigenvalue, λ_j, of A will be transformed into a complex eigenvalue with a negative imaginary part, while a negative λ_j yields μ_j with a positive imaginary part. Next, we observe from Equation (2.19) that, for $\nu = 0$, all eigenvalues, μ_j, of the preconditioned system lie on the

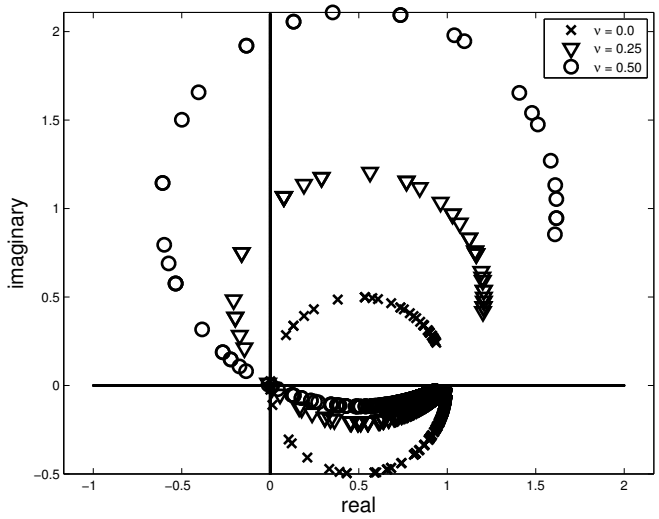


Figure 2.8: Spectrum of the preconditioned matrix, using exact L, U factors of $B = A + \theta I_n$, where $\theta = \nu + 0.25i$.

circle centered at $1/2 + 0i$ with radius $1/2$. All the eigenvalues are on the right side of the complex plane, and the large eigenvalues are mapped near the point 1.0 by this transformation. An illustration of this result, using the matlab example above, is shown in Figure 2.7(b).

From Equation (2.19), it is clear that for $\nu > 0$, a (real) positive λ_j will be mapped to a complex number within a distance of $1/2$ from the center of this circle; whereas a negative λ_j will be mapped to a complex number beyond a distance of $1/2$ from the center. Figure 2.8 shows the spectrum of the preconditioned matrix, using the exact LU factors of B from the above matlab example. The figure shows plots of the preconditioned spectrum with $\theta = \nu + 0.25i$, for different values of ν. We observe a better clustering of the eigenvalues for smaller values of ν. Furthermore, from Equation (2.18), as α tends to zero, the eigenvalues, μ_j, tend to the real values $\lambda_j/(\lambda_j + \nu)$. These values, in turn, tend towards 1.0 as $\nu \to 0$, as expected in the case when $B = A$. This clustering benefits Krylov accelerators used in the solution of the linear system involving this matrix. We note here that the resulting preconditioner based on $A + \theta I_n$ is both indefinite and non-Hermitian; hence, this precludes the use of the Krylov accelerator MINRES to solve the system, and GMRES or BiCGStab must be used instead. As a result of the

benefit of the clustering on the Krylov accelerator, it is not altogether surprising that most of the prior work has relied on the use of purely imaginary perturbations.

The above trivial matlab example uses a constant (*imaginary*) shift ($\alpha i \equiv 0.25i$), and there was no specific effort made in its selection. Finding the optimal value of the shift, α, is not trivial, and much of the work in the literature has chosen α based on some design criterion. For instance, in [72], Van Gijzen et. al. proposed a "quasi" optimal value for the shift, as the value that minimized the upper bound of the GMRES residual norm. However, this optimal shift is derived with the assumption that an accurate (or exact) factorization for the preconditioner is performed, which may not be the case in practice. In [19], Made et. al. base their choice on the discretization of the problem, by relating the shift to the mesh size parameter. In general, a relatively large α improves the diagonal dominance of the matrix B, which could lead to better stability of the L and U factors obtained from the ILU factorization of B. However, the resulting preconditioner, $M = LU$, may be a poor approximation to the original matrix A, making it an ineffective preconditioner. Thus, a good criterion for selecting α is to achieve a compromise between making α as large as possible to improve stability and making it not too large to maintain the accuracy of the ILU factorization.

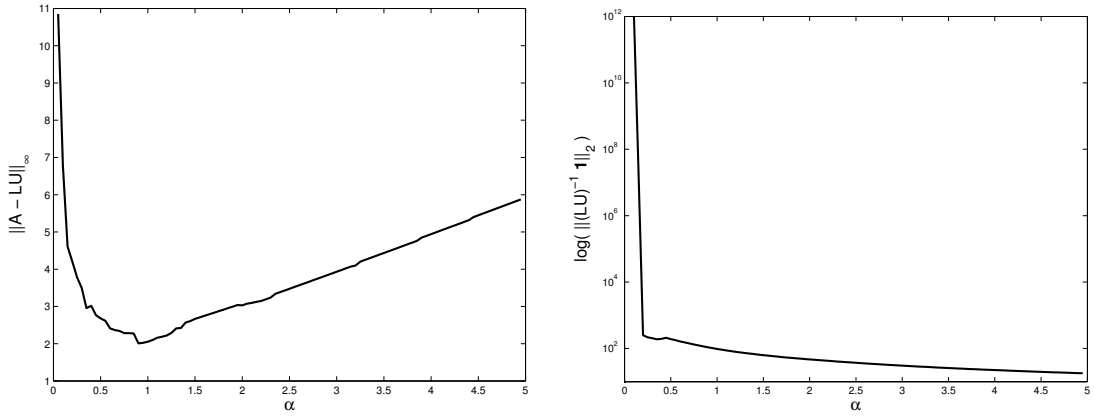
2.2.2 Some sensitivity analysis

In order to better understand the effect of shifting on the factorization, and the quality of the resulting preconditioner, it is necessary to understand how it varies with certain quantities that determine the quality of the factorization.

Consider a small problem in acoustic scattering on a bounded obstacle governed by the Helmholtz equation, with wavenumber $k = 8\pi$. The resulting system matrix has size $n = 7380$, with 63900 non-zero entries. The system is solved using a preconditioned FGMRES solver [11], preconditioned with a shifted ILUT factorization (row version) with drop tolerance $\tau = 0.04$ and the `lfil` parameter set to be large ($\approx n$). The matrix used for constructing the preconditioner is obtained by shifting the diagonal of each row of the original matrix by a constant imaginary shift, αi, and we solve the linear system many times over varying α. Here, the right hand side of the linear system is artificially constructed by assuming a solution of all ones.

Figure 2.9 shows how α varies with $||A - LU||_\infty$ and the log of $||(LU)^{-1}\mathbf{1}||_2$, where

1 is a vector of all ones. The quantity, $||A - LU||_\infty$, is introduced as a measure of the accuracy of LU as an approximation to A. We also introduce the quantity $||(LU)^{-1}\mathbf{1}||_2$, which represents a lower bound estimate of the conditioning of the inverse LU factors, as a measure of the stability of the factorization. An unstable factorization produces inverse factors that can be very large in norm, which renders the preconditioner useless. We see from figure 2.9(a) that the quantity $||A-LU||_\infty$ decreases rapidly to a minimum and then starts to increase again, with increasing α. Increasing α causes more terms to be dropped during the factorization, which may result in LU factors that poorly approximate A. In figure 2.9(b), we see that the stability of the LU factorization generally improves with increasing α. As α gets larger, the matrix B becomes more diagonally dominant, which yields a more stable factorization.



(a) α vs. $||A - LU||_\infty$ (b) α vs. $log(||(LU)^{-1}\mathbf{1}||_2)$

Figure 2.9: An analysis of the effect of shifting on $||A - LU||_\infty$ (left); and $||(LU)^{-1}\mathbf{1}||_2$ (right)

A good choice for α, therefore, is one which balances the accuracy and stability of the factorization. Figure 2.10(a) shows the effect of shifting on the iteration count for convergence. The figure shows that the iteration count decreases as the quality of the factorization improves with increasing α to a minimum (61 iterations, with $\alpha = 0.26$), and increases again after some optimal α. Here ILUT applied to the original matrix A failed to converge. In similar tests (not shown here), we reduced the drop

tolerance parameter and observed a flatter profile for the minimum number of iterations, corresponding to more than one value for the optimal shift. This suggests that the choice of α is sensitive to the drop tolerance. We shall exploit this later in designing a scheme for choosing the shift.

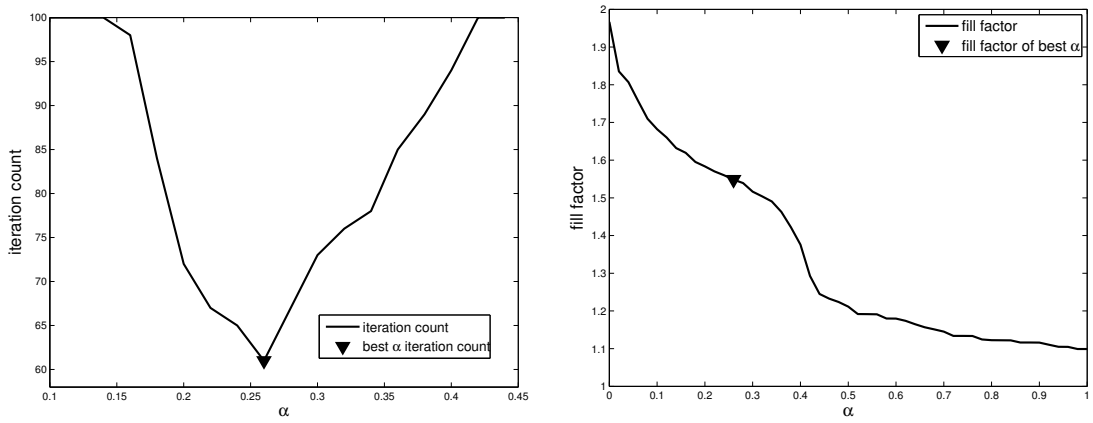
To measure the cost of the storage and computation of matrix-vector products with the resulting ILUT preconditioner, we define the fill-factor (or memory usage) as $c_F = (nnz_L + nnz_U - n)/nnz$, where n is the dimension of the problem, nnz is the number of nonzeros of the original matrix, and nnz_L and nnz_U are the number of nonzeros in the L and U matrices, respectively. Since the unit diagonal of L is not stored, it is compensated for by subtracting n in the equation. Figure 2.10(b) shows the effect of shifting on the fill-factor of the resulting factorization. We observe that the fill-factor generally decreases with increasing α. This is because a large shift could result in small entries that are dropped during the elimination of a particular row of the factorization. As such, the factored row will contain fewer fill-in elements in the L and U parts. The figure shows that performing ILUT on the shifted matrix B, using the optimal (or best) α value ($\alpha = 0.26$) as the shift, results in a preconditioner that is about 25% cheaper in terms of memory costs, than the preconditioner derived from the original matrix.

Thus, according to the above results, modifying the diagonal of the original matrix by adding a small imaginary perturbation could improve the stability and accuracy of the resulting factorization, with the added benefit of reducing memory costs.

2.2.3 Choosing the imaginary shift

In what follows, we present two different algebraic strategies to select the imaginary shift. These strategies are based entirely on the premise that we wish to maximize the diagonal elements of the matrix to improve stability, without compromising the accuracy of the resulting preconditioner. Note that in these techniques, the shift is chosen locally with respect to each row of the matrix A, and hence is not constant (or global) as in the above examples in Sections 2.2.1 and 2.2.2.

Let $a_{kk} = \eta + \beta i$ denote the diagonal entry of the k-th row of the original matrix A. The perturbation strategy follows by perturbing this diagonal term by αi, so that its



(a) α vs. iteration count

(b) α vs. fill-factor

Figure 2.10: An analysis of the effect of shifting on the iteration count for convergence (left); and the fill-factor (right)

squared modulus is increased by (at least) the quantity γ^2, say. We therefore write:

$$\begin{aligned} |\eta + i(\beta + \alpha)|^2 &\geq |\eta + i\beta|^2 + \gamma^2 \\ \eta^2 + \beta^2 + 2\alpha\beta + \alpha^2 &\geq \eta^2 + \beta^2 + \gamma^2 \\ \alpha^2 + 2\alpha\beta - \gamma^2 &\geq 0 \end{aligned}$$

This is satisfied when $\alpha \geq -\beta + \sqrt{\beta^2 + \gamma^2}$ or $\alpha \leq -\beta - \sqrt{\beta^2 + \gamma^2}$. Increasing the square of the modulus of a_{kk} by γ^2 with the smallest perturbation is achieved by the following choice for α:

$$\alpha = \begin{cases} -\beta + \sqrt{\beta^2 + \gamma^2} & \text{if} \quad \beta \geq 0 \\ \\ -\beta - \sqrt{\beta^2 + \gamma^2} & \text{if} \quad \beta < 0 \end{cases} \tag{2.20}$$

In other words, $\alpha = \text{sign}(\beta) \times \left[\sqrt{\beta^2 + \gamma^2} - |\beta|\right]$. The question that now remains is how to select the parameter γ. We propose two different schemes for selecting γ. The first is based on improving stability by improving the diagonal dominance of the rows of the matrix A. From Figure 2.9(a), we observe that even after reaching its minimum, the quantity $||A - LU||_\infty$ only grows slowly with increasing α. Moreover, only a small

perturbation is required to reach the minimum. Thus by focusing on improving stability by improving diagonal dominance, we can obtain a factorization that is potentially more stable, and close to A in norm. Note that α need not be too large (for stability), as only a small shift is necessary to significantly improve stability, as depicted in figure 2.9(b).

Recall that row k is (strictly) diagonally dominant if

$$|a_{kk}| > \sum_{j \neq k} |a_{kj}|,$$

where a_{kj} is the entry in the k-th row and j-th column of the matrix. The strategy is based on selecting γ to be a weighted difference between $\sum_{k \neq j} |a_{kj}|$ and $|a_{kk}|$. The motivation for this is that the disparity between the two terms gives an indication of how far the row is from being diagonally dominant. Weighting is done to ensure that γ is not too large to affect the accuracy of the resulting preconditioner.

The quantity, $\sigma = \sum_{j \neq k} |a_{kj}| - |a_{kk}|$, is referred to as *the diagonal dominance gap.* Note that the diagonal dominance gap is negative for strictly diagonal rows, in which case we need not employ any shifting. Consider the simple situation where the diagonal entry a_{kk} is perturbed so as to increase its modulus by $\rho\sigma$. Note that this is slightly different from the above requirement where the square of the modulus is augmented by γ^2. For some weight ρ, the gap σ_B after the shift is

$$\begin{aligned} \sigma_B &= \sum_{j \neq k} |a_{kj}| - \left[|a_{kk}| + \rho \left(\sum_{j \neq k} |a_{kj}| - |a_{kk}| \right) \right] = \left(\sum_{j \neq k} |a_{kj}| - |a_{kk}| \right) (1 - \rho) \\ &= \sigma(1 - \rho). \end{aligned}$$

With this viewpoint, the diagonal dominance gap will be reduced by a factor of $1 - \rho$ and when $\rho = 1$ we obtain a row that is weakly diagonally dominant since $\sigma_B = 0$. The parameter ρ plays an important role in the size of the resulting shift. Note that ρ can be viewed as a scaling factor for the diagonal dominance gap and it is simpler if this scaling remained consistent across the rows of the matrix. That is, for each row k, select γ as the same fraction of σ. Here, ρ is chosen as $\frac{n}{nnz}$, where n is the size of the matrix, and nnz is the number of non-zero entries in the matrix. Thus, for row k, γ is formally defined as:

$$\gamma_k = \left(\sum_{j \neq k} |a_{kj}| - |a_{kk}| \right) \times \frac{n}{nnz} \ . \tag{2.21}$$

This is simply the diagonal dominance gap of row k scaled by the average number of nonzero elements per row in the matrix. Subsequently, this strategy for choosing α will be referred to as the dd-based (diagonally dominant based) scheme.

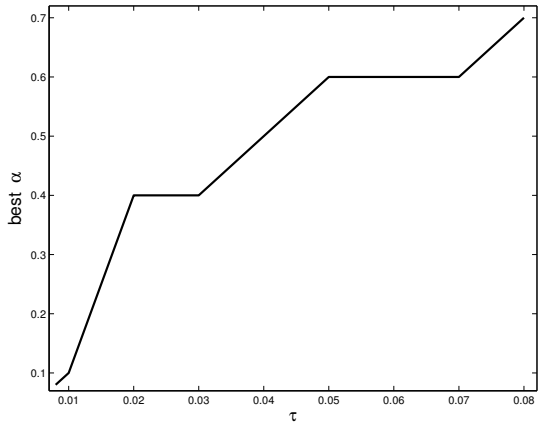


Figure 2.11: The effect of the drop tolerance τ, on the shift α. Values of τ are chosen so that the resulting fill-factor is ≥ 1

The second strategy for selecting γ is based on two ideas. The first is the assumption that each element that is dropped during the factorization is of the order of the drop tolerance τ. This is a reasonable assumption since during the factorization process, terms are dropped relative to some scaling parameter, which is obtained as a multiple of τ. The second comes from the observation that the best choice of α depends on τ. Although this is not readily obvious, it is not altogether surprising. Standard ILUT yields a more accurate factorization (smaller $\|A - LU\|$) each time the drop tolerance is reduced (although this factorization may be unstable). Suppose it is also a relatively stable factorization. Then there need not be a large perturbation that could compromise accuracy. In the case where the factorization is unstable, again only a small perturbation may be necessary to improve stability as shown in Section 2.2.2. Now suppose that the problem is such that a relatively large drop tolerance is needed to stabilize the factorization of the original matrix A. Then, again referring to Section 2.2.2, a large shift could further improve stability without significantly affecting accuracy. Figure 2.11 shows the relation between τ and the resulting optimal (or best) value of α for solving

the above Helmholtz problem over different values of τ. The idea here is to define γ in terms of the drop tolerance τ. A simple approach which works well is to weigh τ by an original norm of the row. Thus, for this scheme, we simply set γ as

$$\gamma_k = \tau \ \|a_{k:}\|_1, \tag{2.22}$$

where $||a_{k:}||_1$ denotes the 1-norm of the k-th row of A. This strategy will be subsequently referred to as the τ-based scheme.

2.2.4 Numerical Results

Application to the Helmholtz Equation

The following set of experiments compares standard ILUT with its shifted variants on problems on acoustic wave diffraction, governed by the Helmholtz equation. The right hand side for the linear system is artificially constructed by assuming a solution of all ones. For each preconditioner, the ILUT `lfil` parameter is set to be large, so that dropping is controlled by the drop tolerance τ. The iterative solution scheme uses restarted (flexible) GMRES, with a restart dimension of 60, and convergence of the iterative solver is assumed whenever the initial residual is reduced by a factor of $1.0\text{x}10^{-8}$.

In the first test case, the model problem has size $n = 29241$, with wave number $k = 8\pi$. This corresponds to a mesh resolution of $\frac{\lambda}{h} = 20$, where λ is the wavelength of the diffracted wave, and h is the element mesh size parameter. The fill-factor for each solve is fixed at ≈ 3.0 to allow for a fair comparison, and a maximum of 500 iterations is allowed for each test case. Figure 2.12 shows the convergence profile for the numerical solution of the above problem with the ILUT preconditioner and its shifted variants. For this example, standard (unshifted) ILUT failed to converge for the fill-factor allowed. In fact the factorization produced by this scheme under the conditions was very unstable - $||(LU)^{-1}\mathbf{1}||_2 \approx 1.3e+24$. Under the same conditions, we see that the shifted ILUT preconditioners handled the problem well. Shifting with the dd-based scheme converged although the factorization was not very stable either ($||(LU)^{-1}\mathbf{1}||_2 \approx 5.3e+14$). However, the result with the τ-based scheme was more impressive. It produced the smallest indicator of stable factorization ($||(LU)^{-1}\mathbf{1}||_2 \approx 2.4e+03$), and also yielded the best convergence.

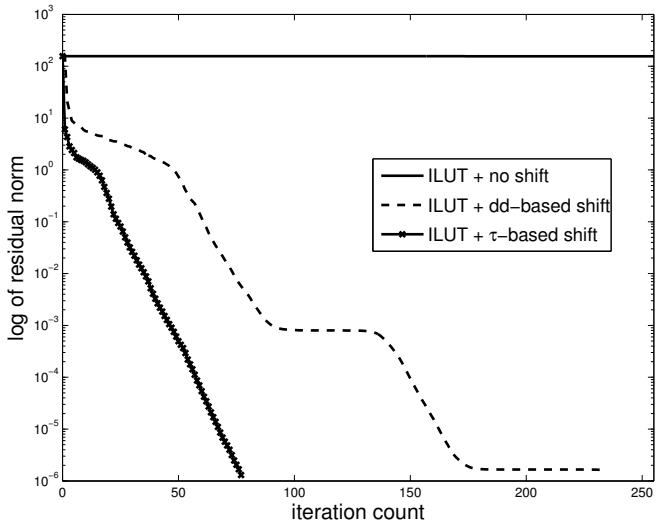


Figure 2.12: Convergence profile for the standard ILUT preconditioner and its shifted variants.

Next we investigate the effect of the wave number k, on the different preconditioners. In particular, we are interested in whether or not shifting can help solve the Helmholtz problem at high wave numbers. The solution to the Helmholtz problem at high wave numbers is a highly researched topic. Earlier work in [65] showed that the solution to this problem with standard ILUT is not very effective. This is because at high wave numbers, the Helmholtz problem is very indefinite, often resulting in an unstable factorization. Sometimes a relatively high drop tolerance (and hence fewer fill-ins) may be required to drop terms that may trigger unstable pivots during the factorization. However, this is usually not enough to guarantee convergence as it may also compromise the accuracy of the factorization.

For this example, we solve the Helmholtz problem on a 121 by 361 regular mesh. The discretized system has size $n = 43,681$, with $nnz = 387,000$ non-zero elements. We solve the problem for the wave numbers $k = 4\pi$, 8π, 16π, and 24π. Note that because we consider a fixed grid, this makes the overall mesh resolution, $\frac{\lambda}{h}$, (measured in number of points per wavelength, ppw) decreasing as we increase k, leading to more indefinite and challenging problems.

Table 2.4 shows the results for this example. Results for the time taken to solve

Preconditioner	k	$\frac{\lambda}{h}$	iters	c_F	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$
ILUT (no shift)	4π	60	134	2.32	$3.65e+03$
	8π	30	263	2.25	1.23e+04
	16π	15	–	–	–
	24π	10	–	–	–
ILUT (dd-based)	4π	60	267	2.24	$2.29e+03$
	8π	30	255	2.23	4.73e+03
	16π	15	101	3.14	6.60e+02
	24π	10	100	3.92	2.89e+02
ILUT (τ-based)	4π	60	132	2.31	$2.98e+03$
	8π	30	195	2.19	4.12e+03
	16π	15	75	3.11	7.46e+02
	24π	10	86	3.85	2.73e+02

Table 2.4: Comparison of the different schemes for the ILUT preconditioner on the Helmholtz application problem with different wave numbers.

the system is not included here since the cost of each of the preconditioners are identical. The memory usage (or fill-factor) and the iteration count, together serve as good measures for the computational cost and hence the overall performance of the solver. The results indicate that standard ILUT stagnated for high values of k. At low values of k, it performed well, even doing better than with the dd-based scheme for the case with $k = 4\pi$. However, as k increases and the system becomes more indefinite, the performance of the unshifted ILUT deteriorates, and the shifted schemes perform better. Once again the τ-based scheme outperforms the rest for all the different values of k.

2.3 conclusion

In this section we have discussed the effect of diagonal perturbation techniques on the standard ILU(0) factorization for structured matrices originating from discretized partial differential equations. A criterion for selecting the shift has been proposed, and numerical examples have been used to illustrate its effect. The results indicate that when imaginary perturbations are used on a real (symmetric) matrix, a normal equations formulation of the problem can yield good results. Note, however, that this approach is not necessarily viable for large linear systems.

Several authors have previously observed that adding purely imaginary shifts to the Laplace operator is a simple yet effective strategy for improving solution methods for the Helmholtz equation. Here, we have adapted this technique to the algebraic context of ILU-type preconditioners. Our observation is that perturbing the diagonal entries of an indefinite system by a small complex perturbation can significantly improve the quality of the incomplete LU factorization of the matrix. Two different heuristics for selecting the complex shift were considered and tested with the ILUT preconditioning technique. Both strategies resulted in improved and more stable factors. The test results suggest that the drop tolerance used for dropping small terms during the factorization should be taken into account in the selection of the shift. Choosing the shift as the product of the drop tolerance and some original row norm (the 1-norm for instance) yielded good results overall, though much remains to be done to determine a truly optimal strategy.

Chapter 3

Modified ILU and Compensation-based Techniques

Among the various options of ILU considered in the literature is the Modified ILU factorization (MILU) proposed by Gustafsson [76] for the symmetric case (Modified Incomplete Cholesky or MIC). Note that for 5-point matrices, MIC(0), where the nonzero pattern of the resulting factors is restricted to match that of the original matrix, is equivalent to the method proposed in 1968 by Dupont, Kendall, and Rachford [28]. The modification in the MIC(0) technique consists of, for every row, tracking the entries dropped in the factorization and adding their sum to the diagonal entry of that same row. This has the same effect as adding to the diagonal entry before computing the factorization, as in the shifted ILU (or shifted incomplete Cholesky) factorization [19,68].

The result of the modified ILU process is that the product of the factors, LU, and the original matrix, A, give the exact same result when applied to a vector with constant entries. This rationale is derived from a continuity argument: the matrix, LU, that approximates A should be exact on constants in the domain when A corresponds to the discretization of an elliptic PDE. It can be observed that for matrices arising from such PDEs, MILU may be vastly superior to a standard ILU, and this improvement comes at virtually no extra cost. The method has been extensively analyzed and has given rise to a few variants; see, for example, [76–80]. However, most of the work on these so-called 'diagonal compensation' techniques, to which MILU belongs, has been devoted

to matrices arising from PDEs, as the analysis for the case of general sparse matrices is difficult. For the same reason, the use of these techniques for the threshold-based factorization, ILUT, has been avoided.

The focus of this chapter is on combining modification strategies with threshold-based factorization, leading to new modified ILUT (MILUT) algorithms. Here, we aim to look at the role of modification of the triangular factors within threshold-based incomplete factorizations from the point of view of both stability and accuracy. In the case of symmetric positive-definite M-matrices, earlier work and computational experience suggests that modification can improve accuracy of level-of-fill based factorizations, while not harming their stability.

In order to best address the questions of accuracy and stability within modified ILUT, we consider the following questions:

1. Can we extend the classical modified ILU idea, based on diagonal compensation techniques, to safeguard the stability of the modified ILUT factorization?

2. For indefinite problems, can we use the ideas of complex-valued (and purely imaginary) perturbations to improve stability of the modified ILUT preconditioners.

To answer these, we present two approaches that can be used to improve standard modification procedures. First, in Section 3.1, we examine the question of relaxed compensation, where the modification process is tempered with the goal of improving the stability of the resulting LU factors. Here, we propose a strategy that aims to balance the accuracy of the MILUT factors with their stability. Secondly, Section 3.2 extends this approach using complex-valued modifications, particularly in the context of indefinite operators; such approaches have been examined in many recent papers aimed at the numerical solution of the Helmholtz Equation (see, for example, [63, 71–73]). Numerical results for these methods are given in Section 3.4. Section 3.5 presents some concluding remarks.

Throughout this chapter, the superscript H is used to denote the Hermitian transpose, as we consider matrices and vectors that may be complex-valued. In the case of real arithmetic, the superscript T is used instead.

3.1 Optimal spreading for modified ILUT

In this subsection, we discuss an extension to the classical modified ILU idea by considering an optimal way of distributing the compensation term among the non-unit diagonal as well as the L part of the factorization. Here and in Section 3.2, we focus on the column version of ILUT given earlier in Section 1.3.1 as Algorithm 3.

Recall that from Equation (1.2), $\hat{\mathbf{w}}$ is the vector that results from eliminating nonzeros in positions 1 through $j-1$ in the j-th column of A, $\mathbf{a}_j$. In the modified version of ILUT, this column undergoes another modification before it is stored as a column of L. Specifically, we write the modification as the addition of a column $\mathbf{s}$, giving

$$\mathbf{a}_j - w_1\mathbf{l}_1 - w_2\mathbf{l}_2 \cdots - w_{j-1}\mathbf{l}_{j-1} - (\hat{\mathbf{w}} + \mathbf{s}) = \epsilon_U + \epsilon_L - \mathbf{s}. \tag{3.1}$$

We then obtain the diagonal entry of U by setting $u_{j,j} = \hat{w}_j + s_j$ and the j-th column of L as $\mathbf{l}_j = (\hat{\mathbf{w}} + \mathbf{s})/w_j$.

Before discussing the choices of $\mathbf{s}$, we note an important practical consequence about the effect of *post-dropping* in the U part of $\mathbf{w}$. By this, we mean dropping further entries among $w_1, ..., w_{j-1}$, after elimination step j is complete (i.e., after Line 5 of Algorithm 3). If, at this stage, any entry, w_k for $1 \leq k \leq j-1$, is dropped (i.e., assigned a value of 0), then the resulting correction to Equation (1.2) becomes more complicated, as it is not just w_k, but the column $w_k\mathbf{l}_k$, that must be subtracted from the right-hand side in order for Equation (3.1) to hold. This means $\epsilon = \epsilon_U + \epsilon_L$ will be modified by the addition of $w_k\mathbf{l}_k$, which can lead to significant additional fill-in elements. Furthermore, this additional fill-in is not easy to track without additional sparse vector calculations. This suggests that it is reasonable to avoid post-dropping in U when we base our analysis on (3.1) and, so, we adopt this strategy here. An analogous strategy is possible for the lower-triangular factor in the row-based ILUT algorithm.

Following (3.1), we consider the modified vector,

$$\hat{\mathbf{w}} + \mathbf{s} = \begin{pmatrix} \mathbf{0} \\ \eta \\ \mathbf{l} \end{pmatrix} + \begin{pmatrix} \mathbf{0} \\ \sigma \\ \mathbf{z} \end{pmatrix}, \tag{3.2}$$

where, as discussed above, we choose $\mathbf{s}$ to only affect the lower-triangular part of the factorization. In Equation (3.2), η represents the diagonal entry of the column after

elimination (but before scaling), σ is a perturbation to this diagonal entry, the column vector, $\mathbf{l}$, represents the strict lower part of the corresponding (unmodified) column in L, and $\mathbf{z}$ is the perturbation to $\mathbf{l}$.

In the classical modified ILU factorization, we simply take $\mathbf{z}$ to be the zero vector and ask that the inner product of the error vector, $\epsilon - \mathbf{s}$, and a given vector, $\mathbf{t}$, be zero. In the most common case, $\mathbf{t} = \mathbf{1} \equiv (1, 1, \cdots, 1)^T$, giving $\mathbf{1}^T(\epsilon - \sigma \mathbf{e}_j) = 0$ (where $\mathbf{e}_j$ is the j-th column of the identity), or $\sigma = \mathbf{1}^T\epsilon$, i.e., the sum of the dropped entries. We refer to this approach as modified ILU with exact diagonal compensation, since the exact sum of the dropped entries is added back onto the diagonal. We shall denote this optimal (exact) compensation, that satisfies the matching constraint with respect to the vector $\mathbf{t}$, as σ^*. The more general scenario, where $\mathbf{t}$ is an arbitrary vector with no zero elements, leads to

$$\mathbf{t}^H(\epsilon - \sigma \mathbf{e}_j) = 0 \quad \rightarrow \quad \sigma = \sigma^* \frac{\mathbf{t}^H \epsilon}{\mathbf{t}^H \mathbf{e}_j} \ . \tag{3.3}$$

3.1.1 Relaxed compensation

As mentioned earlier, ILUT behaves quite differently from the usual ILU(k) strategies and, in particular, exact diagonal compensation may affect the factors in a negative manner when the modified diagonal term, $\eta + \sigma$, is closer to zero than η, decreasing the diagonal dominance of the row or column under consideration.

To motivate our proposed strategy, we begin by examining Equation (3.1). Consider the general situation where the 'compensation column', $\mathbf{s}$, is any column which has at most the same sparsity pattern as $\hat{\mathbf{w}}$. Note that ϵ (the vector of dropped entries) and $\hat{\mathbf{w}}$ are structurally orthogonal and, therefore, so are $\mathbf{s}$ and ϵ. Thus, $\|\epsilon - \mathbf{s}\|_2^2 = \|\epsilon\|_2^2 + \|\mathbf{s}\|_2^2$, which suggests that, from the point of view of accuracy, we should keep $\|\mathbf{s}\|_2$ small. Regarding Equation (3.2), however, we would like to add a portion of the dropped entries to $\hat{\mathbf{w}}$, with the goal of making the scaled column of L as 'stable' as possible. This means that, from the point of view of stability, we want to modify the diagonal entry of $\hat{\mathbf{w}} + \mathbf{s}$ (i.e., the diagonal entry of U) and, possibly, the lower part as well so that (cf. (3.2)),

A. $|\eta + \sigma|$ is not small

B. $\|\mathbf{l} + \mathbf{z}\|_2$ is as small as possible

Considering (A) alone, we must balance the contrasting requirements of accuracy and stability. Although it is desirable to make the modified diagonal entry $|\eta+\sigma|$ large relative to other entries, one should note that choosing σ to be arbitrarily large would result in a factorization that poorly approximates the original matrix, A. Thus, it is necessary to control the size of σ. However, simply applying the exact compensation, by choosing $\sigma = \sigma^*$ as prescribed in (3.3) can adversely affect stability by taking the diagonal term closer to zero when η and σ have opposite signs. Hence, considering (A), one option is to add a fraction, $\alpha > 0$, of σ^*, so that $\sigma = \alpha\sigma^*$.

Considering (B), we aim to choose $\mathbf{z}$, a sparse column with the same sparsity pattern as $\mathbf{l}$, to simultaneously minimize $\|\mathbf{z}\|_2$ (the additional discarded fill-in) and $\|\mathbf{l}+\mathbf{z}\|_2$, relative to $\eta+\sigma$ (the resulting column of L). However, it is difficult to solve this optimization problem, particularly where both unknowns (σ and $\mathbf{z}$) are considered at the same time. Thus, we formulate an optimization strategy that handles conditions (A) and (B) above with $\mathbf{z}$ as the only unknown; subsequently, we exploit the solution of this optimization problem as a guide for choosing σ.

So, fixing σ, we pose the optimization problem of

$$\min_{\mathbf{z}} \quad \frac{\|\mathbf{l}+\mathbf{z}\|_2^2}{|\eta+\sigma|^2} \quad \text{subject to the constraint} \quad |\sigma|^2 + \|\mathbf{z}\|_2^2 \leq \gamma^2 \tag{3.4}$$

where the constraint, $|\sigma|^2 + \|\mathbf{z}\|_2^2 \leq \gamma^2$ (for $\gamma > 0$), is used to safeguard the accuracy of the factorization as an approximation to the original matrix, A. To solve Problem 3.4, we introduce the penalty term, $\mu \geq 0$, and solve instead the penalized problem,

$$\min_{\mathbf{z}} \left(\frac{1}{|\eta+\sigma|^2}(\mathbf{l}^H\mathbf{l} + 2\mathbf{z}^H\mathbf{l} + \mathbf{z}^H\mathbf{z}) + \mu\left(|\sigma|^2 + \mathbf{z}^H\mathbf{z} - \gamma^2\right)\right). \tag{3.5}$$

Proposition 3.1.1. *The minimum of Problem 3.5 is reached when*

$$\mathbf{z} = \frac{-1}{(1+\mu|\eta+\sigma|^2)}\,\mathbf{l}. \tag{3.6}$$

Proof. Taking the derivative of the penalized problem, and setting to zero gives:

$$\frac{2(\mathbf{l}+\mathbf{z})}{|\eta+\sigma|^2} + 2\mu\mathbf{z} = 0.$$

Simplifying, we obtain

$$\mathbf{z}(1+\mu|\eta+\sigma|^2) = -\mathbf{l}$$

and the result follows immediately by solving for $\mathbf{z}$. □

From the KKT conditions, complementary slackness implies that $\mu(|\sigma|^2+\mathbf{z}^H\mathbf{z}-\gamma^2) = 0$. Recall that $\mu \geq 0$, and notice that, if $\mu = 0$, then the constraint is inactive. Hence, for $\mu > 0$, we have

$$\mathbf{z}^H\mathbf{z} = \gamma^2 - |\sigma|^2. \tag{3.7}$$

Substituting $\mathbf{z}$ from Equation (3.6) into Equation 3.7 above, gives:

$$|\eta + \sigma|^4\mu^2 + 2|\eta + \sigma|^2\mu + \left(1 - \frac{\mathbf{l}^H\mathbf{l}}{\gamma^2 - |\sigma|^2}\right) = 0.$$

We can then solve for μ as:

$$\mu = \frac{1}{|\eta + \sigma|^2}\left(-1 + \sqrt{\frac{\mathbf{l}^H\mathbf{l}}{\gamma^2 - |\sigma|^2}}\right). \tag{3.8}$$

To obtain a valid solution for μ, two conditions need to be satisfied. First, we need $\mu > 0$, implying that

$$\frac{\mathbf{l}^H\mathbf{l}}{\gamma^2 - |\sigma|^2} > 1 \quad \Rightarrow \quad |\sigma|^2 > \gamma^2 - \mathbf{l}^H\mathbf{l}.$$

Notice that, since $|\sigma|^2 > 0$ and γ^2 may be less than $\mathbf{l}^H\mathbf{l}$, we can express the above inequality, without loss of generality, as

$$|\sigma| > \sqrt{\max(\gamma^2 - \mathbf{l}^H\mathbf{l}, 0)},$$

where $\max(a, b)$ simply returns the maximum of a and b. While this gives a lower bound for σ, we also need to ensure that σ is not too large, so that μ remains real-valued. Further requiring that

$$\frac{\mathbf{l}^H\mathbf{l}}{\gamma^2 - |\sigma|^2} > 0 \quad \Rightarrow \quad |\sigma| < \gamma.$$

Putting these two inequalities together, we get that

$$\sqrt{\max(\gamma^2 - \mathbf{l}^H\mathbf{l}, 0)} < |\sigma| < \gamma.$$

In real arithmetic, this reduces to

$$-\gamma < \sigma < -\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)} \quad \text{or} \quad \sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)} < \sigma < \gamma. \tag{3.9}$$

3.1.2 Choice of σ

The parameter γ controls the size of the modifications, $\|\mathbf{s}\|_2^2 = |\sigma|^2 + \|\mathbf{z}\|_2^2$, and, hence, must be carefully chosen. Choosing $\gamma = |\sigma^*|$, the exact diagonal compensation factor from Equation (3.3), naturally lends itself to the modified ILUT scheme, since it guarantees that σ will be a fraction of the weighted sum of the dropped terms. For the special case of $\sigma^* = 0$ (when there are no dropped terms or, more generally, when $\mathbf{t}^H\epsilon = 0$), the choice of $\gamma > 0$ can be based on a default value, such as the drop tolerance, τ, used in ILUT. When $\gamma > 0$, the sign of σ is chosen to match that of the exact diagonal compensation factor. Thus, in real arithmetic, if $\sigma^* < 0$, then σ is chosen to satisfy $-\gamma < \sigma < -\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)}$, otherwise, $\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)} < \sigma < \gamma$.

It remains to select the size of σ within the interval $(\sqrt{\max(\gamma^2 - \mathbf{l}^H\mathbf{l}, 0)}, \gamma)$, given that the optimization problem provides no further guidance. We propose a criterion based on choosing σ within its allowable interval in order to address the stability of the resulting factors. Thus, our choice is guided by the size and sign of the resulting perturbations. For simplicity, we assume real arithmetic in our analysis, and provide a generalization to complex arithmetic afterwards.

First note that when $\mathbf{l} = \mathbf{0}$, the optimization problem clearly attains a minimum with $\mathbf{z} = \mathbf{0}$. Furthermore, extending the inequalities in Equation (3.9) gives $|\sigma| = \gamma$. Thus, we must consider the choice of σ only when $\mathbf{l} \neq \mathbf{0}$. In this case, we have a nontrivial range of possible values of σ, and must choose whether $|\sigma|$ should be closer to γ or $\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)}$. To make this decision, we return to criterion (A), which states that $|\eta+\sigma|$ should not be "small". Thus, if η and σ are of different signs, it is sensible to select $|\sigma|$ to be close to the lower bound $\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)}$, so that $|\eta + \sigma|$ is not made smaller than necessary. Assuming this choice of σ is small enough, then from Equation (3.8), we see that μ is not too large (since $\gamma^2 - \sigma^2$ is larger) and, hence, $\mathbf{z}$ is a significant modification on $\mathbf{l}$. In other words, adding σ to the diagonal improves the matching condition, whereas $\mathbf{z}$ serves to stabilize the column (balancing out the negative effect of σ). However, if η and σ are of the same sign, then it is sensible to choose $|\sigma|$ to be close to the upper bound, γ, to benefit from both improving the matching condition as well as stabilizing the column. Notice that this choice of $|\sigma|$ increases the magnitude of μ, and hence, decreases the norm of $\mathbf{z}$.

The choice for σ is, then, formally defined as:

$$\sigma = \begin{cases} 0 & \text{if} \quad \mathbf{l}^T\mathbf{l} = 0, \\ sgn(\sigma^*) \times (\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)} + \beta \times \rho) & \text{if} \quad sgn(\eta) = sgn(\sigma^*), \\ sgn(\sigma^*) \times (\sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)} + \varepsilon \times \rho) & \text{otherwise,} \end{cases} \tag{3.10}$$

where $sgn(.)$ is the signum function, $\rho = \gamma - \sqrt{\max(\gamma^2 - \mathbf{l}^T\mathbf{l}, 0)}$ is the width of the interval for choosing σ, $\varepsilon \in (0, 1)$ is small, and $\beta \in (0, 1)$ is not small. For general sparse matrices, we propose setting ε to be equal to the drop tolerance, τ, of the ILUT factorization and β to be equal to the "diagonal dominance ratio" of the column, given by $\beta = |a_{j,j}|/\|\mathbf{a}_j\|_1$. If the j-th column of the original matrix, A, is diagonally dominant, then β is closer to one, making σ relatively large. As a result, μ is also large, which makes the modification with $\mathbf{z}$ small. Since the column is already diagonally dominant, the modification with $\mathbf{z}$ need not be significant. On the other hand, if the j-th column of A is not diagonally dominant, then β is small, and the resulting modification with $\mathbf{z}$ is significant.

For matrices with complex coefficients, Equation (3.10) takes the more general form:

$$\sigma = \begin{cases} 0 & \text{if } \mathbf{l}^H\mathbf{l} = 0, \\ sgn(\sigma^*) \times (\sqrt{\max(\gamma^2 - \mathbf{l}^H\mathbf{l}, 0)} + \beta \times \rho) & \text{if } \begin{cases} sgn(Re(\eta)) = sgn(Re(\sigma^*)) \text{ and} \\ sgn(Im(\eta)) = sgn(Im(\sigma^*)), \end{cases} \\ sgn(\sigma^*) \times (\sqrt{\max(\gamma^2 - \mathbf{l}^H\mathbf{l}, 0)} + \varepsilon \times \rho) & \text{otherwise,} \end{cases}$$

where $Re(.)$ and $Im(.)$ represent the real and imaginary parts of the argument, respectively. The main difference between this form and Equation (3.10) is that the sign comparison between η and σ^* takes into account both the real and imaginary parts of the complex term. Also, note that since $sgn(\sigma^*) = \sigma^*/|\sigma^*|$ is complex, σ will take on a complex value as well.

3.1.3 Effect of the modification by z

As discussed above, the modification, $\mathbf{z}$, helps to improve the stability of the column during the factorization, particularly when the modification with σ causes the diagonal to be small. Nonetheless, it is worth noting that like any other perturbation method, modification with $\mathbf{z}$ can make the factorization a less accurate representation of the

original matrix, although the resulting factors may be stable. Thus, large modifications with $\mathbf{z}$ should be avoided. It is for this reason that γ is chosen to be small. The net effect of the above optimization problem, together with the discussion on the choice of σ, is to try to balance the effects on the stability and accuracy of the factorization with respect to σ and $\mathbf{z}$; σ improves the matching condition and, in some cases, helps to stabilize the column, while $\mathbf{z}$ improves stability without dominating the effect of σ.

3.2 Complex modification

While the above modification strategy offers an improvement on unmodified ILUT, even for some very poorly conditioned systems (see Section 3.4.1), we still observe poor performance when the original system matrix is very indefinite. Chapter 2 shows that purely imaginary shifts of symmetric and indefinite problems have the effect of clustering eigenvalues on a circle in the right half-plane, with an accumulation point near one. This results in a more stable factorization, which leads to a more effective ILU-based preconditioner. The diagonal compensation technique described in Chapter 2, as well as in previous work such as [19, 71–73], is based on shifting the diagonal entries, by an appropriate perturbation, prior to performing the ILU factorization (shifted ILU). This is somewhat different from the classical modified ILU approach, which is based on matching the effect of the preconditioning matrix and the original matrix on some vector. Nonetheless, ideas from shifted ILU methods can be incorporated into the modified ILU scheme.

3.2.1 Imaginary perturbations for modified ILUT

While the shifted ILU (and also shifted multigrid) approaches have been shown to improve on unshifted preconditioners for indefinite, Helmholtz-type problems, modified approaches based on similar reasoning offer better control of the perturbation and, so, more accurate preconditioners for these problems. In what follows, we present an approach for using imaginary perturbations similar to the compensation strategy described earlier in this chapter, to improve the quality of the modified ILUT factorization. To do so, the above optimization problem is solved replacing σ by $i\sigma$ (where $\sigma \in \mathbb{R}$). In

other words, we rewrite the problem as

$$\min_{\mathbf{z}} \left(\frac{1}{|\eta + i\sigma|^2} (\mathbf{l}^H \mathbf{l} + 2\mathbf{z}^H \mathbf{l} + \mathbf{z}^H \mathbf{z}) + \mu(\ |\sigma|^2 + \mathbf{z}^H \mathbf{z} - \gamma^2) \right).$$

Following the same approach as before yields

$$\mathbf{z} = \frac{-1}{(1 + \mu|\eta + i\sigma|^2)} \ \mathbf{l}.$$

Note that the penalty term, μ, remains essentially the same as in Equation (3.8), giving the optimal ranges for σ as

$$-\gamma < \sigma < -\sqrt{\max(\gamma^2 - \mathbf{l}^H \mathbf{l}, 0)} \quad \text{or} \quad \sqrt{\max(\gamma^2 - \mathbf{l}^H \mathbf{l}, 0)} < \sigma < \gamma.$$

In this approach, the sign of σ is chosen to match the sign of the imaginary part of the diagonal term, η (if η is complex; if η is real, the sign can be chosen arbitrarily). Note that this is similar to the approach taken in Section 2.2 for shifted ILUT with imaginary perturbations. This increases $|\eta + i\sigma|$, making the factorization more diagonally dominant. We then need to decide whether to take σ to be close to its lower or upper bound within this interval. As before, choosing $|\sigma|$ to be close to the lower bound leads to smaller values of μ and, thus, a larger correction in $\mathbf{z}$. Choosing a larger $|\sigma|$ (close to the upper bound), on the other hand, gives a smaller correction in $\mathbf{z}$. This latter option is more appealing, since the increase in the diagonal entries from the imaginary perturbation is offset by the more accurate (less perturbed) modification of the rest of $\hat{\mathbf{w}}$. This gives

$$\sigma = s \times \left(\sqrt{\max(\gamma^2 - \mathbf{l}^H \mathbf{l}, 0)} + \beta \times \rho \right),$$

where ρ is the size of the interval as before, β is close to 1 (e.g., $\beta = 1 - \tau$, where τ is the ILUT drop tolerance), and the sign, s, is defined as:

$$s = \begin{cases} 1 & \text{if} \quad Im(\eta) > 0, \\ -1 & \text{otherwise.} \end{cases}$$

3.3 The modified ILUT algorithm with relaxed compensation

Algorithm 4 formally describes the column version of the modified ILUT scheme discussed above.

Algorithm 4: Left-looking or JKI ordered MILUT.

0. Select a vector, $\mathbf{t}$, for matching
1. For $j = 1 : n$, do:
2. $\quad \mathbf{w} = A_{1:n,j}$
3. $\quad$ Initialize $\gamma = 0$
4. $\quad$ For $k = 1 : j-1$ and if $w_k \neq 0$, do:
5. $\quad\quad$ Apply first dropping rule to w_k
6. $\quad\quad$ If w_k is not dropped, $\mathbf{w}_{k+1:n} = \mathbf{w}_{k+1:n} - w_k \cdot L_{k+1:n,k}$
7. $\quad\quad$ Else, $\gamma = \gamma + w_k \cdot t_j$
8. $\quad$ Enddo
9. $\quad$ Apply second dropping rule to $\mathbf{w}_{j+1:n}$
10. $\quad$ For $k = j+1, \ldots, n$, if $\mathbf{w}_k$ is dropped, update $\gamma = \gamma + w_k \cdot t_j$
11. $\quad$ Do relaxed compensation on $\mathbf{w}_j$ and $\mathbf{w}_{j+1:n}$ with parameter γ
12. $\quad$ For $i = j+1, \ldots, n$, $l_{i,j} = w_i / w_j$ $(l_{j,j} = 1)$
13. $\quad$ For $i = 1, \ldots, j$, $u_{i,j} = w_i$
14. Enddo

The above algorithm extends Algorithm 3, by tracking the dropped terms to satisfy the matching condition. At Line 9 of the above algorithm, we prune the L part of $\mathbf{w}$, by applying the second dropping rule, prior to performing the modification. This is necessary because we need to scale the resulting (modified) column of L by the (modified) diagonal. Furthermore, notice that we avoid post-dropping in U. The only dropping in U is handled by the first dropping rule (at Line 5 of the algorithm).

In Line 10 of the algorithm, the updates to the size parameter, γ, accumulate terms of the form $w_k t_j$ and not, as is usually done in a row-based calculation, $w_k t_k$. Although we calculate the ILU factorization in a column-wise manner, the matching condition is always a row-wise condition, that is, $(A\mathbf{t})_i = (LU\mathbf{t})_i$. Accumulating γ, however, is a column-wise calculation. Thus, we use row-wise values of w_k, the dropped fill-in entries, but fix t_j based on the column that we are considering.

3.4 Numerical Results

In what follows, we present some numerical results for the modified ILUT schemes described in the previous section. These results are obtained from applications governed

by the standard 2D finite-difference Laplacian, some shifted and scaled problems, and the Helmholtz Equation.

3.4.1 Results for modified ILUT with Relaxed Compensation

We present some numerical results for the relaxed compensation strategy for modified ILUT, obtained from the solution to the minimization problem described in Section 3.1.1. The tests shown here are run with the column version of the modified ILUT algorithm, programmed in C, on a dual-core AMD Opteron 1GHz machine with 4GB of RAM.

In the first set of examples, we test the relaxed compensation strategy on systems that are symmetric and positive definite, but poorly conditioned. We follow the discretization of the 2D shifted Laplace operator described in Section 1.4.3, shifted by a small negative term ϱ to make it indefinite. We then construct the normal matrix $A = L^T L$, from the resulting indefinite matrix, L. Note that although A is now symmetric and positive definite, solving a linear system involving A can still be quite challenging, due to its large condition number. Furthermore, shifting L to make it indefinite can result in a cluster of eigenvalues close to zero for the normal matrix, A, which makes it a more challenging problem for solution by Krylov methods. The right-hand side vectors, $\mathbf{b}$, are constructed by choosing the entries of the solution vector, $\mathbf{x}$, from the uniform distribution on $[0, 1]$ and computing $\mathbf{b} = A\mathbf{x}$. Since these vectors are chosen randomly, the results displayed in the tables below are obtained from averaging several runs of the same problem for each test case. The condition numbers of the resulting matrices range from 3×10^8 to 10^{11}, but do not correlate strongly with problem size or shift, due to the formation of the normal equations.

In what follows, we compare the performance of the relaxed compensation scheme discussed in this chapter against that of shifted ILU(0) on solving the system $A\mathbf{x} = \mathbf{b}$. For the shifted ILU(0) preconditioner, the diagonal of matrix A is perturbed by a small shift before performing the incomplete factorization. This shift serves to improve the quality of the ILU(0) factorization, and makes it a more effective preconditioner [10, 68, 69]. In order to select an optimal shift, we run the same problem for several different diagonal shifts within the interval $[0, 1]$, using an increment of 0.1, and select the shift yielding the best result for shifted ILU(0).

ϱ	grid	nnz	c_F	iters	time	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$	$\|\|A - LU\|\|_F$
-0.05	100×100	128004	1.0	99	$0.92s$	$4.77e+03$	137.94
	150×150	289504	1.0	107	$2.23s$	$7.55e+03$	208.48
	200×200	516004	1.0	113	$3.95s$	$1.03e+04$	279.00
	300×300	1164004	1.0	97	$7.60s$	$1.59e+04$	420.00
-0.1	100×100	128004	1.0	109	$0.99s$	$1.08e+03$	128.65
	150×150	289504	1.0	98	$2.02s$	$1.65e+03$	194.16
	200×200	516004	1.0	127	$4.36s$	$2.22e+03$	259.66
	300×300	1164004	1.0	109	$8.18s$	$3.37e+03$	390.68
-0.5	100×100	128004	1.0	169	$1.44s$	$4.48e+02$	142.44
	150×150	289504	1.0	156	$3.20s$	$6.77e+02$	215.17
	200×200	516004	1.0	150	$5.44s$	$9.06e+02$	287.89
	300×300	1164004	1.0	144	$10.47s$	$1.36e+03$	433.34

Table 3.1: Results for shifted ILU(0) on preconditioning the system $A\mathbf{x} = \mathbf{b}$, where $A = L^T L$ and L is a shifted Laplace matrix with the shift $\varrho = -0.05, -0.1, -0.5$.

Table 3.1 provides details on the numerical results for this shifted ILU(0) strategy. Here, we use a right-preconditioned restarted GMRES with a restart dimension of 100, and an initial guess $\mathbf{x}^{(0)} = \mathbf{0}$. The maximum number of outer GMRES iterations for these tests is fixed at 500. We assume convergence of the iteration when the ℓ_2-norm of the residual is reduced by a relative factor of 10^7.

As before, we use the quantity, $||(LU)^{-1}\mathbf{1}||_2$, as a measure of the stability of the factorization, and we use the quantity, $||A - LU||_F$, as a measure of the accuracy of LU as an approximation to A. We report the fill-factor, c_F, as a measure of the cost of storage for the preconditioner. We also report the total time needed for the computation of the factors and solution of the linear system. For this set of examples, we fix the matching vector, $\mathbf{t}$, used to constrain the modified ILUT method, to be the vector of all ones.

Table 3.2 shows results for the modified ILUT method with relaxed compensation on this example. The results from these tables indicate superior performance for the modified ILUT method over the shifted ILU(0) method. We observe that the modified ILUT method yields factors that are often more stable and always more accurate, and that the iteration based on modified ILUT converges in fewer iterations for all the test

ϱ	grid	nnz	c_F	iters	time	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$	$\|\|A - LU\|\|_F$
-0.05	100×100	128004	1.82	59	$0.74s$	$1.67e+03$	25.49
	150×150	289504	1.83	65	$2.05s$	$2.52e+03$	38.15
	200×200	516004	1.83	77	$3.93s$	$3.78e+03$	50.88
	300×300	1164004	1.84	69	$7.46s$	$5.09e+03$	76.42
-0.1	100×100	128004	1.82	62	$0.79s$	$1.52e+03$	25.91
	150×150	289504	1.83	60	$1.68s$	$2.28e+03$	38.11
	200×200	516004	1.83	67	$3.25s$	$3.03e+03$	51.79
	300×300	1164004	1.84	63	$6.78s$	$4.54e+03$	77.82
-0.5	100×100	128004	1.61	83	$1.05s$	$4.24e+02$	25.47
	150×150	289504	1.61	83	$2.30s$	$6.10e+02$	38.08
	200×200	516004	1.61	81	$3.61s$	$7.96e+02$	50.73
	300×300	1164004	1.61	82	$8.09s$	$1.17e+03$	76.15

Table 3.2: Results for modified ILUT with relaxed compensation on preconditioning the system $A\mathbf{x} = \mathbf{b}$, where $A = L^T L$ and L is a shifted Laplace matrix with the shift $\varrho = -0.05, -0.1, -0.5$.

cases.

The memory efficiency of ILU(0) is difficult to beat, particularly since threshold-based ILU strategies generally require some fill-ins (beyond the zero level-of-fill) in order to get a good approximation by the preconditioner. Nonetheless, the resulting added computational cost in the modified ILUT factorization (due to fill-ins), is made up for by the good iteration counts. As shown in Tables 3.1 and 3.2, the iteration count for the modified ILUT method is, on average, about 1.5 times better than that of the shifted ILU(0) method. The reduced iteration count also reduces the time taken for GMRES to converge. Indeed, since GMRES converges faster with the modified ILUT preconditioner, fewer Arnoldi vectors need to be stored and orthogonalized, resulting in even more savings than just by the simple ratio of iterations.

Standard ILUT, using the same dropping criteria as the modified ILUT method with relaxed compensation, yields poor and unstable factors for these problems. This leads to convergence failure of the restarted GMRES iterations. However, adjusting the dropping rule so that it is based on comparing the size of the fill-in entry or column update ($w_k \cdot L_{j,k}$, $j = k+1, ..., n$ in Line 3 of Algorithm 3), to some scaling parameter

ϱ	grid	nnz	c_F	iters	time	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$	$\|\|A - LU\|\|_F$
-0.05	100×100	128004	1.75	183	$2.22s$	$2.26e+02$	38.15
	150×150	289504	1.76	186	$5.03s$	$3.38e+02$	57.47
	200×200	516004	1.76	212	$9.94s$	$4.50e+02$	76.78
	300×300	1164004	1.76	198	$21.70s$	$6.72e+02$	115.40
-0.1	100×100	128004	1.75	157	$1.87s$	$2.17e+02$	39.66
	150×150	289504	1.76	149	$3.96s$	$3.23e+02$	59.88
	200×200	516004	1.76	186	$8.71s$	$4.29e+02$	80.09
	300×300	1164004	1.76	177	$20.32s$	$6.41e+02$	120.52
-0.5	100×100	128004	1.60	130	$1.86s$	$1.26e+02$	62.91
	150×150	289504	1.60	154	$3.88s$	$1.86e+02$	94.90
	200×200	516004	1.61	141	$6.23s$	$2.46e+02$	127.04
	300×300	1164004	1.61	138	$15.30s$	$3.66e+02$	190.94

Table 3.3: Results for standard ILUT (although with a modified dropping rule) on preconditioning the system $A\mathbf{x} = \mathbf{b}$, where $A = L^T L$ and L is a shifted Laplace matrix with the shift $\varrho = -0.05, -0.1, -0.5$.

related to the drop tolerance, yields factors that were "good enough" to handle the problem. Notice that this dropping rule differs from the approach discussed in Section 1.3.1 (see Algorithms 2 and 3), where the comparison is directly between the pivot element, w_k, and the scaling parameter. Results for this approach are shown in Table 3.3. For a fair comparison with the results in Table 3.2, we aim to attain similar fill-factors for both the standard ILUT method and the modified ILUT method. Comparing the results in Tables 3.2 and 3.3, we see that the modified ILUT method shows better performance than the standard ILUT method, particularly in terms of the iteration counts, computational times, and accuracy. Nonetheless, we observe that the standard ILUT method shows better stability measures than the modified ILUT method. This may be attributed to the new dropping rule imposed to help the convergence of the standard ILUT method.

One interesting observation is that as more fill-ins are allowed in the factorization, the performance of the standard ILUT method deteriorates. We further observe that small decrements in the drop tolerance, τ, often result in sudden (significant) jumps in the fill-factor, c_F (sometimes by an order of magnitude). From the standard ILUT

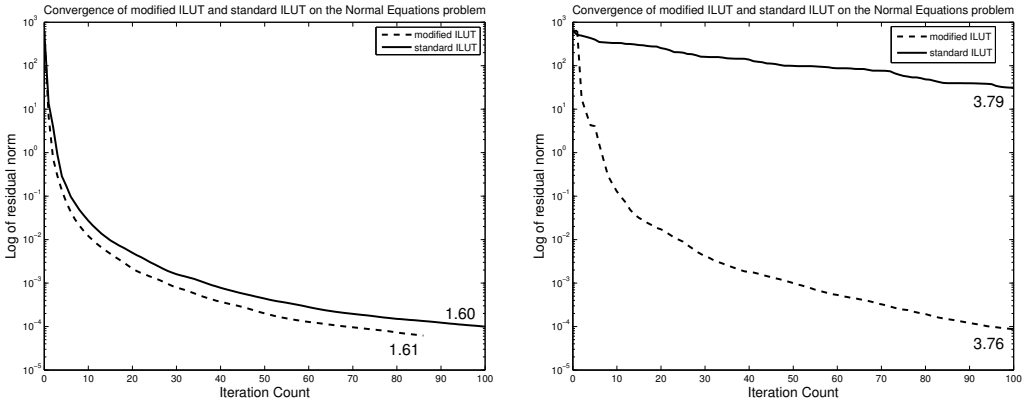
algorithm (Algorithm 2 or 3), we see that the method can be prone to generating dense columns during the factorization. For instance, if the dropping rule is such that very few or no fill-ins are dropped during the elimination of a particular column, then the L and U parts of the column may become dense. This is undesirable because the dense column can very easily be propagated across the remaining columns yet to be eliminated, making them dense as well (leading to large fill-factors). Further, suppose that during the elimination of a subsequent column, a pivot entry is judged small enough and, hence. discarded by the dropping rule. Then the resulting error in the column (and, hence, in the factorization) is a factor of the dense column, which can be significant. As such, the resulting L and U factors could be of poor quality, even though they may be dense (fill-factor may be large).

Modified ILUT with relaxed compensation, on the other hand, is less susceptible to these problems. Adding the compensation, σ, to the diagonal promotes dropping in L, making it less likely to be dense. Furthermore, recall that from Section 3.1.1, the choice of the modification, $\mathbf{z}$, is based on stability considerations. However, it turns out that it may also have some implications on the accuracy of the factorization, particularly when $\mathbf{z}$ is small. The error in the column, induced by dropping the pivot entry, may be reduced by adding the column $\mathbf{z}$ to $\mathbf{l}$, leading to a more accurate factor for the L part of the factorization.

Figure 3.4.1 shows the convergence profiles for standard ILUT and modified ILUT with relaxed compensation for the normal equations problem with $\varrho = -0.5$, on a 100×100 grid. We plot the convergence profiles for the first 100 iterations. As shown in the figure, for a relatively small fill-factor value ($c_F \approx 1.60$), both ILUT and modified ILUT converge quite well (Figure 3.1(a)). However, when the fill-factor is increased ($c_F \approx 3.8$), the performance of ILUT deteriorates (Figure 3.1(b)). Even for relatively similar measures of stability, the convergence of ILUT worsened as the fill-factor increased.

3.4.2 Results for Imaginary Compensation

In what follows, we present some numerical results on the acoustic wave diffraction problem, governed by the Helmholtz equation. We compare the performance of standard ILUT with that of modified ILUT, where the modification is by an imaginary term as discussed earlier in Section 3.2.



(a) Fill-factor, $c_F \approx 1.60$ (b) Fill-factor, $c_F \approx 3.80$

Figure 3.1: Convergence profiles of modified and standard ILUT with relaxed compensation on the normal equations problem with $\varrho = -0.5$, on a 100×100 grid, using different values of the fill-factor

The problem is defined on a 161×361 grid, and the resulting system has size $n = 57960$, with 516600 nonzero entries. We use right-preconditioned restarted GMRES with a restart dimension of 100, and an initial guess $\mathbf{x}^{(0)} = \mathbf{0}$. The maximum number of GMRES iterations is fixed at 500, and we assume convergence when the ℓ_2-norm of the residual is reduced by a relative factor of 10^7. The right-hand side is artificially created by assuming the entries of the solution vector are chosen from the uniform distribution on [0, 1]. We solve the system for increasing values of the wavenumber k, (which corresponds to a decreasing mesh resolution from the highest of 160ppw for $k = 2\pi$ to a low of 10ppw for $k = 32\pi$). For each value of k, we run the problem several times, and the results shown are averaged over the runs.

The fill level parameter for the factorization, p, is fixed at 1000 for both standard ILUT and modified ILUT, and the drop tolerance, τ, is adjusted so that the fill-factor is similar for both methods. As before, the dropping rule for standard ILUT is adjusted so that it is based on the size of the fill-in (update) entry rather than on the pivot entry, in order to yield "good" factors.

From Table 3.4, we observe a remarkable performance for the modified ILUT scheme, compared to standard ILUT. For high values of the wavenumber, the standard ILUT

	k	c_F	iters	total time	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$	$\|\|(A-LU)\mathbf{1}\|\|_2$
ILUT	2π	2.67	135	$23.54s$	$1.92e+03$	10.38
	4π	2.68	156	$31.93s$	$2.06e+03$	10.53
	8π	2.74	198	$32.37s$	$2.94e+03$	11.25
	16π	2.42	> 500	$75.90s$	$1.88e+04$	21.30
	32π	$*$	$*$	$*$	$*$	$*$
MILUT	2π	2.61	114	$21.15s$	$2.38e+03$	18.82
	4π	2.65	121	$22.56s$	$2.61e+03$	18.90
	8π	2.64	134	$20.40s$	$3.32e+03$	22.05
	16π	2.48	206	$31.68s$	$2.44e+03$	37.26
	32π	3.86	182	$34.01s$	$3.03e+02$	36.02

Table 3.4: Results for ILUT and modified ILUT with imaginary modification on the Helmholtz problem with different values of the wavenumber k.

factorization can become unstable. For this set of examples, ILUT requires fewer fill-in elements in order to produce stable factors. However, this implies that the preconditioner is poorly approximated, which makes it difficult for the solver to converge. This is evident for the example with wavenumber, $k = 16\pi$, where the solver fails to converge within the required number of iterations with ILUT as preconditioner. If more fill-in elements are allowed, the resulting factors become unstable, with values of $||(LU)^{-1}\mathbf{1}||_2$ on the order of 10^{100} or larger. This renders the factors useless as preconditioners for any iterative method. Comparing the results for ILUT with those of modified ILUT, we see that modified ILUT is more efficient and robust on the Helmholtz problem. By using imaginary perturbations to modify the diagonal, the resulting factors from the modified ILUT scheme are quite stable and yield good results for the problem, even at high wavenumbers.

For the test problem with wavenumber $k = 32\pi$, ILUT shows no signs of convergence. Even when the fill-ins are reduced so that the fill-factor, c_F, is ≈ 1.0, the value of $||(LU)^{-1}\mathbf{1}||_2$ for the resulting LU factors is of the order of 10^{144}. Modified ILUT, however, produces stable factors and was successful in solving the problem.

In regards to Section 2.2, where imaginary perturbations were used to construct a shifted ILUT preconditioner for the Helmholtz problem, these results agree quite well with the subsequent results in Section 2.2.4. It is worth noting that the approach used in Section 2.2 differs from the optimal strategy discussed in this chapter. The

focus of the approach in Section 2.2 is on improving the quality of the preconditioner by improving the diagonal dominance of the rows of the matrix, prior to performing the ILUT factorization. The perturbation strategy that defines the imaginary shift relies on two heuristics, both aimed at improving diagonal dominance entirely through diagonal compensation. Here, we use a different approach to define the imaginary shift, by solving an optimization problem with stability and accuracy constraints, during the ILUT factorization. The resulting compensation is not only active on the diagonal entry, but also on the entries in the L-part of the corresponding column.

3.5 Conclusion

This chapter describes two procedures for defining modifications of ILU factorizations with threshold-based dropping. The first procedure, based on the column version of ILUT, extends the standard diagonal compensation idea for modified ILUT, by spreading the compensation term over the non-unit diagonal, as well as the nonzero entries in the L part of the column, in an optimal way. The technique is further extended to exploit the use of imaginary shifts for the compensation term. Numerical results show that these modified ILUT methods are robust for both ill-conditioned and indefinite problems. By adding a compensation term to the L part of the column as well as the non-unit diagonal, it appears that the modified ILUT method is less susceptible to the problems that cause instabilities and inaccuracies in the standard ILUT factorization.

Chapter 4

Reordering for ILU

For poorly structured matrices, the ILU factorization could generate significant fill-in elements, making the resulting L and U factors dense and hence inefficient for preconditioning. Reordering techniques, such as reverse Cuthill-McKee (RCM) [39], minimum degree [40–43, 81], and nested dissection [38, 44], have been used to help correct this problem. As noted earlier, standard ILU factorizations can be prone to instability, as a result of the occurrence of unstable pivots during the factorization. Column (or row) pivoting techniques have been typically employed to improve or maintain stability during the factorization [10, 45–47]. These techniques permute the columns (or rows) of the matrix during the factorization, so that columns (or rows) that are likely to yield unstable pivots are moved to the end of the factorization. Since the matrix is sparse, columns or rows near the end of the factorization are less likely to contribute to the elimination of subsequent columns (or rows) during the factorization [70]. Thus moving the columns (or rows) with unstable pivots to the end will have no effect on the stability of the factorization (up to this point), since these pivots may not be used in the factorization.

In recent years, there has been some interest in combining ILU factorizations with multilevel methods to obtain more efficient preconditioners for general sparse linear systems. One such example is the use of ILU(k), in particular ILU(0), as a smoother for the multigrid method [5]. In [36] and [63], Bollhöfer used the idea of column pivoting, in combination with rigorous dropping strategies to construct a multilevel preconditioner

based on the Crout version of ILUT. In [51], Ploeg, Botta and Wubs use a grid dependent reordering scheme to construct an effective multilevel ILU method for elliptic PDEs. An algebraic variant of this was later developed in [53]. Other techniques have combined multilevel graph coarsening strategies with standard threshold-based ILU factorizations to yield robust multilevel preconditioners for general sparse linear systems, see for example [60, 82].

The goal of this chapter is not to explore a new contribution to the class of multilevel ILU techniques. Instead, the focus is to answer the following question: *based on reordering and stability considerations, can we use ideas from multilevel graph coarsening to improve the quality of the ILU factorization?*. In other words, can we incorporate multilevel coarsening ideas within a (single level) ILU framework to yield a robust and stable factorization? Successful results could lead to new and more sophisticated and efficient multilevel ILU techniques.

4.1 Graph Coarsening Strategies

Multi-level methods, such as multigrid [4–6] or Schur-based multi-level techniques [7, 50, 60, 83], rely on graph coarsening strategies to construct the next level matrix in the multi-level process. For multigrid, this corresponds to selecting a subset of the original or fine grid, known as the 'coarse grid', for which the solution to the residual equation $Ae = r$, for some error e and residual r, is slow to converge, see for instance [9]. For multi-level Schur-based methods, such as multi-level ILU, the coarsening strategy may correspond to selecting, from the adjacency graph of the original matrix, a subset of nodes corresponding to an independent set [60], or nodes that are poor in terms of diagonal dominance [82], or nodes that promote growth in the inverse LU factors of the ILU factorization [36, 63].

The general idea of coarsening is quite simple. Given a graph with n vertices, we would like to find a smaller graph, typically of size about $n/2$, which more or less represents well the original graph. Suppose we have an adjacency graph $G = (V, E)$ of some matrix A. For simplicity, we assume that G is undirected. An edge, e_{ij} represents the binary relation $(i, j) : a_{i,j} \neq 0$. Coarsening the graph consists of finding a 'coarse' approximation $(\hat{V}, \hat{E})$ so that $|\hat{V}| < |V|$ and such that the new graph is a faithful

representation of the original graph in some sense. By recursively coarsening some original graph, we obtain a hierarchy of approximations to the original graph.

There are several techniques available for coarsening a graph. Here, we utilize an aggregation-based approach, which is described next. In what follows, we use $adj(i)$ to denote the adjacency (or neighbor) set of node i.

4.1.1 Pairwise Aggregation

Aggregation-based techniques are a common strategy for coarsening; see for instance [5, 7, 84, 85]. The pairwise aggregation strategy seeks to simply coalesce two adjacent nodes in a graph into a single node, based on some measure of nearness. The technique is based on edge collapsing [86], which is a well-known method in the multilevel graph partitioning literature. In this method, the collapsing edges are usually selected using the *maximal matching* method. A *matching* of a graph $G = (V, E)$ is a set of edges $\tilde{E}$, $\tilde{E} \subseteq E$, such that no two edges in $\tilde{E}$ have a vertex in common. A maximal matching is a matching that cannot be augmented by additional edges to obtain another matching. There are a number of ways to find a maximal matching for coarsening a graph. We use a simple greedy strategy similar to the *heavy-edge matching* approach proposed in [87]. The following algorithm gives a general idea of how this matching works.

Algorithm 5: Greedy Matching Algorithm.

1. For $i = 1 : n$, do:
2. $p(i) = 0$ and $max = 0$
3. For $j \in N_i$ and if $|a_{i,j}| \geq max$ do:
3. $p(i) = j$
4. $max = |a_{i,j}|$
5. EndDo
6. EndDo

where $N_i \subseteq adj(i)$ is defined as $N_i = \{j \mid j \in adj(i) \text{ and } j \text{ is unmatched}\}$, a_{ij} represents the weight of edge (i, j), and max keeps track of the maximum of such edge weights for node i. The array $\mathbf{p}$ contains the matched nodes so that node i is matched to node $p(i)$. In algebraic terms, the above greedy matching algorithm simply matches a node

i, with its largest (off-diagonal) neighbor. Notice that this matching is not one-to-one, since more than one node may be matched to a single node.

Once matching is done, each pair of matched nodes $(i, p(i))$ are coalesced into a new coarse node. If $p(i) = p(k)$ for some other node $k \neq i$, then node k is left as a singleton. When two nodes i, and j are coalesced into a new one, the new node is considered the parent of the two nodes i and j, and denoted $par(i, j)$. Singletons represent themselves as parents. Figure 4.1 gives an illustration of a single coarsening step.

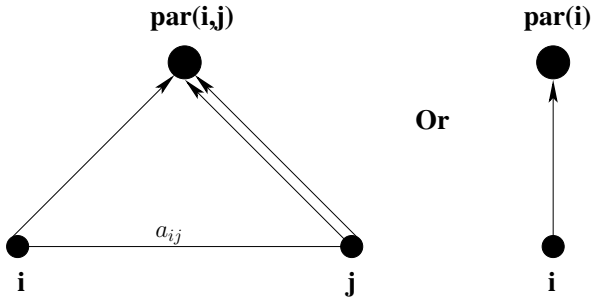


Figure 4.1: The coarsening process. Left: i and j are coalesced into node j; Right: singleton i is mapped into itself.

As in standard agglomeration techniques, the parent node is representative of one of the nodes that combine to form the parent. As shown in the figure, nodes i and j are coalesced into node j, which means node j will represent both nodes i and j in the coarse level graph. This leads to a C/F splitting of the nodes so that the coarse set, C, contains the nodes $k : par(i, j) = k$, for i, $j \in V$ and $k = i$ or j, and the fine set is defined as $F = V \setminus C$.

4.1.2 Multi-level Coarsening Scheme

The coarsening strategy can be expanded into a multi-level framework by repeating the process described above on the graph associated with the nodes in the coarse set. Let us define G_0 to be the original graph G and let $G_1, G_2, \ldots, G_m$ be a sequence of coarse graphs such that G_ℓ is obtained by coarsening on $G_{\ell-1}$ for $0 \leq \ell \leq m$. Let $A_0 = A$ and A_ℓ be the adjacency matrix associated with graph G_ℓ. As previously mentioned, G_ℓ admits a splitting into coarse nodes, C_ℓ, and fine nodes, F_ℓ, so that the matrix A_ℓ

can be reordered in one of the following ways:

$$P_\ell A_\ell P_\ell^T = \begin{bmatrix} A_{C_\ell C_\ell} & A_{C_\ell F_\ell} \\ A_{F_\ell C_\ell} & A_{F_\ell F_\ell} \end{bmatrix} \tag{4.1}$$

or,

$$P_\ell A_\ell P_\ell^T = \begin{bmatrix} A_{F_\ell F_\ell} & A_{F_\ell C_\ell} \\ A_{C_\ell F_\ell} & A_{C_\ell C_\ell} \end{bmatrix} \tag{4.2}$$

where P_ℓ is some permutation matrix. We refer to the first approach as the *Bottom-Up approach*, since we keep putting the next-level nodes (i.e. coarsening) in the upper-left block. The second approach is referred to as the *Top-Down approach* in which the next-level nodes are put in the lower-right block. In both schemes, the next-level matrix is defined as $A_{\ell+1} = A_{C_\ell C_\ell}$ and its adjacency graph, $G_{\ell+1}$, is constructed from the coarse nodes C_ℓ, associated with the graph G_ℓ.

To construct $G_{\ell+1}$, we need to create edges, along with edge-weights, in certain ways between two coarse nodes created during the coarsening process. There are several ways of achieving this. One way is to set two coarse nodes to be adjacent in $G_{\ell+1}$ if they are adjacent in G_ℓ. In other words, suppose $l = par(u, v)$ for some nodes u and v in G_ℓ, then $adj(l)$ in $G_{\ell+1}$ is simply:

$$\{par(l, k) \mid k \in adj(l) \text{ in } G_\ell\}.$$

Note that here, l is either u or v, whichever is the coarse node in G_ℓ. This may be considered as a *one-sided* approach, since the fine nodes in F_ℓ that are matched with the coarse nodes in C_ℓ, do not contribute edges to the adjacency graph of their parents. The edge-weights for the new graph, $G_{\ell+1}$, using this approach, could be taken directly from the entries of the adjacency matrix of the coarse nodes in the previous graph, G_ℓ. In this case, the permutation matrices, P_ℓ, in Equations 4.1 and 4.2 are simply made up of columns of the identity matrix. Although this way of creating edges in the new coarse graph is cheap and easy to implement, it could lead to coarse graphs that are poor approximations of the original graph. Since coarsening may also be seen as a way of 'breaking' connections in the fine level graph, it is easy to see how this one-sided approach could lead to a graph of singletons, after a few coarsening steps.

A more general alternative to this one-sided approach is to define two coarse nodes to be adjacent in $G_{\ell+1}$ if they are parents of adjacent nodes in G_ℓ. In other words,

suppose $z = par(x, y)$ for some nodes x and y in G_ℓ, then $adj(z)$ in $G_{\ell+1}$ is given by:

$$\{par(x,k) \mid k \in adj(x) \text{ in } G_\ell\} \bigcup \{par(y,k) \mid k \in adj(y) \text{ in } G_\ell\}$$

This way of defining edges in the coarse graph is quite common in multilevel graph partitioning techniques, see for instance [87], and other aggregation-based methods, see for instance [5, 85]. It may be considered as a *two-sided* approach, since, unlike the previous approach, the fine nodes in F_ℓ do contribute edges to the adjacency graph of their parents. Figure 4.2 gives an illustration of the approach. The edge-weights of the new coarse graph is typically defined as the sum of the weights of the edges in the fine graph that connects their children, or some weighted average of this sum, see for instance [5, 84, 85].

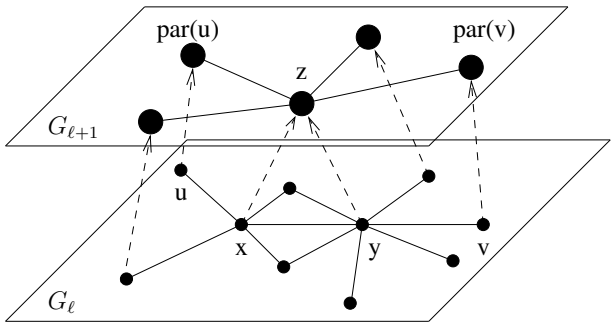


Figure 4.2: Next-level coarsened graph

4.2 Preselection Techniques

The above coarsening strategy is essentially based on a 'strength of connection' idea. That is, each node is compared to its largest off-diagonal neighbor (or most strongly connected neighbor) with respect to some defined weights. The choice of which candidate nodes go into the coarse or fine set during the coarsening process depends on their corresponding values in some weight vector, $\mathbf{w}$. These weights are chosen to capture some intrinsic features of the matrix at the current level, or its adjacency graph. Here, we focus on capturing diagonal dominance, since we wish to apply some form of incomplete factorization on the resulting reordered matrix. The goal is to obtain a reordering

of the matrix such that rows (or columns) corresponding to nodes that satisfy some diagonal dominance criterion are ordered first (in the top-left corner of Equations 4.1 and 4.2). Next, we present three different strategies for selecting candidates for coarsening, based on the notion of diagonal dominance.

4.2.1 Strong Neighbor Connection Ratio

In this approach, weights are defined based on the relative size of the diagonal entry and its corresponding largest off-diagonal neighbor. We define the term γ_i as:

$$\gamma_i = \frac{|a_{ijmax}|}{|a_{ii}| + |a_{ijmax}|},$$

where $jmax = k \in adj(i) : |a_{ik}| > |a_{ij}|, \forall j \in adj(i)$, and $j \neq k$. Note that the matched pair $(i, jmax)$ will be coalesced together during the coarsening process. One can think of γ_i as a measure of the degree of dependence of node i on node $jmax$. If γ_i is large (close to 1), then node i strongly depends on node $jmax$. On the other hand, if γ_i is small, then the dependence of node i on node $jmax$ is only a weak one. This notion of dependence is similar to the 'strength of connection' idea, which is a common practice for defining coarse grid operators in algebraic multigrid, see for instance [5–9]. The weights, w_i, are then formally defined as:

$$w_i = \gamma_i \times (1 + |adj(i)|)$$

The quantity $(1 + |adj(i)|)$ represents the number of non-zero entries in row i of the adjacency matrix (including the diagonal entry). It is used here to scale γ_i so that nodes with fewer non-zeros have a smaller entry in $\mathbf{w}$, and hence are likely to be ordered first (in the reordered system). This serves to exploit structure in the reordering to reduce the amount of fill-ins generated during the factorization of the reordered system.

The decision as to which node from the matched pair, $(i, jmax)$, goes into the coarse or fine set, depends on the relative size of their weights, w_i and w_{jmax}, and on the direction of coarsening. For the bottom-up approach (see Equation 4.1), at level ℓ of the multilevel process, if node i strongly depends on node $jmax$, then node $jmax$ could 'represent' node i in the next level coarse graph. Hence $jmax$ will be a good candidate for the coarse set, C_ℓ. The opposite is true for the top-down approach (see

Equation 4.2). If nodes i and $jmax$ identically depend on each other (i.e. $w_i = w_{jmax}$), then either one is put in C_ℓ and the other in F_ℓ.

4.2.2 ILU(0)

Another way of defining the entries of the vector $\mathbf{w}$ used in the coarsening process is based on the incomplete LU factorization with zero fill-in elements, ILU(0) [10]. Recall from Section 2.1 that for some matrix, A, with regular structure, the ILU(0) factorization may be written in the form

$$M = (D - E)D^{-1}(D - F),$$

where $-F$ is the strict upper part of A, $-E$ is the strict lower part of A, and D is some diagonal matrix which is different from the diagonal of A in general. The entries of the diagonal matrix D, are obtained from the recursive formulation of the factorization given as Equation 2.1.

Rewriting M as $M = LU$, we have that $L = I - ED^{-1}$ and $U = D - F$. Since D satisfies the (non-unit) diagonal of the factorization, the size of the entries of D could give an indication of which nodes are good candidates for the coarse set, and which are better suited for the fine set. In order to be consistent with the previous approach, we define γ_i as:

$$\gamma_i = \frac{1}{|d_i|},$$

where d_i is the i-th diagonal entry of D. Then weights, w_i, can be computed as before: $w_i = \gamma_i \times (1 + |adj(i)|)$.

This approach differs somewhat from the previous approach, where the weights are defined with respect to the entries of the original matrix A. Here, we look instead at the entries of the diagonal of U from the factorization of A. Notice that from Equation 2.1, the k-th entry of the term $\mathbf{e}_i^T D_{i-1}^{-1}\mathbf{f}_i$ satisfies the k-th pivot element used in the elimination of row i, during the factorization process. In the special case where A is derived from a structured finite difference approximation to a second-order difference operator, $\mathbf{e}_i^T D_{i-1}^{-1}\mathbf{f}_i$ satisfies the exact pivot entries used in the elimination of the entries in row i, since no perturbation occurs from fill-in on these entries. If the pivot entry is large, then it is likely that $d_k \to 0$. In the top-down approach, this means node k will

be a good candidate for the coarse set. In other words, nodes that are likely to yield unstable pivots are moved into the coarse set. In the bottom-up approach, the opposite is true. Nodes that are stable (large d_k) will be good candidates for the coarse set.

4.2.3 Relaxation

Yet another option for computing the components of $\mathbf{w}$ for coarsening is to use relaxations in a way similar to multigrid. Relaxations are a common ingredient used in multigrid methods to damp out oscillatory errors in the residual equation. More recently, a technique known as compatible relaxation [88–92] has been used to select candidates for the coarse grid in algebraic multigrid (AMG). In this approach, relaxations are done on the homogeneous equation $A_\ell e_\ell = 0$, where e_ℓ is initially chosen to be in the near null space of A_ℓ. After a few relaxation sweeps, nodes that are quick to converge are put in F_ℓ. The remaining nodes (that are slow to converge) are judged good candidates for the coarse set and put in some set U_ℓ. An independent subset of these candidates are then selected and put in C_ℓ, and the process is repeated on the remaining candidates, $U_\ell \setminus C_\ell$, until a C/F splitting of the nodes are obtained. Notice that nodes in the F_ℓ set represent rows that are 'good' in terms of diagonal dominance with respect to the associated adjacency matrix A_ℓ.

In using relaxations to form $\mathbf{w}$, we follow an approach similar to the compatible relaxation technique described above. We define γ as the solution to $A\gamma = 0$, obtained after a few relaxation sweeps. Following [89], we then define the components of $\mathbf{w}$ as:

$$w_i = \frac{|\gamma_i|}{\|\gamma\|_\infty}.$$

$\mathbf{w}$ now contains a measure of the relative convergence of all the nodes, with respect to the slowest converging node. For the top-down approach, if w_i is small (close to zero), then node i is put in F, otherwise node i is a good candidate for the coarse set. The opposite is true for the bottom-up approach.

Recall that relaxation schemes, such as Jacobi or Gauss-Siedel, are not guaranteed to converge if the matrix A is not strictly or irreducibly diagonally dominant [10]. Thus, they tend to be ineffective on numerically challenging problems, such as ones that are highly indefinite or ill-conditioned. For such problems, it is possible to transform the system matrix into one that can be handled by the relaxation scheme. We transform the

original matrix A into a new matrix L_A, that represents some graph Laplacian derived from A. Compatible relaxation schemes have been successfully used on graph Laplacian matrices as a means of defining algebraic distances between two nodes in some network graph, see for instance [93, 94]. We define the edge weights of the graph Laplacian for the matrix, A, to be based on the neighbor connection ratio as follows:

$$l_{ij} = -\frac{|a_{ij}|}{|a_{ij}| + |a_{ii}|}$$

Here, the $l_{i,j}$ represent the off-diagonal entries of the matrix L_A. This formulation represents a mapping of the edge weights onto the interval $(0, 1)$, so that entries that are small (in magnitude) in A, remain small (close to 0) in L_A, and vice versa. It is clear that if $|a_{ij}| > |a_{ii}|$, then $l_{ij} > 1/2$, else if $|a_{ij}| < |a_{ii}|$, then $l_{ij} < 1/2$. Thus the degree of dependence of node i on node j is realized in L_A as well. To complete the formation of L_A, the diagonal entries are then defined as:

$$l_{ii} = \sum_{j=1}^{n} |l_{ij}|$$

We now have a more general approach to obtain the entries in $\mathbf{w}$ via relaxations, for an arbitrary matrix. That is, instead of doing relaxations on the homogeneous equation $A\gamma = 0$, we do it on the equation $L_A\gamma = 0$. Compatible relaxation in AMG relies on the fact that the matrix, A, is non-singular, and hence the solution to the homogeneous equation converges to zero (when it converges). Notice however, that L_A is by definition, singular, and hence $L_A\gamma = 0$ does not converge to zero in general. In [93], Chen and Safro have detailed several properties about the convergence of several classical relaxations schemes applied to solve the homogeneous equation involving the graph Laplacian. Suppose the matrix L_A admits a splitting of the form $L_A = M - N$, for some matrices M and N. Then the homogeneous equation $L_A\gamma = 0$ can be re-written in the form $M\gamma = N\gamma$, yielding the iterative process:

$$\gamma^{k+1} = H\gamma^k \tag{4.3}$$

where $H = M^{-1}N$ is the iteration matrix. For the Jacobi relaxation method, the iteration matrix has the form:

$$H_{jac} = D^{-1}(E + F)$$

and for the successive over-relaxation (SOR) method, it takes for form:

$$H_{sor} = (\frac{D}{\omega} - E)^{-1}((\frac{1}{\omega} - 1)D + F),$$

where $-F$ is the strict upper part of L_A, $-E$ is the strict lower part of L_A, and D is the diagonal of L_A so that $L_A = D - E - F$, and ω is the relaxation parameter for SOR. Note that $\omega = 1$ gives the Gauss-Seidel relaxation method. Assuming the graph associated with L_A is connected (and not bi-partite), then H is convergent and Equation 4.3 converges [95]. In this case, the theory in [93] shows that $\gamma^k \to \mathbf{0}$ if the initial iterate, γ^0, is in range$(I-H)$, otherwise $\gamma^k \to \mathbf{1}$. Thus, by choosing an appropriate initial guess, the compatible relaxation scheme may be used on the graph Laplacian in a way similar to what has been described above, to define the weights $\mathbf{w}$.

4.2.4 Candidate set selection

In general, all the nodes in the current graph can be considered candidates for coarsening. However, in practice, not all the nodes need to be considered. Nodes that have very small entries in $\mathbf{w}$ can be considered good enough to produce stable pivots during the ILU factorization, and hence can be immediately put in the set constituting the top-left block of the reordered system (according to the top-down or bottom-up scheme). During the preselection phase of the coarsening scheme, a node i, is selected as a candidate for coarsening if w_i is large, relative to some parameter θ. Thus, changing the size of the parameter θ will determine how many nodes are used for coarsening to construct the next level coarse graph.

4.3 The multi-level coarsening algorithm

Next we give a complete algorithm for the multilevel coarsening strategy used to reorder the system before performing the ILU factorization. The algorithm will describe the bottom-up approach, as the top-down approach follows a similar framework. The algorithm only gives a high-level description of the coarsening scheme, in order to keep it simple and more general.

Algorithm 6: Multilevel Coarsening Algorithm.

1. For $\ell = 0 : nlev$, do:
2. Compute $\mathbf{w}$ and form candidate set U_ℓ, for coarsening
3. Initialize $C_\ell = V_\ell \ \setminus \ U_\ell$
4. Initialize $F = \emptyset$
5. Obtain a C/F splitting of U_ℓ to get I_C and I_F
6. Set $C_\ell = C_\ell \bigcup I_C$
7. Set $F_\ell = F_\ell \bigcup I_F$
8. Build next level graph $G_{\ell+1}$
9. EndDo

Algorithm 6 performs $nlev + 1$ levels of coarsening, starting with the original fine graph G_0. In line 2 of the algorithm, $\mathbf{w}$ is computed and a candidate set, U, is extracted for coarsening. The nodes that do not belong to U are assigned to the coarse set, C (line 3), as they represent 'good' nodes that will be ordered first in the final reordering. A C/F splitting of nodes in U is done to obtain coarse nodes, I_C, and fine nodes, I_F, which are used to update C and F (in lines 6 and 7). Finally, the next level graph is assembled using either the one-sided approach or the two-sided approach for defining the edge weights, discussed earlier in section 4.1.2. The overall quality of the coarsening scheme is determined, in part, by the strategy used to obtain the C/F splitting in line 5. A common approach is to use a strategy that promotes independence among the nodes in the coarse set. Another approach, common in the multilevel ILU literature, focuses on promoting diagonal dominance among the nodes in the coarse set. The approach followed in this thesis uses a pairwise aggregation technique to obtain a splitting so that the nodes in the coarse set are diagonally dominant and *nearly* independent. Algorithm 7 formally describes the technique used to obtain the C/F splitting in line 5 of Algorithm 6.

Algorithm 7: Coarse step (C/F splitting) Algorithm.

1. Do matching on $G_\ell = (V_\ell, E_\ell)$ to obtain pairs $(i, p(i))$
2. Set $mark(i) = 0$, $\forall i \in U_\ell$
3. For $i \in U_\ell$, do:
4. Set $j = p(i)$
5. If $mark(i) == 0$ and $mark(j) == 0$ and $i \neq j$, then

6. If $w_i \leq w_j$, then
7. Set $i \in I_C$ and $j \in I_F$
8. Else
9. Set $i \in I_F$ and $j \in I_C$
10. Set $mark(i) = mark(j) = 1$
11. EndIf
12. EndDo
13. Assign any remaining nodes to I_C or I_F

The algorithm begins by doing a matching (see Algorithm 5) on the current level graph G_ℓ to obtain the pairs $(i, p(i))$, which will be used to split the candidate set, U_ℓ. This matching promotes independence among the nodes in the coarse (or fine) set. Furthermore, it also dictates the structure of the resulting reordered matrix, which in turn affects the fill-in pattern of the ILU factorization applied to the matrix. Initially, all the nodes in U_ℓ are unmarked. A pair of unmarked nodes, i and j, that are matched, are compared to each other based on the relative size of their entries in $\mathbf{w}$, and assigned to the coarse set, I_C, or the fine set, I_F. These nodes are then marked, and the process is repeated until all matched nodes in U_ℓ, that are unmarked, have been marked and assigned to I_C or I_F. Finally, to complete the splitting, any remaining nodes that are not marked are assigned to the coarse or fine set. These nodes come from two different contributions. The first consists of nodes that are matched to themselves in the original graph (i.e. singletons). These nodes represent diagonal entries in the associated matrix, and hence will be good candidates for the coarse set. The other contribution comes from nodes that are matched with neighbors that have been previously marked. Recall that the matching described in Algorithm 5 of section 4.1.1 is not one-to-one. Hence a node, whose matched neighbor is involved in another matching, could be skipped by the check in line 5 of Algorithm 7. These nodes, which now behave like singletons, can be simply assigned to the coarse set to be dealt with at the next level of coarsening. Alternatively, they may be assigned to I_C or I_F depending on the size of their entry in $\mathbf{w}$, relative to that of their matched neighbor (albeit this neighbor is already marked). In either case, the notion of independence is compromised, as the unmarked nodes may be assigned to the same set as their matched neighbor. In the former case, independence could be

recaptured at the next level as the matched pair of nodes remain neighbors at the next level. In the latter case, we obtain a C/F splitting that adheres more strictly to the requirement that 'good' nodes be assigned to the coarse set. The result is a trade-off between the diagonal dominance of the nodes in the coarse set, and the independence of the nodes in the coarse (or fine) set (and hence the structure of the resulting reordered matrix).

4.4 Final reordering and ILU factorization

4.4.1 Final reordering

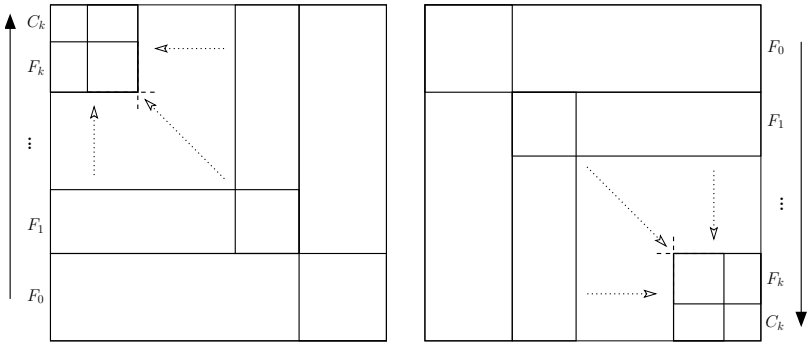
Using Algorithm 6, after each step of coarsening we obtain a C/F splitting of the current graph G_ℓ, in the sets C_ℓ and F_ℓ. For the bottom-up approach discussed above, after k steps of coarsening, we obtain the following sequence of sets: $C_k, F_k, \ldots, F_0$. This sequence represents the final reordering of the system matrix. Nodes in C_k will be ordered first, since they represent the 'best' nodes selected by the coarsening strategy. This will be followed by nodes in F_k, then F_{k-1} and so on, until F_0, which contains the nodes adjudged to be 'worst' by the coarsening strategy. Considering each of these sets as a single level, we obtain $k+1$ levels in the final reordered matrix.

For the top-down approach, the order of the sequence is reversed, since coarsening is done on the 'worst' nodes. Thus, nodes in F_0 represent the 'best' nodes and will be ordered first, and nodes in C_k will be ordered last. Figure 4.3 shows an illustration of the final reordering for the different coarsening approaches.

4.4.2 Adaptation to ILU factorization

Once the matrix has been reordered, an incomplete LU factorization is performed. Here, we consider the column-based implementation of two ILU methods: the incomplete LU factorization with level of fill dropping (ILU(k)); and the incomplete LU factorization with dual threshold dropping (ILUT), described earlier in Section 1.3.1 (see Algorithm 3).

The standard ILUT algorithm can be modified to take advantage of the reordering derived from the coarsening strategies discussed above. One simple approach is to use



(a) Bottom-up approach (b) Top-down approach

Figure 4.3: Final reordering of the original matrix after k steps of coarsening using the bottom-up approach (left) and top-down approach (right).

dropping strategies that are local to each level of the reordered matrix. In other words, if the coarsening strategy yields $\ell + 1$ levels for the reordered matrix, then the drop tolerance, τ, and the fill-level parameter, p, will each take the form of a vector of length $\ell + 1$, so that $\tau(0)$ and $p(0)$ dictate the dropping rules for the factorization of the level-zero block of columns, and so on. An illustration of the elimination process is shown in Figure 4.4, and a sketch of the general structure of the algorithm is given next as Algorithm 8. Here, we assume that **lev** is a vector of length $\ell + 1$, containing the block size (number of columns) for each level.

Algorithm 8: Left-looking or JKI ordered ILUT.

0. Set $bsize = \mathbf{lev}(1)$
1. Set $\ell = 1$
2. For $j = 1 : n$, do:
3. $\quad \mathbf{w} = A_{1:n,j}$
4. $\quad$ If $j > bsize$
5. $\quad\quad \ell = \ell + 1$
6. $\quad\quad bsize = bsize + \mathbf{lev}(\ell)$
7. $\quad$ EndIf
8. $\quad$ For $k = 1 : j - 1$ and if $w_k \neq 0$, do:
9. $\quad\quad$ Apply first dropping rule to w_k using $\tau(\ell)$

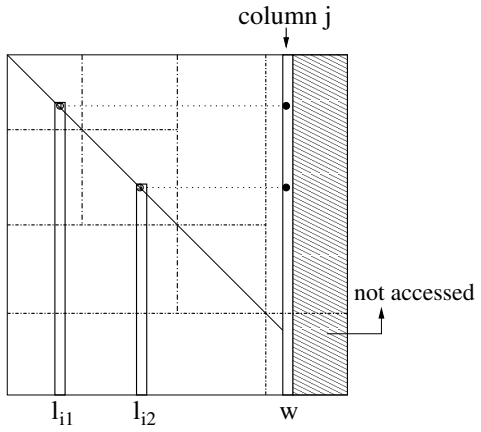


Figure 4.4: Column-based ILUT on reordered matrix with four levels of coarsening (bottom-up scheme).

10. If w_k is not dropped, $\mathbf{w}_{k+1:n} = \mathbf{w}_{k+1:n} - w_k \cdot L_{k+1:n,k}$
11. Enddo
12. For $i = j+1, \ldots, n$, $l_{i,j} = w_i/w_j$ $(l_{j,j} = 1)$
13. Apply second dropping rule to $L_{j+1:n,j}$ using $\tau(\ell)$ and $p(\ell)$
14. For $i = 1, \ldots, j$, $u_{i,j} = w_i$
15. Apply second dropping rule to $U_{1:j-1,j}$ using $\tau(\ell)$ and $p(\ell)$
16. Enddo

The algorithm begins by setting the block size parameter, *bsize*, to the size of the first level block. This block contains the columns (or rows) associated with the nodes adjudged to be the 'best' according to the coarsening strategy. As in Algorithm 3, at the start of a given step j, a working column, $\mathbf{w}$, is created and set to the initial j-th column of A, $\mathbf{a}_j$, on which elimination operations will be performed. The index j is then compared to *bsize* to determine which parameters are to be used for the elimination. The level counter ℓ, and *bsize* are then updated accordingly (lines 5 and 6). The elimination process progresses similarly to Algorithm 3, using $\tau(\ell)$ and $p(\ell)$ to determine the dropping criteria for each level ℓ.

Although the dual truncation property of ILUT makes it generally more robust than the level-based ILU(k), it is prone to yielding unstable factors, where $\|L^{-1}\|$ (and or $\|U^{-1}\|$) is very large. Here, we explore the possibility of using reordering by coarsening

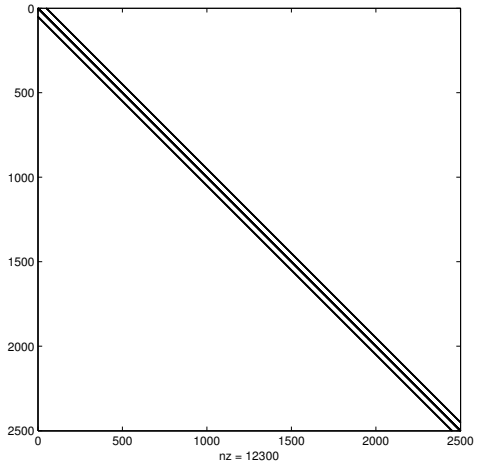
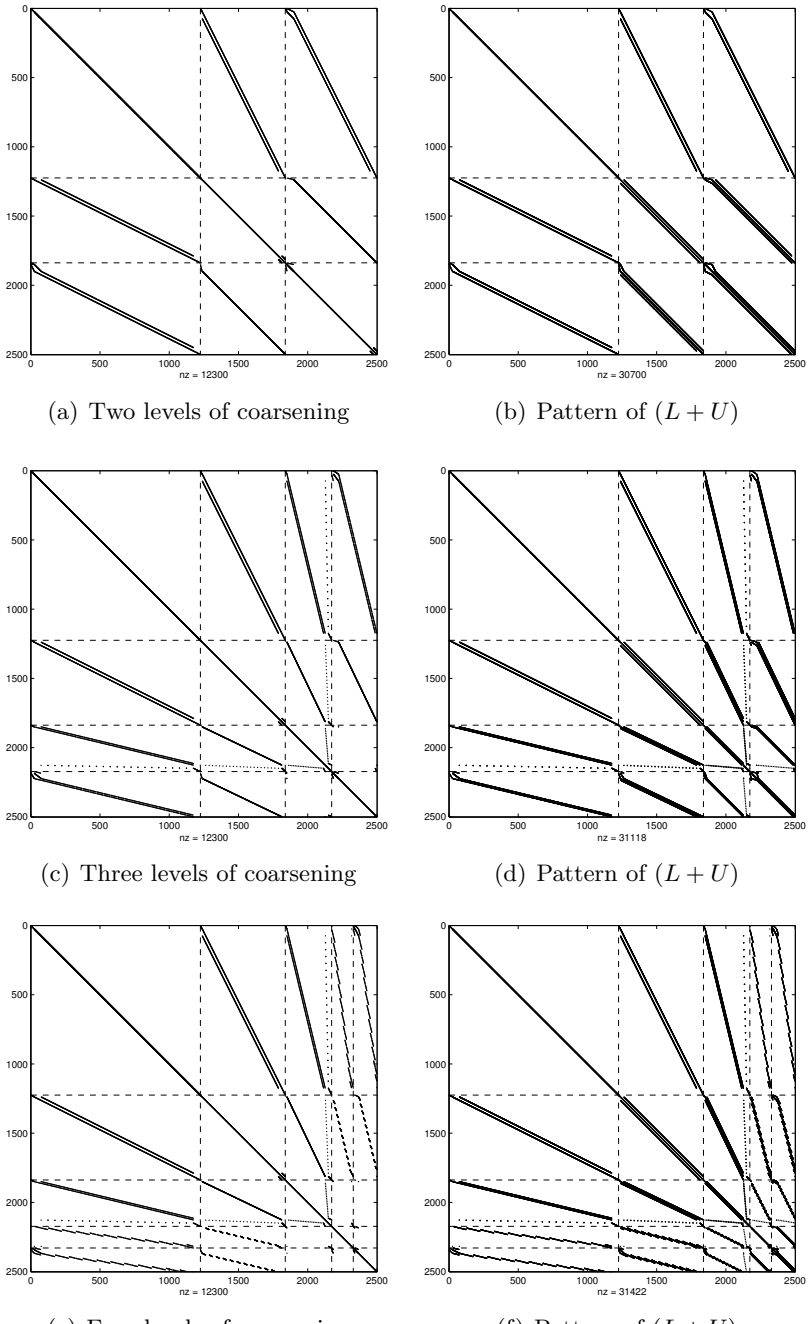


Figure 4.5: Nonzero pattern of the 2D Laplacian matrix.

to improve the stability of the factors, or prevent them from getting worse. By adopting a level-based dropping strategy, we have better control of how to drop small terms during the factorization at the different levels. Like standard ILUT, from a practical standpoint, it is often better to keep $p(\ell)$ large and vary $\tau(\ell)$ to control the dropping. One simple approach is to choose $\tau(1)$ to be relatively large for the first block, and (progressively) smaller for the subsequent blocks during the factorization. This makes sense, since the first block (or first set of blocks) contains nodes that are considered to be 'good' in terms of diagonal dominance. Thus dropping during the factorization of the columns associated with these nodes could be done in a more aggressive manner. At later levels, the columns are poor in terms of diagonal dominance, and a more accurate factorization may be needed to yield factors that better approximate the original matrix.

A similar approach may be used for the level-based ILU(k) method. During the factorization, the level-of-fill parameter k, can be (progressively) increased at each subsequent level from the first level. Like the ILUT method, this offers better control of how terms are dropped during the factorization. However, note that dropping in ILUT is quite different from dropping in ILU(k); nonetheless, both can be adapted to take advantage of the level structure of the reordered matrix.

Figure 4.5 shows the nonzero pattern of the 2D Laplacian matrix originating from the finite difference discretization of the Laplace operator on a 50×50 rectangular



(a) Two levels of coarsening (b) Pattern of $(L+U)$

(c) Three levels of coarsening (d) Pattern of $(L+U)$

(e) Four levels of coarsening (f) Pattern of $(L+U)$

Figure 4.6: An example of the nonzero pattern of the reordered matrix and the resulting ILU factorization for the 2D Laplacian.

grid, with homogeneous Dirichlet boundary conditions. To illustrate the effect of the reordering strategy, figure 4.6 shows the nonzero pattern of the same matrix reordered by the top-down version of the multilevel graph coarsening scheme. The figure depicts an example of the pattern of the reordered matrix, and the corresponding pattern of $(L + U)$ from the resulting ILU factorization, after 3, 4, and 5 levels of coarsening. The fill-in resulting from the ILU factorization for each of the reordered matrices is approximately 2.5.

4.5 Numerical Results

In what follows, we present some numerical results that compare the performance of different reordering strategies on examples from the Helmholtz equation application. We test the effect of reordering by the multilevel graph coarsening strategy described above, on the Helmholtz problem, and compare its performance against that of standard reordering techniques for ILU. In particular, we compare results against that of the reverse Cuthill-McKee algorithm (RCM), the multiple minimum degree algorithm (MMD), and the nested dissection algorithm (ND). We use the version of these algorithms as implemented in METIS [96]. For completeness, we also include results for standard ILUT using the natural (row-based) ordering obtained from the discretization. The results presented for the multilevel graph coarsening reordering technique uses the strong neighbor connection ratio technique to define the coarsening strategy. The direction of coarsening is top-down, and the two-sided scheme is used to determine the weights of the graph at the next level. We allow a maximum of 5 levels of coarsening, so that the cost of constructing the preconditioner is similar to that of the other reordering strategies.

The physical description of the problem is identical to the problem defined in Section 3.4.2. The resulting problem has size $n = 57,960$, with $nnz = 516,600$ nonzero elements. We use right-preconditioned restarted GMRES with a restart dimension of 100. The maximum number of GMRES iterations is fixed at 500, and we assume convergence when the ℓ_2-norm of the residual is reduced by a relative factor of 10^7. The right-hand side vector is artificially generated by assuming a solution of all ones.

First, we compare the performance of the different ordering schemes on the Helmholtz

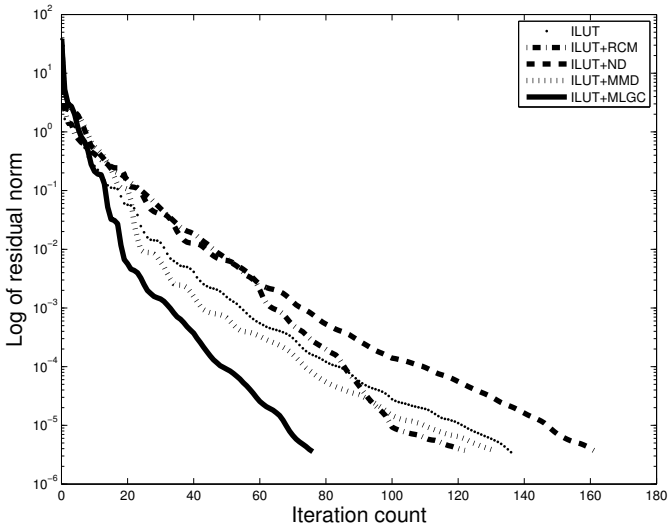


Figure 4.7: ILUT with different orderings on Helmholtz problem.

problem with wavenumber $k = 2\pi$. The problem is slightly indefinite with a mesh resolution of 160 points per wavelength. Figure 4.7 shows the convergence profiles for standard ILUT with the natural ordering, and ILUT with RCM, ND, MMD, and the multilevel graph coarsening (MLGC) orderings. The memory usage (or fill-factor), for each of the preconditioners, is ≈ 2.5. As shown in the figure, the MLGC ordering yields the best convergence performance. It converges after 77 iterations, followed by RCM (122 iterations), then MMD (131 iterations), then the natural ordering (136 iterations), and then finally ND (161 iterations). This results suggests that the performance of ILUT on the Helmholtz problem can be significantly improved by reordering the system matrix by the MLGC scheme, prior to performing the ILUT factorization. To see whether this holds true for higher values of the wavenumber, we apply the approach to solve a number of systems corresponding to increasing values of the wavenumber, as in Section 3.4.2.

Recall that as the wavenumber increases, the Helmholtz problem becomes more indefinite, and hence more challenging to solve by standard ILUT. As shown in previous chapters, standard ILUT applied to this problem fails to converge at high wave numbers. As a result, each preconditioner, in the results that follow, is obtained by performing shifted ILUT on the (reordered) matrix of the discretized problem. The shifted ILUT

method used here is based on the τ-based scheme (i.e. shifting based on the drop tolerance) described earlier in Section 2.2.3.

Table 4.1 presents results for solving the linear system resulting from the Helmholtz problem, for different wavenumbers k. The table shows results for the total number of iterations, and the total time taken to construct the preconditioner and solve the linear system. The memory usage (or fill-factor, c_F) is kept at ≈ 3.0 for each preconditioner, to allow for a fair comparison.

Preconditioner	k	$\frac{\lambda}{h}$	No. iters	c_F	total time
Shifted ILUT	4π	80	146	3.00	$6.24s$
	8π	40	136	3.01	$5.90s$
	16π	20	142	3.00	$6.47s$
	32π	10	106	3.00	$4.92s$
Shifted ILUT + RCM	4π	80	166	3.01	$7.13s$
	8π	40	154	3.02	$6.65s$
	16π	20	119	3.00	$5.59s$
	32π	10	87	3.01	$4.33s$
Shifted ILUT + ND	4π	80	180	3.01	$8.51s$
	8π	40	159	3.02	$7.11s$
	16π	20	131	3.00	$6.42s$
	32π	10	131	3.00	$6.07s$
Shifted ILUT + MMD	4π	80	159	3.00	$7.06s$
	8π	40	140	3.01	$6.31s$
	16π	20	125	3.01	$6.42s$
	32π	10	138	3.00	$6.35s$
Shifted ILUT + MLGC	4π	80	84	3.00	$4.03s$
	8π	40	60	2.99	$2.92s$
	16π	20	57	3.00	$2.91s$
	32π	10	79	2.99	$4.05s$

Table 4.1: Shifted ILUT with different reorderings on the Helholtz problem.

The results from the table indicate that at lower values of the wavenumber, shifted ILUT with natural ordering performs better compared to shifted ILUT with RCM, ND, or MMD ordering. At higher wavenumbers, when the system is very indefinite, the RCM ordering appears to perform better compared to the ND, MMD, and natural orderings. However, shifted ILUT with the MLGC ordering is by far the winner here. It exhibits

superior performance over the other ordering strategies for all the different values of the wavenumber. It requires fewer number of iterations to converge, and converges in very little time compared to the rest. We note here that the time to compute each preconditioner (including the time to perform the reordering) is similar (under 1 second) for the different methods, and hence the total time presented in the table gives a good indication of the solution time, and correlates with the number of iterations required for convergence.

As in the earlier result in Figure 4.7, we see a significant improvement in performance over standard (shifted) ILUT, when the MLGC scheme is used to reorder the linear system. A possible explanation for this comes from considering the effect of the MLGC scheme on the stability and accuracy of the ILUT factorization. First, recall that the coarsening strategy compares a node to its largest neighbor (based on some weight), and assigns one to the coarse set and the other to the fine set. The result is that any long recurrences during the preconditioner solve (i.e. forward solve with L and backward solve with U) involving large (neighboring) entries in L and U are avoided. In other words, the norm of L^{-1} or U^{-1} is not too large, leading to more stable triangular solves. A theoretical basis of this result for the symmetric positive definite case is presented in [97]. Furthermore, the splitting of nodes into coarse or fine sets leads to a hierarchy of (block) independent sets, which can be eliminated with very little fill-ins during the factorization. As a result, few entries are dropped during the factorization, which suggests that the preconditioner obtained from the ILUT factorization with this ordering, should be more accurate. Second, since nodes that are poor in terms of diagonal dominance, (or nodes with unfavorable pivots), are relegated to be ordered last, the adverse effects of these poor nodes on the factorization is minimized. The result is that, large entries in L or U, which could lead to unstable and inaccurate factors, are avoided. Unlike the MLGC scheme, the other reordering techniques (i.e. RCM, ND, and MMD) are based primarily on graph theory, and do not consider the algebraic values associated with the system matrix. Thus, when the natural ordering gives an ordering that is good in terms of minimizing fill-ins, it may not be beneficial to reorder the system by an approach that is based solely on graph theoretic principles. This is in agreement with the results in [97], where a weighted nested dissection (WND) approach is used to reorder an approximate inverse preconditioner. The results indicate

that the WND approach yields a more effective preconditioner for anisotropic problems, than the unweighted version. In [98], D'Azevedo-et-al present a minimum discarded fill (MDF) algorithm that utilizes the algebraic values of the system matrix, to construct a reordering for ILU. Their results also indicate that the algebraic-based ordering yields a more effective preconditioner for anisotropic problems, compared to the (purely) graph-based orderings. It is therefore not surprising that, for the examples considered here, the performances of RCM, ND, and MMD are not significantly different from the result using the natural ordering.

4.6 Conclusion

This chapter presents a new reordering technique based on a multilevel graph coarsening strategy. The technique incorporates the use of algebraic information within a graph theoretic framework, to construct an effective reordering strategy for indefinite linear systems. Numerical results on examples from the Helmholtz equation application are presented, and a comparison of results against that of the natural ordering, as well as other standard reordering techniques, are made. The results indicate that the multilevel graph coarsening reordering strategy is superior to the natural ordering, or the ordering given by RCM, ND, or MMD on the Helmholtz problem. We observe significant gains in terms of the number of iterations required to reach convergence, and the overall computational time, even for the highly indefinite cases. Note that at high wavenumbers, reordering alone may not be enough to efficiently solve the Helmholtz problem. More robust algebraic strategies such as diagonal compensation (Shifted ILUT) or modified ILUT may be necessary to ensure good convergence. Nonetheless, the results suggest that the MLGC reordering technique can significantly improve the performance of these algebraic strategies. Observations made from the results may be summarized as follows:

- The natural ordering generally works quite well, particularly when the ordering reduces fill-ins
- Reorderings based solely on graph theory may be ineffective (or not beneficial) for highly indefinite problems, such as the Helmholtz problem at high wavenumbers

- Combining ideas from graph theory with algebraic information from the underlying adjacency matrix can be quite beneficial for indefinite problems

These observations agree with the results from the literature. See for instance [97–99] and references therein.

Chapter 5

Multilevel ILU

As problems become larger and more complex, obtaining an efficient and effective numerical solution becomes even more challenging. Large problems require optimal solvers, and optimal solvers require hierarchical algorithms [100]. Moreover, the difficulty for standard numerical solvers to solve a particular problem depends largely on the properties of the model, which can be seen explicitly in terms of structure; or implicitly in terms of ill-conditioning and indefiniteness of the resulting linear system of equations. Multilevel methods are a class of hierarchical algorithms that exploit these properties to solve a reduced system, corresponding to a smaller scale model of the original problem, from which the solution to the original problem may be obtained. The multilevel framework is characterized by three main schemes [7]. In the first scheme, the problem domain is fixed and a subset of the nodes is selected for the (approximate) solution at each level. This leads a sequentially efficient scheme by constructing multiple hierarchies of levels within the single domain. Both the geometric and algebraic multigrid class of solvers belong to this scheme. In the second scheme, the problem domain is decomposed into smaller subdomains, where the solution to each subproblem may be obtained simultaneously. Clearly, this scheme has good parallel efficiency. The class of Schur-based multilevel methods, to which multilevel ILU belongs, are one example of this scheme. The third scheme is simply a parallel combination of the first two scheme. A typical example is the parallel adaptation of domain decomposition techniques.

Much of the recent research effort on solving sparse linear systems by iterative techniques has been devoted to the development of effective preconditioners that scale

well, while offering good reliability. In this regard, multilevel methods that rely on incomplete LU (ILU) factorizations have been advocated by many authors in recent years [50–60]. While multigrid techniques [9,48] and their algebraic counterparts (AMG) [6, 49] are known to be optimally efficient for solving some classes of discretized partial differential equations on regular meshes, they may become ineffective when faced with more general types of sparse linear systems. However, the 'multilevel' or 'multistage' ingredient of multigrid can be easily married with general-purpose qualities of ILU preconditioners to yield efficient, yet more general-purpose, solvers.

5.1 Multilevel ILU

5.1.1 Domain Decomposition ILU

Domain decomposition [101,102] is a well-known method in numerical analysis typically used in the numerical solution of partial differential equations, particularly defined on (but not restricted to) complicated geometries. The basic idea is to decompose the problem into smaller subproblems on subdomains whose union constitutes the whole. These subdomains are independent of each other, which makes the domain decomposition method suitable for parallel computing. The motivation for the decomposition into the different subdomains may be the result of differing physics within different subdomains; the availability of fast solvers, such as multigrid or approximate inverse, for each subdomain; or the need to balance computational effort to facilitate parallel computing [103]. Consider a decomposition of the global domain into $k > 2$ subdomains,

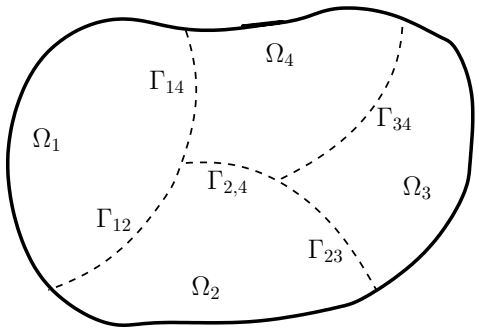


Figure 5.1: Example of a domain decomposed into four subdomains.

as in Figure 5.1 (for four subdomains). The nodes in the interior of these subdomains, Ω_i, are independent of each other, and the subdomains i and j are connected by the boundary Γ_{ij}. Suppose the problem matrix is assembled by ordering the interior nodes first, followed by the boundary nodes, we obtain:

$$PAP^T = \left[\begin{array}{cccc|c} B_{\Omega_1} & & & & F_{\Omega\Gamma}^{(1)} \\ & B_{\Omega_2} & & & F_{\Omega\Gamma}^{(2)} \\ & & \ddots & & \vdots \\ & & & B_{\Omega_k} & F_{\Omega\Gamma}^{(k)} \\ \hline E_{\Gamma\Omega}^{(1)} & E_{\Gamma\Omega}^{(2)} & \cdots & E_{\Gamma\Omega}^{(k)} & \sum_{j=1}^{k} C_{\Gamma\Gamma}^{(j)} \end{array}\right], \tag{5.1}$$

where P is some permutation matrix that achieves the desired ordering, B_{Ω_i} is the block matrix associated with the subdomain Ω_i, $E_{\Gamma\Omega}^{(i)}$ and $F_{\Omega\Gamma}^{(i)}$ are the submatrices associated with the boundary-interior interactions and interior-boundary interactions for subdomain i respectively, and $C_{\Gamma\Gamma}^{(j)}$ represents the boundary-boundary interactions within each separator j. The above equation may be written in a more concise 2×2 form as:

$$PAP^T = \begin{bmatrix} B & F \\ E & C \end{bmatrix}, \tag{5.2}$$

which after block factorization gives:

$$\begin{bmatrix} B & F \\ E & C \end{bmatrix} = \begin{bmatrix} B & 0 \\ E & I \end{bmatrix} \begin{bmatrix} I & B^{-1}F \\ 0 & S \end{bmatrix}, \tag{5.3}$$

where the Schur complement is S, is defined as $S = C - EB^{-1}F$ and can be obtained as a sum of the local Schur complements corresponding to each subdomain. In other words, we can write

$$S = \sum_{j=1}^{k} (C_{\Gamma\Gamma}^{(j)} - E_{\Gamma\Omega}^{(j)} B_{\Omega_k}^{-1} F_{\Omega\Gamma}^{(j)}),$$

which alludes to the local nature of the computations.

The linear system $Ax = b$, to be solved, now takes the form:

$$\begin{bmatrix} B & 0 \\ E & I \end{bmatrix} \begin{bmatrix} I & B^{-1}F \\ 0 & S \end{bmatrix} \begin{bmatrix} x_B \\ x_C \end{bmatrix} = \begin{bmatrix} b_B \\ b_C \end{bmatrix}, \tag{5.4}$$

and can be solved in the following three steps:

$$\begin{cases} Bz_B &= b_B \\ Sx_C &= b_C - Ez_B \\ Bx_B &= b_B - Fx_C \end{cases} \tag{5.5}$$

By simply expanding the above set of equations, one can clearly see how, except for the solution of the second equation, all assembly and solution steps (involving B) can be done in parallel. Furthermore, one can see how the above method can be suitable for direct solution, using the exact L, U factors of B and S. Note that since$B$ is block diagonal, the factorization of B does not incur any additional fill-ins outside the diagonal blocks. In other words, the fill-ins are contained within the local blocks constituting the individual subdomains. The Schur complement matrix can also be assembled appropriately to minimize fill-ins.

There are several techniques in the literature that adapt the above method into a preconditioner for an iterative method. These so called Schur-based preconditioners rely on developing a suitable approximation for the Schur complement matrix. Examples include the iterative substructuring methods [101, 104–106], and the hierarchical interface partitioning methods [54, 107].

5.1.2 Algebraic Recursive Multilevel ILU

Algebraic multilevel ILU methods, that combine the robustness of ILU factorizations with the multilevel efficiency of multigrid methods, have been proposed by many authors. See for instance [50–60, 64]. In this section, we discuss a recursive approach to construct an algebraic multilevel ILU method, designed primarily to precondition iterative solvers.

Like multigrid, one of the most important components of a multilevel ILU method is the strategy used in constructing the level structure of the method. This so-called coarsening strategy is essential to the overall performance of the multilevel method. As a result of the connection between graph theory and the Gaussian elimination process, it is not surprising that many of coarsening strategies for multilevel ILU are based on graph coarsening techniques.

The multilevel process begins by coarsening on the nodes in the adjacency graph of

the associated system matrix. This is done to obtain a C/F splitting of the nodes into 'coarse' and 'fine' sets. The result is an independent set ordering P, which permutes the matrix into the form

$$PAP^T = \begin{bmatrix} B & F \\ E & C \end{bmatrix}, \tag{5.6}$$

where B is a block diagonal matrix corresponding to a block independent set, as in Equation 5.2 above.A block factorization of the matrix is then obtained from this splitting and the process is continued recursively on the Schur complement associated with the coarse set.

At level ℓ of the recursion, the multilevel procedure reorders the matrix in the above form and then factors it as follows,

$$P_\ell A_\ell P_\ell^T = \begin{pmatrix} B_\ell & F_\ell \\ E_\ell & C_\ell \end{pmatrix} \approx \begin{pmatrix} L_\ell & 0 \\ E_\ell U_\ell^{-1} & I \end{pmatrix} \begin{pmatrix} U_\ell & L_\ell^{-1} F_\ell \\ 0 & A_{\ell+1} \end{pmatrix}, \tag{5.7}$$

where P_ℓ and P_ℓ^T are the permutation and its transpose, respectively. On the right-hand side of the equation, B_ℓ is factored into L_ℓ and U_ℓ by an incomplete LU factorization.

The idea then is to consider the matrix $A_{\ell+1}$ as the matrix of the next level and process it in turn by using the above strategy recursively. Algorithm 9 formally describes the process by performing k levels of the factorization.

Algorithm 9: Algebraic recursive multilevel factorization.

0. If $\ell = k$ then
1. Compute $A_\ell \approx L_\ell U_\ell$
2. Else:
3 Obtain a permutation P_ℓ
4. Apply permutation $A_\ell := P_\ell^T A_\ell P$
5. Compute block factorization
6. Recurse on $A_{\ell+1}$
7. EndIf

In line 1, the matrix at the last level, corresponding to the last level Schur complement, is factored by an incomplete LU factorization and used as a preconditioner for the solution of the reduced system at the last level. In line 3, the permutation is determined by the coarsening strategy used to obtain the C/F splitting of the adjacency graph.

Independent set algorithms yield permutations that tend to make B as diagonal (or block diagonal) as possible. One such example is the red-black ordering, which offers the possibility of reducing the unknowns by about half for the next level. Independent set algorithms generally allow for a cheaper approximation of the L and U factors of B, during the block factorization, which in turn facilitates a cheaper formulation of the Schur complement for the next level matrix. Several variations exist in the literature in constructing this permutation matrix. The general objective for multilevel ILU methods is to select a B-block (fine set) that is diagonally dominant so that solves with B are efficient [53,61,62]. Bollhöfer defines P_ℓ so that nodes that promote growth in the inverse LU factors of the ILU factorization at each level are put in the coarse set C (C-block of the reordered matrix) [36,63]. This helps to promote stability in the factorization of the B-block, leading to a stable overall multilevel factorization. In [60], Saad and Suchomel combine simple heuristics with an independent set algorithm to improve the quality of the resulting factors in the multilevel process: all rows that are judged poor from the point of view of diagonal dominance, are not considered as part of the independent set. In [53], Botta and Wubs define a reordering to select a nearly independent set of nodes that are diagonally dominant, and extract from this a B-block that is diagonal. This helps to promote stability in the factorization, while reducing the cost in the formation of the Schur complement matrix for the next level. In [64], Chow and Vassilevski use the strength-of-connection idea from multigrid to define the C/F splitting of the nodes. Unlike the other procedures based on attaining a diagonally dominant B-block, this approach tries to ensure that the coarse set provides a good interpolation for the original (fine grid) problem.

Line 5 of the algorithm computes the block factorization of the current matrix, and forms the next level matrix $A_{\ell+1}$. In exact terms, this matrix is the Schur complement associated with C_ℓ and defined as $A_{\ell+1} = C_\ell - E_\ell U_\ell^{-1} L_\ell^{-1} F_\ell$, where $B_\ell = L_\ell U_\ell$. Forming $A_{\ell+1}$ in this way can be quite expensive, and so in practice, an approximation of the Schur complement is used instead. One common approach is to approximate B^{-1} by a sparse approximation so that $A_{\ell+1}$ can be computed in a cheaper way. Some of these methods approximate B by a diagonal matrix [53], while others promote diagonal dominance in B and compute its sparse LU factors [36,60]. Other methods compute the sparse approximation $H_\ell \approx U_\ell^{-1} L_\ell^{-1} F_\ell$, so that the Schur complement is approximated

as $A_{\ell+1} = C_\ell - E_\ell H_\ell$. A variation of this approach, given in [60], computes the sparse approximation $G_{\ell+1} = E_\ell U_\ell^{-1}$ and $W_{\ell+1} = L_\ell^{-1} F_\ell$, so that $A_{\ell+1} = C_\ell - G_\ell W_\ell$. Notice that $G_{\ell+1}$ and $W_{\ell+1}$ appear in the block factorization and hence do not need to be explicitly computed. Thus, $A_{\ell+1}$ is obtained directly from a partial ILU factorization [60]. Another approach for approximating the Schur complement that is common in the multigrid literature defines $H_\ell \approx U_\ell^{-1} L_\ell^{-1} F_\ell$ as before, and constructs the operator

$$Q_\ell = \begin{bmatrix} H_\ell \\ I \end{bmatrix},$$

where I is the identity matrix. The Schur complement is then computed by the so-called Galerkin form as $A_{\ell+1} = Q_\ell^T A_\ell Q_\ell$.

Once the next level matrix is formed, the algorithm recurses on this matrix, and the process continues until the last level is reached. Recall that the above procedure is to be used as a preconditioner. At the last level, $A_{\ell+1}$ is approximated by an incomplete LU factorization, and used to precondition the coarse level system. In other words, the coarse level preconditioner is defined as: $M_\ell = A_{\ell+1}$. Thus, the finer level preconditioners can be computed as ($j = \ell - 1, \cdots, 1$)

$$M_j = P_j^T \begin{pmatrix} L_\ell & 0 \\ E_\ell U_\ell^{-1} & I \end{pmatrix} \begin{pmatrix} U_\ell & L_\ell^{-1} F_\ell \\ 0 & A_{\ell+1} \end{pmatrix} P_j. \tag{5.8}$$

The preconditioner solve, $P_j M_j P_j^T x = b$, using Equation 5.8 is obtained as

$$\begin{cases} L_j U_j z_1 &= b_1 \\ M_{j+1} x_2 &= b_2 - E_j z_1 \\ L_j U_j x_1 &= b_1 - F_j x_2 \end{cases} \tag{5.9}$$

To obtain x_2 requires a solve with M_{j+1}, leading to a (downward) recursion until the last level preconditioner M_ℓ is reached, and the coarse level system is solved. The solution x_1, is then obtained by an (upward) recursion using previously computed values for x_2. This recursion depicts a V-cycling pattern, which is a common occurrence in multigrid methods.

To improve robustness and performance on systems with poorly structured matrices, many preconditioners have been developed which make use of some form of permutation in their construction, see for instance [45–47]. It is clear that in lines 1 and 5 of

Algorithm 9, column pivoting techniques, such as ILUT with pivoting (ILUTP) [10], may be used to improve the quality of the factorization of $A_{\ell+1}$ or B respectively. Incomplete LU factorization with column pivoting will usually result in a more robust preconditioner than standard ILU, but this often comes with an additional cost in memory usage as pivoting with ILU will tend to generate more fill-in elements. In a more global sense, non-symmetric permutations may be used to construct the C/F splitting at each level. The diagonally dominant PQ ordering of Saad [82] is one such approach that takes advantage of considering non-symmetric permutations instead of the simple strategies based on independent sets. It follows a two-sided approach where permutations P and Q are applied to the rows and columns respectively, to transform a matrix A into

$$PAQ^T = \begin{pmatrix} B & F \\ E & C \end{pmatrix}, \tag{5.10}$$

instead of Equation 5.6. Unlike the symmetric version, there is no particular structure and no assumptions for the B-block. The permutation pair P, Q is selected such that the B-block has the most diagonally dominant rows after the non-symmetric permutation, and a few nonzero elements, so as to reduce fill-ins. Additional details of the technique may be found in [82, 108].

5.2 Some Numerical Results

In what follows, we present some results on the iterative solution of the Helmholtz problem, using a multilevel preconditioner. We test the performance of the algebraic recursive multilevel solver (ARMS) [60], from the complex iterative solver package ZITSOL[1] . We use the non-symmetric (ddPQ [82]) version of ARMS (ARMS-ddPQ) as it produced similar results as with the symmetric ARMS for simpler problems, and was more robust for harder problems. We also allowed a maximum of 4 intermediate levels for the ARMS factorization. The test problems considered here are identical to the problems considered in Section 2.2.4 of Chapter 2. Since these problems are indefinite, we also consider the shifted variants of the ARMS preconditioner, allowing for a comparison between the multilevel ILU preconditioner and the single-level ILUT

[1] http://www-users.cs.umn.edu/s̃aad/software/ITSOL/

preconditioner used in Section 2.2.4. In what follows, shifted ARMS (or ARMS-ddPQ) refers to using shifted ILUT to form the approximation to B, as well as the factorization at the last level (i.e. for $A_{\ell+1}$). Again, we consider both the τ-based scheme and the *dd*-based scheme for perturbing the matrix.

The iterative solution scheme uses restarted (flexible) GMRES, with a restart dimension of 60, and convergence of the iterative solver is assumed whenever the initial residual is reduced by a factor of $1.0\text{x}10^{-8}$.

In the first test case, the model problem has size $n = 29241$, with wavenumber $k = 8\pi$. This corresponds to a mesh resolution of $\frac{\lambda}{h} = 20$, where h is the element mesh size. The fill factor for each solve is fixed at ≈ 3.0 to allow for a fair comparison, and a maximum of 500 iterations is allowed for each test case. Figure 5.2 shows the convergence profile for the numerical solution of the above problem with the ARMS-ddPQ preconditioner and its shifted variants. The results show that the unshifted ARMS-ddPQ preconditioner is the least effective for this problem. The shifted variants showed better performance, with the τ-based scheme yielding the best results. Comparing this with Figure 2.12 shows a better convergence result for ARMS-ddPQ over ILUT in general. In particular, comparing the unshifted variants reveals that ARMS-ddPQ converges within the required number of iterations, whereas ILUT fails to converge.

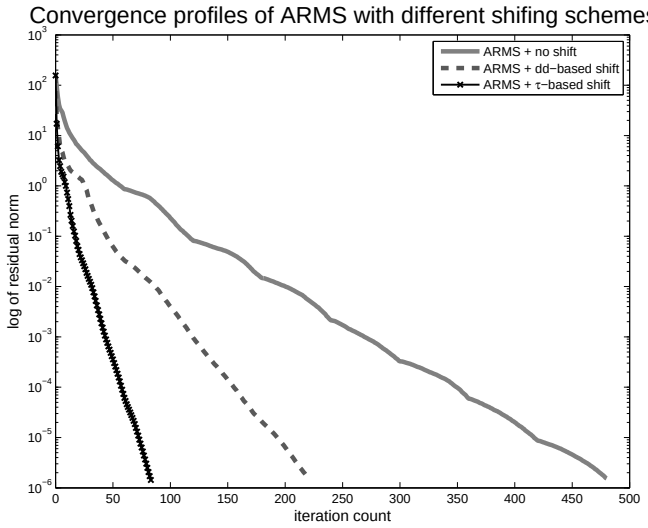


Figure 5.2: Convergence profile for the standard ARMS-ddPQ preconditioner and its shifted variants.

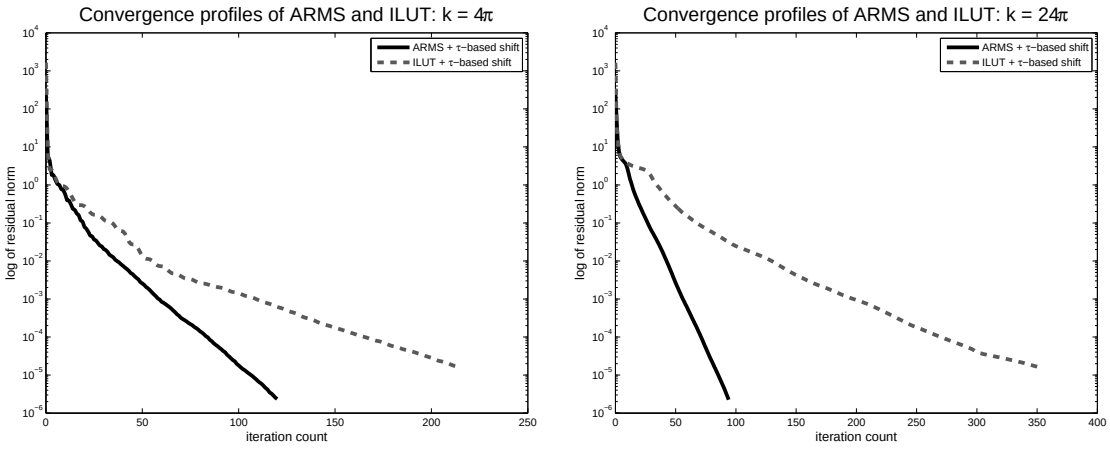
In the next example, we investigate the effect of the wavenumber k, on the different preconditioners. Here, the Helmholtz problem is solved on a 121 by 361 mesh. The resulting system has size $n = 43,681$, with $nnz = 387,000$ non-zero elements. We solve the problem for the wavenumbers $k = 4\pi$, 8π, 16π, and 24π.

Preconditioner	k	$\frac{\lambda}{h}$	iters	c_F	$\|\|(LU)^{-1}\mathbf{1}\|\|_2$
ARMS-ddPQ (no shift)	4π	60	120	3.50	$7.48e+03$
	8π	30	169	4.03	$1.66e+04$
	16π	15	282	4.50	$2.44e+03$
	24π	10	–	–	–
ARMS-ddPQ (dd-based)	4π	60	411	3.83	$5.12e+02$
	8π	30	311	4.37	5.67e+02
	16π	15	187	4.71	$3.92e+02$
	24π	10	185	3.00	$2.54e+02$
ARMS-ddPQ (τ-based)	4π	60	106	3.45	$7.56e+03$
	8π	30	69	3.84	$6.41e+03$
	16π	15	39	3.95	$1.26e+03$
	24π	10	94	3.02	$4.71e+02$

Table 5.1: Comparison of the different schemes for the ARMS-ddPQ preconditioner on the Helmholtz application problem with different wavenumbers.

Table 5.1 shows the results with the ARMS-ddPQ preconditioner. Here, standard ARMS-ddPQ without shifting stagnates for $k = 24\pi$. As in Section 2.2.4, we observe the interesting situation where it performs better compared to the dd-based shifted scheme for relatively low values of k. But as k increases, the dd-based shifted scheme starts to do better. At low values of k, the system is less indefinite, and standard unshifted ARMS can produce factors that are fairly stable and accurate. However, depending on the choice of the shift, the shifted scheme can yield a more stable factorization that is less accurate. The table shows that the choice of α used by the dd-based scheme seems to produce stable factors (as indicated by lower values for $||(LU)^{-1}\mathbf{1}||_2$) for all the different values of k. This enables it to perform well at high wavenumbers where the system is very indefinite. The τ-based scheme, however, does not suffer at lower values of k, and shows the best convergence for all choices of the wavenumber. Comparing these results with those in Table 2.4 shows superior performance for the multilevel ARMS-ddPQ over

the single-level ILUT.



(a) comparison for $k = 4\pi$

(b) comparison for $k = 24\pi$

Figure 5.3: Convergence profiles for the τ-based shifted ARMS and ILUT preconditioners

At relatively low wave numbers, the ARMS-ddPQ and ILUT preconditioning techniques both give good results. However, as the wave number increases and the problem becomes more indefinite, ILUT begins to struggle. Figure 5.3 shows the convergence profiles of ARMS-ddPQ and ILUT, shifted by the τ-based scheme. The fill factor is fixed at approximately 3.0, and we plot results for $k = 4\pi$ and $k = 24\pi$. As observed from the plot, the gap in performance between the two widens significantly for $k = 24\pi$.

5.3 Possible Contributions and Conclusion

In [109], Notay presents a theoretical comparison of algebraic multigrid and algebraic multilevel methods. He showed that the convergence of multilevel ILU methods like ARMS can be improved when the approximation to the B-block ($B \approx LU$) in the block ILU factorization is improved (in the spectral sense). As shown in Chapter 3, modified ILU techniques offer a good alternative to standard (unmodified) ILU methods as they yield factors that are generally more stable. Moreover, the success of shifted ILU methods on indefinite problems suggests that modified ILU techniques can be

successfully adapted to the multilevel framework. Modified approaches offer better control of the perturbation and, so could yield a better approximation to the B-block, leading to an overall better preconditioner.

Furthermore, the reordering techniques discussed in Chapter 4 provide an alternative to obtaining the C/F splitting needed by the multilevel framework. By following a pairwise aggregation approach, these techniques attempt to achieve independence while promoting diagonal dominance among the nodes in the B-block. The resulting coarse graph is thus a decent approximation to the fine level graph, which is a desirable property for multilevel methods, as the structure of the fine level graph is (somewhat) preserved [5, 7].

We have examined the performance of multilevel ILU factorizations as preconditioners for highly indefinite problems. Our results indicate that the multilevel preconditioners are much more effective and efficient on these problems. Incorporating strategies such as diagonal compensation also appears to significantly improve performance. In summary, multilevel methods are generally more robust compared to standard single-level techniques. However, their convergence is not fully understood and there is still some work to be done to improve performance and efficiently solve difficult problems.

Chapter 6

Adaptation to Parallel

The advent of super computers, coupled with the demand for the efficient solution of large applications in a reasonable amount of time, requires the development of scalable solvers and preconditioning techniques. However, development of "purely" parallel solver algorithms are much harder than the development of serial solvers. While some solvers such as multigrid and some implementations of the approximate inverse method [70] naturally fit into the parallel framework, many general purpose parallel solver algorithms are adaptations of their serial versions. However, adapting a serial solver into the parallel framework can result in a less effective parallel solver [24]. As such, the development of efficient parallel solver technologies, based on the improvement of serial adaptations to the parallel framework, continues to be an active area of research.

As noted earlier, the domain decomposition method is well suited to parallel computing, as a result of the decomposition of the domain into independent subdomains. This provides the opportunity to obtain efficient preconditioning strategies for Krylov accelerators, such as GMRES. The idea is to decompose the global domain across separate processors, where the local subproblems can be solved. The global solution to the original problem is then obtained by iteratively combining the solution between adjacent subdomains. Figure 6.1 shows a decomposition of a domain onto four processors: $P0$, $P1$, $P2$, and $P3$. Each processor solves a local problem, which corresponds to the global problem defined on the local domain. The computed solution is then communicated between adjacent nodes to correct the global solution. This is repeated iteratively,

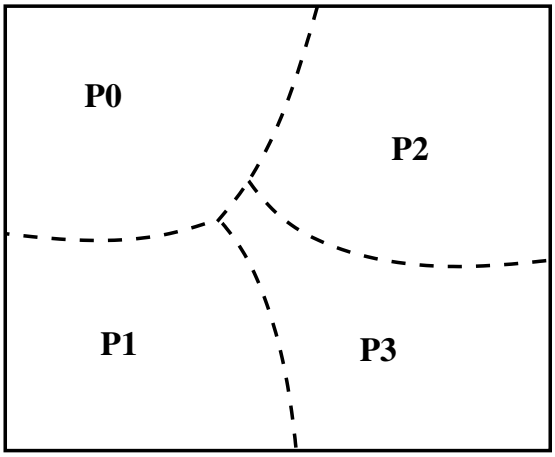


Figure 6.1: Example of a domain decomposed onto four processors.

until the global solution converges.

An important consideration for a domain decomposition-based parallel solver is how the domain is partitioned into the local subdomains. For some applications, the partitioning may be defined by the intrinsic physics within each subdomain, such as heterogeneity or anisotropy. However, for many practical applications, the decomposition into the separate subdomains is based on balancing the computational work load as equally as possible among the processors, and minimizing the communication volume between adjacent processors.

6.0.1 Partitioning the domain

Based on computational cost considerations, it is clear that balancing the work load can enhance performance. Furthermore, control over the communication volume between processors can significantly improve the scalability of the algorithm. Scalability here means the ability of the parallel algorithm to maintain or improve performance as the work load increases. This requires the efficient distribution of the linear system among the different processors such that inter-processor interaction is minimized.

The most common partitioning algorithms for parallel solvers are based on one-dimensional (1D) graph (or hypergraph) partitioning methods. A graph represents an ordered pair, $G = (V, E)$, consisting of a set of vertices, V, and a set of edges, E, defined

by a pair of vertices. A hypergraph is simply an extension of a graph, which allows a hyperedge to have more than two vertices. In these 1D methods, the partitioning is defined only on the set of equations or unknowns of the underlying linear system of equations, with the primary goal of achieving load balancing. This is equivalent to mapping the (possibly multi-dimensional) global domain onto a one-dimensional array, which can then be partitioned into chunks of equal weights [110]. Some of the commonly used algorithms are the nested dissection algorithm [44]; space filling curves [111, 112]; the level set expansion method [113]; and the hypergraph partitioning technique [114–116].

Two-dimensional (2D) partitioning techniques provide a way to partition the rows(equations) and columns(unknowns) of the underlying linear system, which may be exploited to minimize communication and significantly improve performance. The idea of 2D matrix partitioning dates as far back as the mid 1990's where it was mostly used as a means of improving computation and computational load balance, rather than minimizing communication volume [117–121]. In [122], Bisseling presents a 2D algorithm aimed at both improving computation and reducing communication [117]. Later in [123] and [124], Çatalyürek and Aykanat proposed two algorithms (coarse-grain and fine-grain) for 2D matrix partitioning using hypergraphs. This was based on an extension to a 1D algorithm by the same authors [125]. Hypergraph partitioners are known to accurately model the true communication volume (and not merely an approximation to it, as it is with other methods), making it the preferred approach [117, 126]. Later on in [117], Bisseling proposed a new 2D partitioning algorithm (Mondriaan) that uses recursive bipartitioning to minimize the true communication volume and improve load balancing. This algorithm, together with the coarse and fine-grain algorithms of Çatalyürek and Aykanat, have become the standard for two-dimensional matrix decomposition.

While the 2D schemes based on the hypergraph partitioning techniques are theoretically optimal, they are expensive to compute, which makes the 1D schemes quite competitive. Nonetheless, the 2D schemes provide the possibility to partition highly non-symmetric systems, as well as rectangular systems, in a more effective way, which makes the linear solver more robust. In [127], Boman and Wolf define a 2D partitioning strategy based on the nested-dissection algorithm that is cheap to compute and yields an effective reduction in communication costs.

The graph (or hypergraph) partitioning problem is known to be NP-complete. As such, many of these algorithms need to implement some heuristics to improve performance and obtain a good quality partitioning within a reasonable amount of time. Multilevel graph partitioning is one common approach used. Here, the original (fine) graph is coarsened into a smaller graph, where the partitioning algorithm may be efficiently implemented. The resulting partition on the coarse graph is then mapped back onto the original (fine) graph [96, 128].

There are several software packages for graph partitioning, that implement some of these algorithms and more, to minimize an objective function that defines the communication volume. The most commonly used code is METIS, a serial multilevel graph partitioning and fill-reducing matrix ordering software (parMETIS - parallel version) by Karypis [96]. Other codes include: Zoltan, a parallel partitioning, load balancing and data-management services software developed at SANDIA [129]; Chaco, a multilevel graph partitioning tool (similar to METIS) by Hendrickson and Leland; and hypergraph-based partitioning tools PaToH by Çatalyürek and Aykanat [130], and hMETIS also by Karypis [131].

6.0.2 Obtaining the parallel solution

There are several variations in how the final global solution is obtained. The Block Jacobi technique is the simplest of these methods. In this technique, only contributions from the interior of the local subdomains are used in updating the global solution. The resulting parallel preconditioner may be viewed as a block diagonal matrix, with each diagonal block corresponding to a local subdomain matrix. It is easy to see the parallelism in this method, as all computations are local and no communication is needed to update the global solution. This computational efficiency comes at a cost, as the overall parallel solution is slow to converge, due to the lack of domain-to-domain information exchange to update the solution.

Other methods extend the block Jacobi method by taking into consideration information exchange between adjacent processors. Examples of these are the additive Schwarz method [101, 132] and the Schur-based (substructuring) method [103]. These methods generally converge in fewer iterations that the block Jacobi method, but are typically more expensive to construct.

6.0.3 Additive Schwarz Method

The additive Schwarz method (ASM) is a simplification of the overlapping Schwarz method [101] for solving boundary value problems on multiple domains. Consider the solution to the discretized problem $Ax = b$ on some domain Ω. For simplicity we consider a partitioning of the domain into two subdomains, Ω_1 and Ω_2 with corresponding boundaries Γ_1 and Γ_2, as shown in Figure 6.2. Then the global solution by the

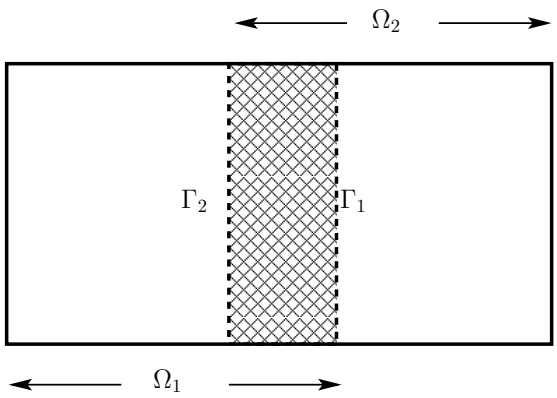


Figure 6.2: Example of a domain decomposed into overlapping subdomains.

overlapping Schwarz method is obtained by solving the local problems:

$$\begin{aligned} A_1 x_1^{k+\frac{1}{2}} &= b_1, \text{ with } x_1^{k+\frac{1}{2}} = x_2^k \\ A_2 x_2^{k+1} &= b_2, \text{ with } x_2^{k+1} = x_1^{k+\frac{1}{2}} \end{aligned}$$

where the subscripts 1 and 2 indicate the partitioning of the linear system matrix and the corresponding solution and right-hand side vectors for the respective domains Ω_1 and Ω_2, and the superscript k indicates the current iterate for the solution. With a little bit of manipulation, we can write the above equation in terms of the global variables as follows:

$$\begin{aligned} x^{k+\frac{1}{2}} &= x^k + G_1(b - Ax^k) \\ x^{k+1} &= x^{k+\frac{1}{2}} + G_2(b - Ax^{k+\frac{1}{2}}) \end{aligned} \tag{6.1}$$

where $G_i = R_i^T A_i^{-1} R_i$ is (in this example) a 2×2 matrix with A_i^{-1} on the i-th diagonal, and zero everywhere else. The restriction operator R_i restricts the global variables to

the subdomain owned by processor i, and the prolongation operator R_i^T maps the local variables in subdomain i, onto the global domain. This approach to solving the linear system is commonly referred to as the multiplicative Schwarz method, since the scheme may be written as a product of operators to update the global solution.

From Equation 6.2, each iteration involves sequential fractional steps. Furthermore, the residual, $b - Ax^k$, has to be recomputed at each of these steps. As a result, the multiplicative Schwarz scheme is not very suitable for parallel computing. In [133], Dryja and Widlund suggest the use of multi-coloring to reorder the subdomains to improve parallelism. Dryja and Widlund later developed the additive Schwarz method, mainly designed to take advantage of parallelism. Unlike the multiplicative Schwarz method, this scheme may be written as the sum of operators to update the global solution. With respect to the above example, the solution by ASM is obtained as follows:

$$x^{k+1} = x^k + (G_1 + G_2)(b - Ax^k), \tag{6.2}$$

and for multiple subdomains, this generalizes to:

$$x^{k+1} = x^k + \sum_i G_i(b - Ax^k).$$

To further increase parallel efficiency by cutting communication costs, the restricted additive Schwarz method (RAS) was proposed by Cai and Sarkis [134, 135]. In this method, during the iterative solution process, a prolongation operator, corresponding to the non-overlapping part of the local subdomain is used instead. This scheme may be formally expressed as:

$$x^{k+1} = x^k + \sum_i \tilde{G}_i(b - Ax^k),$$

where $\tilde{G}_i = (R_i^0)^T A_i^{-1} R_i$, and $(R_i^0)^T$ maps variables from the non-overlapping part of subdomain i, onto the global domain. Note however, that residual information from the overlapped region is considered in the solution update, and hence the restricted additive Schwarz method is generally more effective than the block Jacobi method. Observe also that when the operator $\tilde{G}_i$ is defined as $\tilde{G}_i = (R_i^0)^T A_i^{-1} R_i^0$, then RAS (and ASM for that matter), are equivalent to the block Jacobi method.

Like block Jacobi, the additive schwarz methods can be used as preconditioners for Krylov accelerators. In this case, it becomes unnecessary to iterate until the solution

converges. In practice, preconditioning with RAS (or ASM) applies one step of RAS (or ASM) to perform the preconditioner solve, for each iteration of the Krylov method. This makes for a cheap preconditioning strategy that is still quite effective.

6.0.4 Schur-based methods

The parallel solution by Schur-based methods is similar to the substructuring technique described in Section 5.1.1. Suppose that the global domain is decomposed into s non-overlapping subdomains. Then the global system matrix may be written by ordering the local subdomain matrices first, followed by the matrices corresponding to the interior boundary interfaces, as follows:

$$A = \left[\begin{array}{cccc|c} A_1 & & & & F_{1\Gamma}^{(1)} \\ & A_2 & & & F_{2\Gamma}^{(2)} \\ & & \ddots & & \vdots \\ & & & A_s & F_{s\Gamma} \\ \hline E_{\Gamma 1} & E_{\Gamma 2} & \cdots & E_{\Gamma s} & C_{\Gamma\Gamma} \end{array}\right], \tag{6.3}$$

where A_i is the local subdomain matrix residing on processor i, $E_{\Gamma i}$ and $F_{i\Gamma}$ are the submatrices associated with the boundary-interior coupling and interior-boundary coupling for the subdomain on processor i respectively, and $C_{\Gamma\Gamma}$ represents the boundary-boundary interactions of the interior domain boundary. The unknowns associated with the local subdomain matrices on each processor can be eliminated simultaneously, since they are independent of each other. This is equivalent to performing some form of block factorization on the global system matrix to obtain the global Schur complement system defined as

$$S = C_{\Gamma\Gamma} - \sum_{i=1}^{s} E_{\Gamma i} A_i^{-1} F_{i\Gamma}.$$

Notice that the local subdomain matrix on each processor can be reordered in the same way as in Equation 6.3 corresponding to local interior variables and local interface variables adjacent to the processor boundary. In this way, it is easy to see that each local processor will have a local contribution to the global Schur complement system.

In other words, the global Schur complement matrix will have a block structure.

$$\begin{bmatrix} S_1 & H_{12} & H_{13} & \cdots & H_{1s} \\ H_{21} & S_2 & H_{23} & \cdots & H_{2s} \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ H_{s1} & H_{s2} & H_{s3} & \cdots & S_s \end{bmatrix}, \tag{6.4}$$

where S_i corresponds to the local Schur complement matrix on processor i, and H_{ij} corresponds to variables from adjacent processors j, that contribute to the local equations on processor i.

Suppose now, that the right-hand side and solution vectors are partitioned to conform to the global partitioning, and let the subscript c associate a variable to the global Schur complement system, then the parallel solution to the global system $Ax = b$ may be obtained by solving

$$Sx_c = b_c - \sum_{i=1}^{s} E_{\Gamma i} A_i^{-1} b_i \tag{6.5}$$

to obtain the solution to the unknowns at the processor boundaries, and then solving

$$A_i x_i = b_i - F_{i\Gamma} x_c \tag{6.6}$$

to obtain the solution to the unknowns in the local processor subdomain. Equation 6.6 is local to each processor, and hence can be done in parallel once the solution x_c is obtained. However, as evidenced by the block structure of the Schur complement matrix, solving Equation 6.6 cannot be done in parallel, as it involves updates to the local solution between adjacent processors. Nonetheless, in the context of preconditioning, an approximate solution to the Schur complement system is sufficient. A simple approach is to approximate the Schur complement matrix by a block diagonal matrix corresponding to the diagonal of S. In this way, the global preconditioner is equivalent to the block Jacobi preconditioner. Alternatively, a more efficient parallel solution can be obtained by applying a preconditioned Krylov method to solve the Schur complement system. Here, block Jacobi, as well as the additive Schwarz methods discussed earlier, may be used as the preconditioner [55, 136, 137].

The requirement to obtain a good (parallel) solution to the Schur complement system generally makes the Schur-based preconditioning methods computationally expensive,

compared to the additive Schwarz variants. However, the flexibility of the Schur-based method to accommodate different solution schemes for the Schur complement system makes it a viable alternative. Once a good solution to the Schur complement system is obtained, the Schur-based method is generally performs better than the additive Schwarz variants, and has been shown to be very effective on many different applications [136].

For more information on domain decomposition methods for preconditioning, and some parallel implementations of the method, refer to [101–106, 132–135]

6.0.5 Solving the local system

Solving the parallel system requires the solution of local systems corresponding to the interior of the subdomains. Since these systems are independent, the solves can be performed simultaneously in parallel. Some parallel solvers such as HIPS [107], which is based on the Schur-based approach, make use of direct methods to obtain the exact solution for the unknowns in the interior of the subdomains. This can be very effective if the partitioning is such that the local subdomain is small enough, otherwise it can be quite expensive. Sophisticated techniques, such as supernode algorithms, symbolic factorization and innovative pivoting strategies are usually incorporated to improve performance. Alternatively, approximate solution techniques such as multigrid, approximate inverse, or incomplete LU factorizations may be used, see for instance [55, 138, 139].

6.1 pARMS: A parallel ILU solver

This section presents an overview of the parallel algebraic recursive multilevel solver package (pARMS[1]) [55, 140, 141], developed in collaboration with Zhongze Li[2] , Yousef Saad[3] , and Masha Sosonkina[4] , and which benefits from some of the ideas presented in this thesis.

pARMS is an ILU-based preconditioned Krylov solver package for the parallel solution of sparse linear systems. The Krylov accelerators currently supported by the package are GMRES and flexible GMRES. pARMS implements three global (parallel)

[1] http://www-users.cs.umn.edu/~saad/software/pARMS/
[2] Texas Instruments, Dallas, TX
[3] Dept. of Computer Science, Univ. of Minnesota, Twin Cities, MN
[4] Scalable Computing Lab., Ames Lab, Ames, IA

preconditioning strategies, each of which can use one of four local preconditioning strategies. The global preconditioners are the block Jacobi method, the restricted additive Schwarz method, and the Schur-based method. The local preconditioning strategy used to approximate the local (subdomain) matrices are the ILU(0) factorization, the ILU(k) factorization, the ILUT factorization, and the multilevel ILU preconditioner, ARMS, (a sequential algebraic recursive multilevel solver).

The distributed matrices stored on the local processors consist of two main parts. The first is the submatrix corresponding to variables native to the local subdomain. The second is an extension of the local submatrix to include variables from adjacent processors that contribute to the local equations of the current processor. Furthermore, each processor permutes the local submatrix so that variables that are not coupled with other variables on adjacent processors are ordered first. Next in the ordering are variables that contribute to the local equations of adjacent processors, and then finally variables that receive contributions from adjacent processors. This allows for a non-symmetric communication pattern during the parallel solve, which benefits non-symmetric linear systems. Moreover, extending the local submatrix by including data from adjacent processors makes it easy and efficient to perform operations like matrix-vector products, involving the subdomain variables. Furthermore, it also improves efficiency when the Schur-based preconditioner is used, as the local contributions to the global Schur complement matrix, are easily obtained.

6.1.1 Some Results

The following are some results from using pARMS within the Cats3D code to solve the buoyancy-driven flow problem described in Section 1.4.2. The pARMS solvers considered here use GMRES as the accelerator and block Jacobi as the global (or parallel) preconditioner. The local preconditioners considered are the ILUT factorization and its multilevel counterpart, ARMS. The fill-factor for these local preconditioners is kept under 3.0. Results are also presented for using GMRES preconditioned with the block Jacobi with ILU(0) preconditioner from PETSC[5] , the default solver in Cats3D; the RAS preconditioner also from PETSC; and the AMG preconditioner from HYPRE[6] .

[5] http://www.mcs.anl.gov/petsc/
[6] https://computation.llnl.gov/casc/hypre/software.html

We consider three different examples in order of problem size. For each example, we present results for increasing values of the Rayleigh number (from a low of $1.0e1$, to a high of $1.0e5$), which leads to an increasingly challenging problem for the solver. Note that as a result of the incompressibility constraint in the velocity-pressure formulation of the Navier-Stokes equations, the Jacobian matrix associated with the numerical solution of the problem has some zero diagonals. This makes it unfavorable to construct an effective preconditioner to solve the Jacobian linear system, by a black-box solver. As such, the matrix used to construct the preconditioner is first perturbed on the diagonal, by a small term $\approx 1.0e-4$, before the preconditioner is formed. All the examples were run on the IBM BladeCenter machine at the Minnesota Supercomputing Institute (MSI). The first example simulates a problem of size $300,000$ on 6 processors. Here, we use restarted GMRES with a restart dimension of 50, and set the maximum number of iterations to be 300. Table 6.1 shows the results for this example. For the second example, we simulate a problem of size $1,500,000$ on 32 processors, using the same preconditioners. Here, we increase the dimension of the Krylov dimension to 100 to avoid stagnancy. The results are presented in Table 6.2. The third example simulates a problem of size $3,940,000$ on 80 processors, again using the same preconditioners. Here, the GMRES restart dimension is set to 200, with a maximum number of iterations of 600. Table 6.3 contains the results for this example.

The tables report results in terms of the total number of Newton iterations required, as well as the time taken for the overall simulation to reach convergence. The AMG preconditioner appears to be ineffective for high values of the Rayleigh number. The simulation either timed-out or did not converge within the maximum allowable number of Newton iterations. All the other preconditioners converged within the required number of iterations, for the different values of the Rayleigh number. Comparing the two pARMS preconditioners, it appears that the (single level) ILUT preconditioner is more efficient in terms of computational time, than the (multilevel) ARMS preconditioner. While both preconditioners took relatively similar number of iterations to converge, the ARMS preconditioner took longer to converge compared to the ILUT preconditioner. This may be attributed, in part, to the potential cost of computing the Schur complement as an approximation to the next-level matrix in the multilevel hierarchy. The ARMS preconditioner considered here uses an independent set algorithm to define the

Ra	10		1e3		1e5	
	Niters	Time	Niters	Time	Niters	Time
Default	8	1.4s	11	1.9s	10	1.7s
RAS	7	1.4s	8	1.7s	10	2.0s
AMG	3	3.2s	4	4.8s	>30	>28.8s
ilut	6	1.2s	7	1.4s	11	2.3s
arms	5	1.4s	6	1.7s	10	2.9s

Table 6.1: Total number of Newton iterations (Niters) and the corresponding CPU time (Time). $n = 300,000$; nproc = 6; GMRES(50, 300)

Ra	10		1e3		1e5	
	Niters	Time	Niters	Time	Niters	Time
default	16	4.2s	24	6.0s	25	6.2s
RAS	11	3.8s	13	4.1s	19	5.8s
AMG	4	7.6s	5	10.2s	>21	>30.0s
ilut	8	2.3s	12	3.4s	17	4.9s
arms	9	3.6s	13	4.4s	20	7.8s

Table 6.2: Total number of Newton iterations (Niters) and the corresponding CPU time (Time). $n = 1,500,000$; nproc = 32; GMRES(100, 300)

Ra	10		1e3		1e5	
	Niters	Time	Niters	Time	Niters	Time
default	7	4.5s	12	7.5s	14	8.7s
RAS	8	6.1s	10	7.6s	15	11.2s
AMG	3	11.2s	4	17.7s	>19	>30.0s
ilut	7	4.7s	7	4.7s	16	10.5s
arms	6	5.3s	7	6.1s	16	13.9s

Table 6.3: Total number of Newton iterations (Niters) and the corresponding CPU time (Time). $n = 3,940,000$; nproc = 80; GMRES(200, 600)

coarsening strategy for the multilevel scheme, which may not be ideal for this problem. It may be necessary to consider more robust coarsening strategies the convergence and efficiency of the multilevel method, for this problem.

Considering the overall performance of the different preconditioners on the different examples, it appears that the ILUT preconditioner from pARMS yields the best results. In terms of computational time, it wins 6 out of 9 times, and generally converges in fewer iterations than the RAS and default PETSC preconditioners. The default PETSC solver does better than the rest on 3 occasions. However, it generally requires more Newton iterations to reach convergence. For lower values of the Rayleigh number, the AMG preconditioner requires the fewest number of iterations to converge. However, this convergence is computationally expensive, which makes the AMG preconditioner the least efficient in terms of computational time, among the different preconditioners and for all the different examples. Like ARMS, a possible explanation for this could be the result of large operator complexities in the formation of the coarse grid operators within the AMG framework.

6.1.2 Conclusion: Improving the parallel ILU solver

From the earlier discussion in this chapter, it is obvious that the performance of a parallel iterative solver depends on the quality of the preconditioning technique used. The performance of parallel preconditioning techniques, based on domain decomposition strategies, depend on both the quality of the global preconditioning scheme as well as that of the local preconditioning scheme. Thus improving either scheme can contribute to a better performance for the parallel solver.

Regarding the parallel framework, it is clear that a good partitioner that appropriately partitions the problem can improve performance. This is still an active area of research. Two-dimensional partitioners are gaining popularity as they provide a compromise between balancing the work load and minimizing communication volume. One-dimensional strategies like the chain-on-chain partitioning (CCP) algorithms have been shown to provide fast optimal load balancing, and facilitate an easy mapping between processors and the computational domain [110,142]. Also, physics based strategies, that exploit the algebraic properties of the linear system to define the partitioning have been proposed [143].

The Schur-based scheme is another area where improvements can be made. Improvement here refers to obtaining an effective preconditioner for the solution of the global Schur complement system. Numerical results in [136] show that the performance of the Schur preconditioned solver can be significantly superior to the additive Schwarz variants when RAS is used to precondition the solution to the Schur complement system. Algebraic multigrid techniques provide another option, since some implementations are well suited to the parallel framework. Solving the Schur complement system by an algebraic multigrid solver may not be ideal, as the properties of the system may be unfavorable for the AMG method. Nonetheless, the AMG method can serve as an effective preconditioning strategy for the solution of the Schur complement system.

Solving the local subdomain problem requires the same considerations as solving a linear system sequentially. In particular, the local systems can be indefinite and ill-conditioned, which can prove challenging for the ILU preconditioner. The ideas presented in this thesis can be incorporated here to improve the quality of the resulting ILU factorization. This leads to an improvement in the overall performance of the global preconditioner. In this regard, the shifted ILU schemes presented in this thesis have been implemented in the pARMS solver package.

Chapter 7

Conclusion

Physics-based preconditioners, in which the preconditioner is implicitly or explicitly derived from the underlying physics of the problem, is generally more effective than a general purpose preconditioner on the same problem. However, obtaining an effective physics-based preconditioner is not trivial, particularly when the problem contains physics that span multiple scales. General purpose preconditioners provide an alternative to physics-based preconditioners, in that, they are easily attainable and can be applicable to a wide range of problems. Furthermore, ideas from general purpose preconditioners can serve as a starting point for developing an effective physics-based preconditioner.

This thesis presents and discusses several strategies for developing an effective and efficient general purpose ILU-based preconditioner, for the iterative solution of indefinite and ill-conditioned linear systems, by a preconditioned Krylov solver. The Krylov solver considered here utilizes the (flexible) GMRES method as the accelerator. The numerical examples presented are from applications in acoustic wave scattering, crystal simulation, and the standard 5 point finite difference discretization of the Laplace equation with homogeneous boundary conditions.

7.1 Summary

The ILU factorizations of indefinite and ill-conditioned systems are prone to unstable pivots. Diagonal compensation techniques have been shown to be quite effective for

improving the stability of the L and U factors, by ensuring that the size of the entries in L and U are contained within reasonable limits. The success of these so-called shifted ILU preconditioners lies in the choice of the compensation strategy. In Chapter 2, we illustrate the effect of shifting on the stability and accuracy of the ILU factorization. We begin with an analysis of ILU(0) factorization defined on a regular 2D 5-point stencil grid. We show the effect of shifting on the ILU(0) factorization, and propose strategies for selecting the shift based on stability and accuracy considerations. Numerical results on the shifted Laplace problem are presented to support the theory. We then present two new criteria for automatically selecting the shift for complex symmetric and indefinite matrices, within the context of the more general ILUT factorization. These strategies are based on a simple heuristic to safeguard stability by improving diagonal dominance, without compromising the accuracy of the factorization. Numerical results on applications from acoustic scattering indicate that the new shifted ILUT schemes are more robust than the standard ILUT scheme, for highly indefinite problems.

Modified ILU methods improve the preconditioner by enforcing a matching condition for the preconditioner and the original system matrix, on some vector. In classical modified ILU, during the elimination of a particular row (or column), the diagonal entry is modified by adding to it the sum of the dropped terms. The result is that the preconditioner, $M = LU$, and the original matrix, A, give the exact same result when applied to a vector with constant entries, typically the vector of all ones. The quality of the resulting preconditioner depends on how the matching is done. That is, what vector or vectors are used in the matching, and, more generally, what entries are modified. Recent work in [144] discusses this in detail, and presents an algorithm that constrains M and A to give the same result on an arbitrary set of 'matching' vectors, and uses a least-squares approach to modify multiple coefficients in each row of the factorization. In Chapter 3, we present a new modified ILU technique based on the threshold-based ILUT algorithm. Here, we introduce a column-based modified ILUT algorithm that spreads the compensation (or modification term) over the non-unit diagonal, as well as the nonzero entries in the L part of the column, in an optimal way. This strategy is further extended for complex indefinite systems by considering a purely imaginary compensation term. Numerical results indicate that these modified ILUT methods are robust for both poorly conditioned and indefinite problems.

Chapter 4 presents a new reordering strategy for ILU, based on a multilevel graph coarsening idea. The new technique uses algebraic information from the system matrix to define a coarsening strategy, based on a pairwise aggregation technique, for the multilevel framework. The strategy ensures a stable factorization by moving 'poor' nodes to the end of the factorization, to avoid unstable pivots. Furthermore, the use of algebraic information in the coarsening process helps to control the size of the norms of L^{-1} and U^{-1}, so that they are not too large, which leads to more stable triangular solves. Numerical results show superior performance over standard graph-based strategies such as RCM, ND, and MMD, as well as the natural ordering, on applications from acoustic wave scattering.

In recent years, there has been some interest in combining the robustness of ILU methods, with the multilevel ingredient of multigrid methods. Several authors have proposed several formulations, and consequently, adaptations of the so-called multilevel ILU methods. See for instance [36, 51–53, 60, 64]. In Chapter 5, we present a discussion of the key features of the multilevel ILU factorization. This is done within the context of an algebraic recursive multilevel preconditioning scheme. We discuss issues relating to the selection of the hierarchy of algebraic 'grid' levels within the multilevel framework, that is, the way in which the coarse and fine nodes are selected at each level, as well as issues relating to the formation of the matrix for the next level (i.e the Schur complement). We also discuss and show how some of the ideas presented in this thesis may be incorporated into the multilevel scheme. Numerical results on examples from acoustic scattering illustrate the benefit of multilevel ILU methods, over standard single level ILU techniques, on some highly indefinite problems.

The advent of advanced supercomputers, coupled with the need for large scale simulations, has led to growing interest in the development of robust parallel solver technologies. Domain decomposition techniques are a popular choice for parallel solver development, since they are naturally adaptable to parallel computing. In Chapter 6, we review the basic infrastructure for parallel ILU solvers, based on the domain decomposition framework. We introduce the parallel ILU solver, pARMS, which has been partly developed and maintained as part of this doctoral research, and discuss how the ideas presented in this thesis may be incorporated within the parallel ILU framework, to improve solver performance. We also present some numerical results that show the

effectiveness of the pARMS solver on applications from crystal simulation, compared to other parallel algebraic solvers.

7.2 Future Work

The ideas presented in this thesis are based on some fundamental research in the literature. As such, this thesis is not an indication of completed work, but rather, a contribution to ongoing research. Much of the work presented here leave room for future research. In what follows, we mention a few of these areas, and make some suggestions regarding the direction of future work.

In order to take advantage of row-based dropping strategies, or to be consistent with codes that use row-based data structures, the column-based modified ILUT strategy presented in Chapter 3 may need to be adapted to a row-based scheme. This adaptation can be quite straightforward. Recall that the optimal spreading idea for ILUT is based on stability considerations, which is inherently column-based. Nonetheless, one can consider a 'transposed' version of the modified ILUT algorithm (Algorithm 4), in which the compensation term is spread over the diagonal entry, as well as the U-part of the row. Consider an LDU data structure, where L stores the lower triangular part of the factorization (with a unit diagonal), D stores the (non-unit) diagonal, and U stores the strict upper part of the factorization. Then the elimination of a particular row can be done exactly as in Algorithm 4 for the column version. Here, the U-part of the row would be scaled by the diagonal entry after the modification by the optimal spreading strategy. Note that in order for this row-based scheme to yield similar results to the column-based variant, it is necessary to transpose the system matrix prior to the (row-based) modified ILUT factorization, and transpose it back afterwards. We are currently working on an implementation of the modified ILUT technique described in this thesis, within the row-based pARMS solver package. It is possible to adapt the modified ILUT scheme to other formulations of the ILUT algorithm, for instance the Crout version of ILUT. However, note that for the Crout version to be efficient, row and column storage schemes will have to be used for the L and U parts of the factorization, respectively [145]. Nonetheless, the factorization benefits from more robust strategies for dropping terms [36, 145, 146], which could lead to very interesting and potentially

more effective variations of the modified ILUT algorithm.

In Chapter 5, we mentioned how the multilevel graph coarsening idea described in Chapter 4 may be incorporated within the multilevel framework. Ongoing work is focused on the use of some of these graph coarsening ideas to define the coarsening for a new multilevel ILU preconditioner, based on the algebraic recursive multilevel solver (ARMS) framework. One important aspect of the multilevel scheme is how the next (coarse) level matrix is formed. For multilevel ILU methods such as ARMS, this coarse level matrix takes the form of the Schur complement. The formation of the Schur complement can be quite expensive, and the cost can dominate the overall cost of the multilevel preconditioner. Nonetheless, for some poorly conditioned and indefinite problems, it may be necessary to have a good approximation to the Schur complement matrix in order to yield a good multilevel preconditioner. In general, in order to obtain a robust multilevel ILU preconditioner, it is necessary to have an approach for computing the next-level matrix that is relatively inexpensive, and provides a reasonably accurate representation of the underlying problem. Thus a key area of interest for possible future work is in the efficient composition or formation of the Schur complement matrix, for the multilevel ILU scheme. Another area of work worth exploring is the development and use of more robust ILU factorizations within the multilevel framework. For instance, one can incorporate the modified ILUT strategy from Chapter 3 of this thesis, to obtain a multilevel modified ILUT preconditioner.

Finally, as discussed in Section 6.1.2, improving the parallel ILU solver requires improving the different components of the parallel solution scheme. Here, the future work will focus primarily on improving the pARMS solver to make it a more robust and efficient parallel solver. This will include incorporating some of the new ideas presented in this thesis into the parallel framework; incorporating non-symmetric partitioning techniques to complement and benefit the non-symmetric data structure and parallel programming model within pARMS; and exploring new ways to solve the global Schur complement system, for the Schur-based parallel preconditioner. Moreover, a version of pARMS that exploits symmetry could be more appealing and efficient for symmetric problems.

References

[1] F.H. Streitz, J.N. Glosli, M.V. Patel, B. Chan, R.K. Yates, B.R. de Supinksi, J. Sexton, and J.A. Gunnels. Simulating solidification in metals at high pressure: the drive to petascale computing. *J. of Physics: Conf. Series*, 46:254–267, 2006.

[2] A. Brandt. Multi-level adaptive solutions to boundary value problems. *Mathematics of Computation*, 31:333–390, 1977.

[3] S. F. McCormick. *Multigrid Methods*. Frontiers in Applied Mathematics. SIAM, Philadelphia, 1987.

[4] W. Hackbusch. *Multi-Grid Methods and Applications*, volume 4 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1985.

[5] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, San Diego, CA, 2001.

[6] A. Ruge and K. Stüben. Algebraic multigrid. In S. McCormick, editor, *Multigrid Methods*, volume 3 of *Frontiers in Applied Mathematics*, chapter 4. SIAM, 1987.

[7] K. Chen. *Matrix Preconditioning Techniques and Applications*. Cambridge University Press, Cambridge, UK,, 2005.

[8] A. Brandt, S. F. Mc Cormick, and J. Ruge. Algebraic multigrid (amg) for sparse matrix equations. In D. J. Evans, editor, *Sparsity and its applications*, Cambridge, 1984. Cambridge Univ. Press.

[9] W. L. Briggs, V. E. Henson, and S. F. Mc Cormick. *A multigrid tutorial*. SIAM, Philadelphia, PA, 2000. Second edition.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd edition.* SIAM, Philadelpha, PA, 2003.

[11] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[12] Y. Saad and K. Wu. DQGMRES: a direct quasi-minimal residual algorithm based on incomplete orthogonalization. *Numerical Linear Algebra with Applications*, 3:329–343, 1996.

[13] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. *Numerical Linear Algebra with Applications*, 4:43–66, 1997.

[14] Y. Saad. Analysis of augmented Krylov subspace techniques. *SIAM J. Matrix Anal. Appl.*, 18:435–449, 1997.

[15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM Journal on Scientific and Statistical Computing*, 14:461–469, 1993.

[16] S. Gerschgorin. Ueber die abgrenzung der eigenwerte einer matrix. *Izv. Akad. Nauk. SSSR Ser. Mat.*, 1:749754, 1931.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, 1996.

[18] R. S. Varga. *Matrix Iterative Analysis, Second ed.* Springer-Verlag, Berlin, 2002.

[19] M. Magolu Monga Made, R. Beauwens, and G. Warzee. Preconditioning of discrete Helmholtz operators perturbed by a diagonal complex matrix. *Comm. in Numer. Meth. in Engin.*, 16(11):801–817, 2000.

[20] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers.* SIAM, Philadelphia, 1998.

[21] L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditioners i - theory. *SIAM Journal on Matrix Analysis and Applications*, 14:45–53, 1993.

[22] P. M. Vaidya. *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis.*

[23] D. Chen and S. Toledo. Vaidya's preconditioners: Implementation and experimental study. *Electronic Transactions on Numerical Analysis*, 16:30–49, 2003.

[24] Y. Saad. *Preconditioning techniques for highly indefinite systems - Slides from Workshop on Solution Methodologies for Scattering Problems in Pau, France*, 2007.

[25] N.I. Buleev. A numerical method for the solution of two-dimensional and three-dimensional equations of diffusion. *Math. Sb.*, 51:227–238, 1960.

[26] T.A. Oliphant. An implicit numerical method for solving two-dimensional time-dependent diffusion problems. *Quart. Appl. Math.*, 19:221–229, 1961.

[27] R.S. Varga. Factorization and normalized iterative methods. In R.E. Langer, editor, *Boundary problems in differential equations*, pages 121–142. Univ. of Wisconsin Press, Madison, 1960.

[28] T. Dupont, R. Kendall, and H. Rachford. An approximate factorization procedure for solving self-adjoint elliptic difference equations. *SIAM J. Numer. Anal.*, 5:559–573, 1968.

[29] O. Axelsson. A generalized SSOR method. *BIT*, 12:443–467, 1972.

[30] J. A. Meijerink and H. A. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of Computation*, 31(137):148–162, 1977.

[31] J. W. Watts III. A conjugate gradient truncated direct method for the iterative solution of the reservoir simulation pressure equation. *Society of Petroleum Engineers Journal*, 21:345–353, 1981.

[32] N. Munksgaard. Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method. *ACM Transactions on Mathematical Software*, 6:206–219, 1980.

[33] Y. Saad. ILUT: a dual threshold incomplete ILU factorization. *Numerical Linear Algebra with Applications*, 1:387–402, 1994.

[34] H. C. Elman. A stability analysis of incomplete LU factorizations. *Mathematics of Computation*, 47:191–217, 1986.

[35] A. M. Bruaset, A. Tveito, and R. Winther. On the stability of relaxed incomplete lu factorizations. *Mathematics of Computation*, 54:701–719, 1990.

[36] M. Bollhöfer. A robust ILU with pivoting based on monitoring the growth of the inverse factors. *Linear Algebra and its Applications*, 338(1-3):201–213, 2001.

[37] N. Li, Y. Saad, and E. Chow. Crout versions of ILU for general sparse matrices. *SIAM Journal on Scientific Computing*, 25(2):716–728, 2003.

[38] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, 1986.

[39] A. George and J. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[40] A. George and J. Liu. Evolution of the minimum degree ordering algorithm. *SIAM review*, 31:1–19, 1989.

[41] W. F. Tinney and J. W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE*, 55:1801–1809, 1967.

[42] D. J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph Theory and Computing, R. C. Read, ed., Academic Press, New York*, pages 183–217, 1972.

[43] P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996.

[44] A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

[45] M. Benzi, J. C. Haws, and Tůma. Preconditioning highly indefinite and nonsymmtric matrices. *SIAM Journal on Scientific Computing*, 22(4):1333–1353, 2000.

[46] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM Journal on Matrix Analysis and Applications*, 22(4):973–996, 2001.

[47] M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra and its Applications*, 240(1–3):131–151, 1996.

[48] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, New York, 2001.

[49] P. Vanek, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56:179–196, 1996.

[50] R. E. Bank and C. Wagner. Multilevel ILU decomposition. *Numerische Mathematik*, 82(4):543–576, 1999.

[51] A. van der Ploeg, E.F.F. Botta, and F.W. Wubs. Nested grids ILU-decomposition (NGILU). *J. Comp. Appl. Math.*, 66:515–526, 1996.

[52] E.F.F. Botta, A. van der Ploeg, and F.W. Wubs. A fast linear-system solver for large unstructured problems on a shared-memory computer. In O. Axelsson and B. Polman, editors, *Proceedings of the Conference on Algebraic Multilevel Methods with Applications*, pages 105–116, 1996.

[53] E.F.F. Botta and F.W. Wubs. Matrix Renumbering ILU: an effective algebraic multilevel ILU. *SIAM Journal on Matrix Analysis and Applications*, 20:1007–1026, 1999.

[54] Pascal Henon and Yousef Saad. A parallel multistage ILU factorization based on a hierarchical graph decomposition. *SIAM Journal on Scientific Computing*, 28(6):2266–2293, 2006.

[55] Z. Li, Y. Saad, and M. Sosonkina. pARMS: a parallel version of the algebraic recursive multilevel solver. *Numerical Linear Algebra with Applications*, 10:485–509, 2003.

[56] M. Magolu monga Made and H.A. van der Vorst. A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings. *Future Generation Computer Systems*, 17(8):925–932, 2001.

[57] M. Magolu monga Made and H.A. van der Vorst. Parallel incomplete factorizations with pseudo-overlapped subdomains. *Parallel Computing*, 27(8):989–1008, 2001.

[58] M. Magolu monga Made and H.A. van der Vorst. Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap. *Numerical Linear Algebra with Applications*, 9(1):45–64, 2002.

[59] Y. Saad. ILUM: a multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 17(4):830–847, 1996.

[60] Y. Saad and B. Suchomel. ARMS: An algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9, 2002.

[61] Yousef Saad and Jun Zhang. Diagonal threshold techniques in robust multi-level ILU, preconditioners for general sparse linear systems. *Numerical Linear Algebra with Applications*, 6:257–280, 1999.

[62] L. Grosz. Preconditioning by incomplete block elimination. *Numerical Linear Algebra with Applications*, 7:527–541, 2000.

[63] M. Bollhöfer, M. J. Grote, and O. Schenk. Algebraic multilevel preconditioner for the helmholtz equation in heterogeneous media. *SIAM Journal on Scientific Computing*, 31:3781–3805, 2009.

[64] E. Chow and P. S. Vassilevski. Multilevel block factorizations in generalized hierarchical bases. *Numerical Linear Algebra with Applications*, 1:1–22, 2002.

[65] Riyad Kechroud, Azzeddine Soulaimani, Yousef Saad, and Shivaraju Gowda. Preconditioning techniques for the solution of the Helmholtz equation by the finite element method. *Math. Comput. Simul.*, 65(4-5):303–321, 2004.

[66] A. Yeckel and J. J. Derby. Convective heat and mass transport in novel bridgman configurations for cadmium zinc telluride growth. In *Proc. XXI ICTAM*, ICTAMXXI, Warsaw, Poland, 2004.

[67] D. S. Kershaw. The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26(1):43–65, 1978.

[68] Thomas A. Manteuffel. Shifted incomplete Cholesky factorization. In *Sparse Matrix Proceedings 1978 (Sympos. Sparse Matrix Comput., Knoxville, Tenn., 1978)*, pages 41–61. SIAM, Philadelphia, Pa., 1979.

[69] Tony F. Chan and Henk A. van der Vorst. Approximate and incomplete factorizations. In *ICASE/LARC INTERDISCIPLINARY SERIES IN SCIENCE AND ENGINEERING*, volume 56, pages 167–202, 1994.

[70] E. Chow. *Robust Preconditioning for Sparse Linear Systems.* PhD thesis, Dept. of Comp. Sc., Univ. of Minnesota, Twin Cities, 1997.

[71] Daniel Osei-Kuffuor and Yousef Saad. Preconditioning Helmholtz linear systems. *Applied Numerical Mathematics*, 60(4):420–431, 2009.

[72] M. B. van Gijzen, Y. A. Erlangga, and C. Vuik. Spectral analysis of the discrete helmholtz operator preconditioned with a shifted laplacian. *SIAM J. Sci. Comput.*, 29:1942–1958, 2007.

[73] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. A novel multigrid based preconditioner for heterogeneous helmholtz problems. *SIAM J. Sci. Comput.*, 27:1471–1492, 2006.

[74] T. Airaksinen, E. Heikkola, A. Pennanen, and J. Toivanen. An algebraic multigrid based shifted-laplacian preconditioner for the helmholtz equation. *J. Comp. Phys.*, 226(1):1196 – 1210, 2007.

[75] Y. A. Erlangga, C. W. Oosterlee, and C. Vuik. Comparison of multigrid and incomplete lu shifted-Laplace preconditioners for the inhomogeneous helmholtz equation. *Appl. Numer. Math.*, 56:648–666, 2006.

[76] I. Gustafsson. A class of first order factorization methods. *BIT*, 18:142–156, 1978.

[77] Y. Notay. DRIC: a dynamic version of the RIC method. *Numer. Lin. Alg. Applic.*, 1994. to appear.

[78] T.F. Chan and H. Elman. Fourier analysis of iterative methods for elliptic problems. *SIAM Review*, 31(1):20–49, 1989.

[79] H. van der Vorst. The convergence behaviour of preconditioned CG and CGS in the presence of rounding errors. In O. Axelsson and L.Y. Kolotilina, editors, *Preconditioned conjugate gradient methods*. vol. 1457, Lecture notes in Math., Springer Verlag, 1990.

[80] S. Doi and A. Hoshi. Large numbered multicolor MILU preconditioning on SX-3/14. *Int'l J. Computer Math.*, 44:143–152, 1992.

[81] J. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[82] Y. Saad. Multilevel ILU with reorderings for diagonal dominance. *SIAM Journal on Scientific Computing*, 27(3):1032–1057, 2005.

[83] O. Axelsson and P. Vassilevski. Algebraic multilevel preconditioning methods. I. *Numer. Math.*, 56:157–177, 1989.

[84] A. C. Muresan and Y. Notay. Analysis of aggregation-based multigrid. *SIAM Journal on Scientific Computing*, 30:1082–1103, 2008.

[85] Y. Notay. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis*, 37:123–146, 2010.

[86] Y. F. Hu and R. J. Blake. *Load balancing for unstructured mesh applications*, pages 117–148. Nova Science Publishers, Inc., Commack, NY, USA, 2001.

[87] George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *Proc. 24th Intern. Conf. Par. Proc., III*, pages 113–122. CRC Press, 1995.

[88] A. Brandt. Generally highly accurate algebraic coarsening. *Electronic Transactions on Numerical Analysis*, 10:1–20, 2000.

[89] O. Livne. Coarsening by compatible relaxation. *Numerical Linear Algebra with Applications*, 11:205–227, 2004.

[90] R.D. Falgout and P.S. Vassilevski. On generalizing the amg framework. *SIAM Journal on Numerical Analysis*, 42:1669–1693, 2005.

[91] R. E. Bank. Compatible coarsening in the multigraph algorithm. *Advances in Engineering Software*, 38:287–294, 2007.

[92] J. Brannick. Compatible relaxation and coarsening in algebraic multigrid. *SIAM Journal on Scientific Computing*, 32:1393–1416, 2010.

[93] J. Chen and I. Safro. Algebraic distance on graphs. Technical Report MCS-P1696-1009, ANL, Argonne National Laboratory, 2010.

[94] D. Ron, I. Safro, and A. Brandt. Relaxation-based coarsening and multiscale graph organization. *SIAM Multiscale Modelling and Simulations*, 9:407–423, 2011.

[95] A. Dax. The convergence of linear stationary iterative processes for solving singular unstructured systems of linear equations. *SIAM review*, 34:611–635, 1990.

[96] G. Karypis and V. Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and computing Fill-Reducing Orderings of Sparse Matrices, version 4.0.1,University of Minnesota, Dept. of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis*, 1998.

[97] R. Bridson and W.-P. Tang. Ordering, anisotropy, and factored approximate inverses. *SIAM Journal on Scientific Computing*, 21(3):867882, 1999.

[98] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang. Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems. *SIAM Journal on Matrix Analysis and Applications*, 13(3):944961, 1992.

[99] R. Bridson and W.-P. Tang. A structural diagnosis of some ic orderings. *SIAM Journal on Scientific Computing*, 22(5):15271532, 2000.

[100] K. Stüben. *Multigrid Tutorial: Multigrid (MG) and Local Refinement for Elliptic Partial Differential Equations, Slides on Multigrid Overview, url: http://www.ipam.ucla.edu/publications/es2002/es2002_1449.pdf*, 2002.

[101] B. Smith, W. Gropp, and P. Borjstad. *Domain Decomposition Methods*. Cambridge University Press, Cambridge, UK, 1996.

[102] A. Toselli and O. Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer Verlag, Berlin, Germany, 2005.

[103] M. O. DEVILLE, P. F. FISCHER, and E. H. MUND. *High-Order Methods for Incompressible Fluid Flow*. Cambridge University Press, Cambridge, UK, 2002.

[104] P. Le Tallec. Domain decomposition methods in computational mechanics. *Comp. Mech. Adv.*, 2:121–220, 1994.

[105] T. Lu, T. M. Shih, and C. B. Liem. *Domain Decomposition Methods - New Techniques for Numerical Solution of Partial Differential Equations*. Science Press, Beijing, China, 1992.

[106] T. F. Chan and T. P. Matthew. Domain decomposition algorithms. In *Acta Numerica 3*, pages 61–143. Cambridge University Press, 1994.

[107] J. Gaidamour and P. Henon. A parallel direct/iterative solver based on a schur complement approach. In *IEEE 11th International Conference on Computational Science and Engineering*, pages 98–105, Sao Paolo, Brazil, 1996.

[108] S. MacLachlan and Y. Saad. Greedy coarsening strategies for non-symmetric problems. *SIAM Journal on Scientific Computing*, 29(5):2115–2143, 2007.

[109] Y. Notay. Aggregation-based algebraic multilevel preconditioning. *SIAM Journal on Matrix Analysis and Applications*, 27:998–1018, 2006.

[110] A. Pinar, E. K. Tabak, and C. Aykanat. One-dimensional partitioning for heterogeneous systems: Theory and practice. *J. Parallel Distrib. Comput.*, 68:1473–1486, November 2008.

[111] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with spacefilling curves. *IEEE Trans. Parallel Distrib. Syst.*, 7:288–300, March 1996.

[112] H. Kutluca, T. M. Kurç, and C. Aykanat. Image-space decomposition algorithms for sort-first parallel volume rendering of unstructured grids. *J. Supercomput.*, 15:51–93, January 2000.

[113] T. Goehring and Y. Saad. Heuristic algorithms for automatic graph partitioning. Technical Report UMSI-94-29, UMN, University of Minnesota, 1994.

[114] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph based clustering in high-dimensional data sets: A summary of results. *IEEE Bulletin of the Technical Committee on Data Engineering*, 21:1–8, 1998.

[115] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10:673–693, July 1999.

[116] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. Faik J. D. Teresco, J. E. Flaherty, and L. G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52:133–152, February 2005.

[117] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Rev.*, 47:67–95, January 2005.

[118] J. G. Lewis and R. A. van de Geijn. Distributed memory matrix-vector multiplication and conjugate gradient algorithms. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, Supercomputing '93, pages 484–492, New York, NY, USA, 1993. ACM.

[119] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *Internat. J. High Speed Comput.*, 7:73–88, 1995.

[120] A. T. Ogielski and W. Aiello. Sparse matrix computations on parallel processor arrays. *SIAM J. Sci. Comput.*, 14(3):519–530, May 1993.

[121] A. Pinar and C. Aykanat. Sparse matrix decomposition with optimal load balancing. In *Proceedings of the Fourth International Conference on High-Performance Computing*, HIPC '97, pages 224–, Washington, DC, USA, 1997. IEEE Computer Society.

[122] R. H. Bisseling. *Parallel iterative solution of sparse linear systems on a transputer network, in Parallel Computation*, volume 46, pages 253–271. Oxford University Press, Oxford, UK, 1993.

[123] Ü. V. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '01, pages 28–28, New York, NY, USA, 2001. ACM.

[124] Ü. V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proceedings of the 15th International Parallel & Distributed Processing Symposium*, IPDPS '01, pages 118–, Washington, DC, USA, 2001. IEEE Computer Society.

[125] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Syst.*, 10(7):673–693, July 1999.

[126] E. G. Boman. A nested dissection approach for sparse matrix partitioning. In *Proc. Appl. Math. Mech. PAMM,*, ICIAM07, 2007.

[127] E. G. Boman and M. M. Wolf. A nested dissection approach to sparse matrix partitioning for parallel computations. Technical Report 2008-5482J, SNL, Sandia National Laboratories, 2008.

[128] Erik G. Boman, Umit V. Catalyurek, Cedric Chevalier, Karen D. Devine, Ilya Safro, and Michael M. Wolf. Advances in parallel partitioning, load balancing and matrix ordering for scientific computing. *Journal of Physics: Conference Series*, 180(1), 2009.

[129] K. Devine, E. Boman, R. Heapby, B. Hendrickson, and C. Vaughan. Zoltan data management service for parallel dynamic applications. *Computing in Science and Engg.*, 4(2):90–97, March 2002.

[130] Ü. V. Çatalyürek and C. Aykanat. Patoh: A multilevel hypergraph partitioning tool, version 3.0. Technical report, Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey, 1999.

[131] G. Karypis, V. Kumar, R. Aggarwal, and S. Shekhar. *hMETIS: A Hypergraph Partitioning Package, version 1.5.3, University of Minnesota, Dept. of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis*, 1998.

[132] M. Dryja and O. B. Widlund. An additive variant of the schwarz alternating method for the case of many subregions. Technical Report TR 339, NYU, Dep. Of Comput. Sci., Courant Institute, 1987.

[133] M. Dryja and O. B. Widlund. Domain decomposition algorithms with small overlap. *SIAM J. Sci. Comput.*, 15:604–620, May 1994.

[134] J. H. Kimn. *Additive Schwarz and Multilevel Methods, handout, url:https://www.math.lsu.edu/k̃imn/DD.pdf.*

[135] X. C. Cai and M. Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21:792–797, 1997.

[136] Z. Li and Y. Saad. SchurRAS: A restricted version of the overlapping Schur complement preconditioner. *SIAM Journal on Scientific Computing*, 27:1787–1801, 2006.

[137] S. Ma and Y. Saad. Distributed ILU(0) and SOR preconditioners for unstructured sparse linear systems. Technical Report ahpcrc-94-027, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, 1994.

[138] S. Balay, W.D. Gropp, L. C. McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[139] M. Heroux, R. Bartlett, V. Howle R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. An Overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories, 2003.

[140] M. Sosonkina, Y. Saad, and X. Cai. Using the parallel algebraic recursive multilevel solver in modern physical applications. *Future Generation Computer Systems*, 20:489–500, 2004.

[141] Yousef Saad and Masha Sosonkina. pARMS: A package for the parallel iterative solution of general large sparse linear systems user's guide. Technical Report UMSI-2004-8, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2004.

[142] A. Pinar and C. Aykanat. Fast optimal load balancing algorithms for 1d partitioning. *J. Parallel Distrib. Comput.*, 64:974–996, August 2004.

[143] Masha Sosonkina and Yousef Saad. Hypergraph partitioning for sparse linear systems: A case study with a simple discontinuous PDE. Technical Report UMSI-2009-29, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2009. Submitted.

[144] S. Maclachlan, D. Osei-Kuffuor, and Y. Saad. Modification and compensation strategies for threshold-based incomplete factorizations. Technical Report UMSI-2011-104, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2011.

[145] N. Li and Y. Saad. Crout versions of the ILU factorization with pivoting for sparse symmetric matrices. *Electronic Transactions on Numerical Analysis*, 20:75–85, 2006.

[146] M. Bollhöfer and Y. Saad. Multilevel preconditioners constructed from inverse–based ILUs. Technical Report UMSI-2004-75, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2004. In print - special issue on the 8-th Copper Mountain Conference.