SUPPORT GRAPH PRECONDITIONING FOR ELLIPTIC FINITE ELEMENT PROBLEMS

A Dissertation

by

MEIQIU WANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2008

Major Subject: Computer Science

SUPPORT GRAPH PRECONDITIONING FOR ELLIPTIC FINITE ELEMENT PROBLEMS

A Dissertation

by

MEIQIU WANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee, Vivek Sarin
Committee Members, Ricardo Gutierrez-Osuna
Jianer Chen
Raytcho Lazarov
Head of Department, Valerie E. Taylor

December 2008

Major Subject: Computer Science

ABSTRACT

Support Graph Preconditioning
for Elliptic Finite Element Problems. (December 2008)
Meiqiu Wang, B.S. Peking University;
M.S., Peking University;
M.S., University of Houston
Chair of Advisory Committee: Dr. Vivek Sarin

A relatively new preconditioning technique called support graph preconditioning has many merits over the traditional incomplete factorization based methods. A major limitation of this technique is that it is applicable to symmetric diagonally dominant matrices only. This work presents a technique that can be used to transform the symmetric positive definite matrices arising from elliptic finite element problems into symmetric diagonally dominant M-matrices. The basic idea is to approximate the element gradient matrix by taking the gradients along chosen edges, whose unit vectors form a new coordinate system. For Lagrangian elements, the rows of the element gradient matrix in this new coordinate system are scaled edge vectors, thus a diagonally dominant symmetric semidefinite M-matrix can be generated to approximate the element stiffness matrix. Depending on the element type, one or more such coordinate systems are required to obtain a global nonsingular M-matrix. Since such approximation takes place at the element level, the degradation in the quality of the preconditioner is only a small constant factor independent of the size of the problem. This technique of element coordinate transformations applies to a variety of first order Lagrangian elements. Combination of this technique and other techniques enables us to construct an M-matrix preconditioner for a wide range of second order elliptic problems even with higher order elements.

Another contribution of this work is the proposal of a new variant of Vaidya's support graph preconditioning technique called modified domain partitioned support graph preconditioners. Numerical experiments are conducted for various second order elliptic finite element problems, along with performance comparison to the incomplete factorization based preconditioners. Results show that these support graph preconditioners are superior when solving ill-conditioned problems. In addition, the domain partition feature provides inherent parallelism, and initial experiments show a good potential of parallelization and scalability of these preconditioners.

To my parents

ACKNOWLEDGMENTS

First and foremost, I would like to express my sincere thanks to my advisor Dr. Vivek Sarin, for his guidance, support, and confidence in me. His invaluable help makes every step of progress possible. Without his insightful guidance this work wouldn't be finished. He also has been supporting me financially during all these years, thus that I can concentrate on my work without worries.

I also would like to thank my committee members: Dr. Gutierrez-Osuna, Dr. Chen and Dr. Lazarov. I took classes from each one of them and have learned a lot. All their classes are amazing like themselves — the best in town! I appreciate their time to serve on my committee, and their guidance, support and help during my graduate study.

Thanks also go to all the faculty and staff of Computer Science Department, for making my graduate studies at Texas A&M University a great experience.

I also would like to take this opportunity to express my gratitude to Alice Backsen and her family, their friendship enriched my graduate life so very much. I am also thankful to my lab-mates, especially Hemant and Thomas, a casual conversation with them was often a delightful moment during the day. My other friends, Bo, Wenjuan, Yufei and Alex, I cannot imagine how to survive all these years without them.

Of course, I am also extremely thankful to my family. Especially to my little sister Meiyan, for being the best daughter for my parents and the best sister I have ever dreamed to have! Her love and encouragement sustained me through the toughest time. I am also grateful that both of my parents are healthy while I am far away from them. I will always be in debt to their sacrifice, love and patience.

Finally, I would like to thank Slavik who opened a brand new chapter of my life.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Consider solving the linear system,

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

where matrix $A \in \mathbb{R}^{N \times N}$, $x, b \in \mathbb{R}^N$, and N is an integer (only real matrices are considered in this dissertation). In practice, A is often large and sparse, as those appear in solving partial differential equations (PDEs) by finite difference or finite element methods (FEM). In fact, solving (1.1) is the innermost computational kernel of many large-scale scientific applications and industrial numerical simulations; it typically consumes a significant portion of the overall computation. How to solve (1.1) efficiently is one of the most important problems in scientific computing.

The term *large* varies with respect to current computer hardware technology. As for now, a large linear system usually has the number of unknowns N above a hundred thousands (in practice, it is usually up to millions). As computer technology advances and the demand for a more accurate solution increases, the definition of a large system is likely to raise the value of N. We say an $N \times N$ matrix A is *sparse* if A has only $O(N)$ non-zero entries. To take advantage of the sparsity, zero entries of A are not explicitly stored. A dense matrix requires storage of $O(N^2)$, whereas only $O(N)$ is needed for a sparse matrix of the same order. Another aspect of dealing with sparse matrix is to avoid operations on zero entries to minimize the number of operations throughout the computations. Sparse linear systems are often solved using different techniques from those employed to solve dense systems. A large sparse

The journal model is SIAM Journal on Scientific Computing.

linear system that cannot be solved by a standard solver on modern computers may be solved by a sparse solver.

I.1. Direct methods vs. iterative methods

There are two general approaches for solving linear systems: direct methods and iterative methods. A direct method produces the solution x after a finite number of operations (see, e.g., [8]). A common direct method is a variant of Gaussian elimination that factors A into matrices L and U, where L is lower triangular and U is upper triangular, the solution is obtained by solving the triangular systems by forward and backward substitution. For a sparse system, a naive direct method often suffers prohibitive complexity due to *fill-in* during factorization: storage complexity of $O(N^2)$ and computation complexity of $O(N^3)$. Special techniques are often employed to reduce *fill-in* such as the *minimum degree* ordering [22], the *approximate minimum degree* ordering [1], the *reverse Cuthill-McKee* (RCM) ordering [23], and the *nested dissection* (ND) ordering [21, 33] etc. In two dimensional problems, the nested dissection has been shown to be asymptotically optimal. A direct method using nested dissection technique may require storage of $O(N \log N)$ and computation of $O(N^{3/2})$.

Thus, successful direct methods are based on clever implementation of Gaussian elimination that exploits the sparsity structure of A as much as possible to minimize *fill-in* and avoid computations with zero entries. However, for large systems, these methods are often too expensive, except where A has a special structure. For a general PDE-related problem discretized over grids in three dimensional domain, optimal direct techniques require $O(N^{2.3})$ floating point operations [44]. Moreover, the storage requirement for large three dimensional numerical simulations makes direct methods prohibitively expensive. Another big disadvantage of direct methods is that they are difficult to parallelize, since solving a sparse triangular system had been shown to be the bottleneck in parallelization [27].

An alternative to direct methods are iterative methods. Iterative methods are often the choice of solving large sparse linear systems efficiently. Iterative methods construct a sequence of approximations $\{x^k\},\ k = 1, 2, \cdots$, of the solution x, for which $x^k \to x$ as $k \to \infty$. Some well known simple iterative methods include Jacobi method, Gauss-Seidel method and Successive Overrelaxation (SSOR) method (see, e.g., [26, 37]). Iterative methods are both storage and work efficient – as for a simple iterative method, it requires only $O(N)$ storage, the work per iteration is proportional to N. If an iterative method converges in considerably fewer than N iterations, then it would be more efficient than a direct solver. Moreover, the primary operation in iterative methods is the matrix-vector multiplication, which is fairly easy to parallelize.

However, for large linear systems, these basic iterative methods typically converge very slowly and often not at all. Acceleration techniques are needed, such as Krylov subspace methods. These methods are based on a projection process, with the search subspace being the Krylov space $p(A)$, where p is a polynomial. Different choice of the projection process gives a different Krylov subspace method. (see, e.g., [45, 37]). The convergence of Krylov subspace methods usually can be accelerated further by applying preconditioning techniques. Using preconditioning increases the work per iteration, and in general it decreases the parallel performance of iterative methods.

It is worth mentioning that preconditioning has a more important role than just speeding up the iterative process. When solving the linear system (1.1), if the condition number $\kappa(A)$ is large, then a small change in b or A may cause a relatively large change in the solution x, and we say (1.1) is *ill-conditioned* (see, e.g., [41]). In practice, b and A often involve some form of approximation. For example, if they are computed, the rounding errors were inevitably introduced. Thus, the solution of

(1.1) is unreliable if $\kappa(A)$ is very large. By applying preconditioning technique, the condition number of the modified linear system is decreased, a reliable solution may be obtained.

I.2. Support graph preconditioning

We restrict (1.1) to the problems of large sparse symmetric positive definite (SPD) systems. This class of problems seems special. However, a substantial fraction of linear equation problems arising in science and engineering have this property (see, e.g., [23]). The linear system obtained from finite difference or finite element discretization of second order elliptic boundary value problems belongs to this class. When the matrix A is SPD, one of the most popular Krylov subspace methods called conjugate gradient method (CG) is the algorithm of choice. The number of iterations of CG is bounded above by the square root of the spectral condition number of A. To increase the rate of convergence, one can use the preconditioned CG method (PCG) where a preconditioner M is used to approximate A. PCG converges rapidly when the condition number $\kappa(M^{-1}A)$ is small (see, e.g., [37]). The main challenge for iterative methods in general and PCG in particular, is to construct a preconditioner such that the linear system $My = d$ can be solved efficiently in every iteration, in addition the number of iterations is reduced considerably.

Support graph preconditioning in the context of PCG algorithm is a relatively new technique that has gained attention in recent years. Support graph preconditioners were first proposed by Vaidya [42] for symmetric, positive semidefinite, diagonally dominant M-matrices. The preconditioner M is constructed by dropping off-diagonal entries from the original matrix A. Unlike the conventional incomplete factorization based preconditioning, matrix M is factored exactly. Thus, it is important to drop entries that lower the *fill-in* in the factors without compromising the preconditioner's quality. This approach has several attractive features: the scheme yields robust preconditioners whose effectiveness can be controlled during the edge elimination process; the quality of the preconditioner appears to be relatively insensitive to boundary conditions and domain characteristics such as anisotropy and inhomogeneity; there are theoretical results to quantify the behavior of the preconditioner. References on this topic include [24, 18, 16, 46]. A recent study [40] shows that this approach is able to solve (1.1) in almost linear time.

I.3. Proposed work

Emerged as a very promising preconditioner with many good features, however, the most severe limitation of support graph preconditioning technique is that it is only applicable to a small set of matrices – the symmetric diagonally dominant matrices. In order to extend this technique to the second order elliptic finite element problems, which give SPD matrices in general, we use a strategy of two-level preconditioning: first, a symmetric diagonally dominant M-matrix A' is constructed to approximate the coefficient matrix A; then, a support graph preconditioner M constructed via the matrix A' is used for the original matrix A. This requires that the condition number $\kappa(A'^{-1}A)$ should be as small as possible, and in the ideal case, it does not depend on the number of unknowns.

This dissertation presents a novel technique that can be used to transform the symmetric positive definite matrices arising from elliptic finite element problems into symmetric diagonally dominant M-matrices. The basic idea is to approximate the element gradient matrix by taking the gradients along chosen edges, whose unit vectors form a new coordinate system. For Lagrangian elements, the rows of the element gradient matrix in this new coordinate system are scaled edge vectors, thus a diagonally dominant symmetric semidefinite M-matrix can be generated to approximate the element stiffness matrix. Depending on the element type, one or more such coordinate systems are required to obtain a global nonsingular M-matrix. Since such approximation takes place at the element level, the degradation in the quality of the preconditioner is only a small constant factor independent of the size of the problem. We show that this technique of element coordinate transformations applies to a variety of first order Lagrangian elements. Combination of this technique and other techniques enables us to construct an M-matrix preconditioner for a wide range of second order elliptic problems even with higher order elements.

The advance of today's technology has led to a dramatic growth in the size of the linear systems to be handled, and it yet to be seen to grow over the years. The use of parallel computers becomes necessary. However, it is a big challenge to design a good algorithm which performs well both on serial and parallel computers. We propose a new variant of support graph preconditioner called modified domain partitioned support graph (MDPSG) preconditioner. Compared to our earlier scheme – domain partitioned support graph (DPSG) preconditioner [46], the construction of MDPSG does not involve the finite element mesh and can therefore be served as a general black box algorithm for all symmetric diagonally dominant M-matrices. Numerical experiments and performance comparison with the incomplete factorization based preconditioners are conducted for various second order elliptic finite element problems. Results show that these support graph preconditioners are superior when solving ill-conditioned problems. In addition, the domain partition feature provides inherent parallelism, which is performance friendly on multi-core or multiple processors. Initial experiments show a good potential of parallelization and scalability of these preconditioners.

The rest of the dissertation is organized as follows. Chapter II provides mathematical background for support graph theory and finite element methods. Chapter III outlines the element level coordinate transformation technique that gives an M-matrix approximation for the stiffness matrix arising from the second order elliptic boundary value problems. Strategies on solving inhomogeneous problems, anisotropic problems and high order element problems are also discussed. This is followed by a description and analysis of MDPSG in Chapter IV. Experimental results of the MDPSG preconditioners for various second order elliptic finite element problems and the performance comparison to the incomplete factorization based preconditioners are reported in Chapter V. The potential parallelization of MDPSG is also discussed. Finally, Chapter VI gives the concluding remarks and future work.

CHAPTER II

BACKGROUND

This chapter introduces the definitions and notations used throughout this dissertation. Section II.1 introduces definitions and operators from basic calculus. Section II.2 introduces definitions, propositions and theorems from basic linear algebra. With that, a simple introduction on the support graph theory is followed in Section II.3. Finally, Section II.4 presents the second order elliptic boundary value problem and the finite element formulation.

II.1. Basic calculus

The material of this section is based on [32, 8]. We use $\mathbb{R}$ to denote the set of real numbers. A subset of $\mathbb{R}^d$ (d is an integer) is called a domain if it is open and connected. A bounded domain in $\mathbb{R}^d$ often is denoted as Ω, its boundary is denoted as $\partial\Omega$ or Γ. Let u be a scalar function, and $\mathbf{v} = (v_1, \cdots, v_d)$ a vector valued function, of $\mathbf{x} \in \mathbb{R}^d$. Then the *gradient*, the *divergence*, and the *Laplace* operator are defined as,

$$\begin{array}{ll}
\text{gradient} & \bigtriangledown u = grad\ u = (\dfrac{\partial u}{\partial x_1}, \cdots, \dfrac{\partial u}{\partial x_d}) \\
\text{divergence} & \bigtriangledown \cdot \mathbf{v} = div\ \mathbf{v} = \displaystyle\sum_{i=1}^{d} \frac{\partial v_i}{\partial x_i} \\
\text{Laplace} & \triangle u = \bigtriangledown \cdot \bigtriangledown u = \displaystyle\sum_{i=1}^{d} \frac{\partial^2 u}{\partial x_i^2}
\end{array}$$

A *multi-index* α is a d-vector $\alpha = (\alpha_1, \cdots, \alpha_d)$, where α_i's are non-negative integers. The length of a multi-index is defined as $|\alpha| = \sum_{i=1}^{d} \alpha_i$. A partial derivative of u of order $|\alpha|$ can be denoted as,

$$D^{\alpha} = \frac{\partial^{|\alpha|} u}{\partial x_1^{\alpha_1} \cdots \partial x_d^{\alpha_d}}.$$

$C(\Omega)$ denotes the linear space of continuous functions on Ω. $C^k(\Omega)$ denotes the set of k ($k \geq 0$) times continuously differentiable functions in Ω, i.e., let $u \in C^k(\Omega)$, then $\forall\ |\alpha| \leq k, D^{\alpha}u \in C(\Omega)$. We use $C_0^k(\Omega)$ to denote the set of functions $u \in C^k(\Omega)$ that vanish outside some compact subset of Ω.

A linear space of *square integrable* functions in Ω is denoted as $L_2(\Omega)$. Let $u, v \in L_2(\Omega)$, the space $L_2(\Omega)$ is equipped with the inner product and norm as,

$$\begin{aligned} (u, v) &= (u, v)_{L_2(\Omega)} = \textstyle\int_{\Omega} uvdx \\ \|u\| &= \|u\|_{L_2(\Omega)} = (\textstyle\int_{\Omega} u^2 dx)^{1/2} \end{aligned}$$

We say a function u belongs to the *Sobolev* space $H^k(\Omega)$ ($k \geq 1$) if $D^{\alpha}u \in L_2(\Omega)$ for all $|\alpha| \leq k$. The Sobolev space is equipped with the following norm and seminorm,

$$\begin{aligned} \|u\|_k &= \|u\|_{H^k(\Omega)} = (\sum_{|\alpha| \leq k} \|D^{\alpha}u\|^2)^{1/2} \\ |u|_k &= |u|_{H^k(\Omega)} = (\sum_{|\alpha| = k} \|D^{\alpha}u\|^2)^{1/2} \end{aligned}$$

Finally, we give two important formulas that are used in deriving the weak form for the finite element problems,

$$\begin{array}{ll} \text{divergence theorem} & \int_{\Omega} \bigtriangledown \cdot \mathbf{v} dx = \int_{\Gamma} \mathbf{v} \cdot \mathbf{n} ds \\ \text{Green's formula} & \int_{\Omega} \mathbf{v} \cdot \bigtriangledown u dx = \int_{\Gamma} \mathbf{v} \cdot \mathbf{n} u ds - \int_{\Omega} \bigtriangledown \cdot \mathbf{v} u dx \end{array}$$

II.2. Basic linear algebra

The material of this section is based on [41, 35, 37, 15]. Following is a collection of concepts and propositions that are important for the iterative methods. Notice that we consider only real square matrices and some definitions are adapted to our usage.

sparse matrix An n-by-n matrix A is *sparse* if the number of its nonzero entries is in the order of $O(n)$.

positive matrix A matrix A is *positive* if all its entries are positive, i.e. $A_{ij} > 0$.

M-matrix A matrix A is called an *M-matrix* if its diagonals are positive, its off-diagonals are nonpositive and its inverse is a nonnegative matrix.

Stieltjes matrix A symmetric M-matrix is called a *Stieltjes* matrix.

symmetric positive definite (SPD) Matrix A real matrix A is *symmetric positive definite*, if it is symmetric, i.e. $A^T = A$, and $x^T Ax > 0$ for any $x \in \mathbb{R}^n, x \neq 0$. If $x^T Ax \geq 0$ instead, then A is a *symmetric positive semidefinite*.

diagonally dominant An n-by-n matrix A is *(weakly) diagonally dominant* if $\forall i \in \{1, \cdots, n\}$, $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$. If at least one of the diagonal entries is strictly larger than the sum of the absolute value of the off-diagonal entries, then A is *strictly diagonally dominant*. Notice that a diagonally dominant symmetric M-matrix is a subset of *Stieltjes* matrices.

Laplacian matrix A diagonally dominant symmetric matrix with nonpositive off-diagonals and zero row sums is called a *Laplacian* matrix.

edge and vertex vectors A vector with two nonzero entries with the same magnitude in location i and j represents an *edge* between vertices i and j. A vector with one nonzero entry in location i represents a *vertex* i. The unit *(positive) edge vector*

$e_{\langle ij\rangle}$, *negative edge vector* $e^{-}_{\langle ij\rangle}$ and *vertex vector* $e_{\langle ii\rangle}$ are defined as,

$$e_{\langle ij\rangle} = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{bmatrix} \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \end{bmatrix}, \qquad e^{-}_{\langle ij\rangle} = \begin{array}{c} \\ i \\ \\ j \\ \\ \end{array} \begin{bmatrix} \vdots \\ 1 \\ \vdots \\ 1 \\ \vdots \end{bmatrix}, \qquad e_{\langle ii\rangle} = \begin{array}{c} \\ i \\ \\ \end{array} \begin{bmatrix} \vdots \\ 1 \\ \vdots \end{bmatrix}.$$

Proposition II.1 ([15]) *A symmetric matrix A is diagonally dominant if and only if there is a decomposition of the form $A = UU^T$, where each column of U is either a scaled positive edge vector, a scaled negative edge vector or a scaled vertex vector.*

Proposition II.2 ([15]) *A symmetric matrix A is a diagonally dominant M-matrix if and only if there is a decomposition of the form $A = UU^T$, where each column of U is either a scaled positive edge vector or a scaled vertex vector.*

Proposition II.3 ([15]) *A symmetric matrix A is a Laplacian if and only if there is a decomposition of the form $A = UU^T$, where each column of U is a scaled positive edge vector.*

eigenvalue λ is an *eigenvalue* of matrix A, if there exist a vector $x \neq 0$ such that $Ax = \lambda x$. And x is the corresponding *eigenvector* of eigenvalue λ.

spectrum The *spectrum* of A is defined as the set of all its eigenvalues, usually it is denoted by $\sigma(A)$.

Gershgorin's theorem Any eigenvalue of matrix A is located in one of the closed discs of the complex plane centered at A_{ii} and having the radius of the sum of the absolute value of the off-diagonal entries at row i, for $i = 1, \cdots, n$. That is,

$$\forall \lambda \in \sigma(A),\ \exists i \in \{1, \cdots, n\}, \quad \text{such that} \quad |\lambda - A_{ii}| \leq \sum_{j \neq i} |A_{ij}|.$$

condition number The *(spectral) condition number* of matrix A is defined as the product of the two-norm of A and A^{-1}:

$$\kappa(A) = \|A\|_2\|A^{-1}\|_2 = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}.$$

Here $\sigma(A)$ refers to the singular values of A. For an SPD matrix A, the condition number $\kappa(A)$ is equal to the ratio of the extreme eigenvalues of A:

$$\kappa(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}.$$

II.3. M-matrix and support graph

By generalizing a maximum weight spanning tree to a maximum weight basis ([15]), support graph preconditioning technique applies to symmetric diagonally dominant matrices. In this dissertation, we focus on support graph preconditioners that apply to symmetric diagonally dominant M-matrix. So all the support graph concepts are presented in the classical sense.

The construction of a support graph preconditioner is based on the combinatorial properties of the graph corresponding to the matrix of (1.1). One central point in the support graph preconditioning theory is the connection between a matrix and a graph. We associate a symmetric, diagonally dominant M-matrix $A \in \mathbb{R}^{n \times n}$ with a weighted undirected graph $G_A = (V_A, E_A)$. Each row of A defines a node in G_A, i.e., the i-th row defines the node i. The weight of vertex i is defined as the sum of elements in row i of A. Nonzero weight of a vertex can be seen as a self loop edge with the same weight. $V_A = \{1, 2, \ldots, n\}$ is the vertex set of G_A. Each off-diagonal entry of A defines an edge in G_A. $E_A = \{(i,j) : i \neq j, A_{ij} \neq 0\}$, with edge weight $w_{ij} = |A_{ij}|$, is the edge set of G_A. Shown in Fig. II.1 is an example of an M-matrix and its corresponding graph.

$$A = \begin{bmatrix} 5 & -3 & -1 & -1 \\ -3 & 4 & -1 & 0 \\ -1 & -1 & 5 & -2 \\ -1 & 0 & -2 & 3 \end{bmatrix}$$

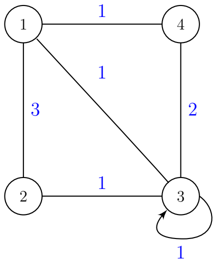


FIG. II.1. *An M-matrix and its corresponding graph.*

The support graph preconditioner considered in this work is a subgraph of the underlying graph of the coefficient matrix, so the following concepts are introduced under this premise. Let $E_M \subseteq E_A$, $G_M = (V_A, E_M)$. Consider a graph embedding of G_A into G_M, where each edge of G_A is mapped to a path in G_M. The *support path* of an edge $e \in G_A$ is a path $p \in G_M$ whose end nodes are the end nodes of e. G_M is a *support graph* of G_A if there exists a support path $p \in G_M$ for every edge $e \in G_A$. For a support path p, the *support path weight* w_p is defined as the weight of the edge e which is supported by p. For an edge $e \in G_A$, *edge dilation* δ_e is the number of edges in its support path. The *dilation* δ of G_A is the maximum edge dilation over all edges in G_A. For an edge $e \in G_M$, *edge congestion* c_e is the number of support paths passing through e. In weighted graphs, c_e is the sum of weight of the support paths through e divided by edge weight w_e. The *congestion* c of G_M is the maximum congestion over all edges in G_M.

The support graph preconditioner M is usually constructed in such a way that the row sums of M equal to those of A. As shown in Bern et al. [14], the analysis of bounding the extreme eigenvalues of $M^{-1}A$ can assume A and M having zero row sums, that is A and M are *Laplacians.* For a Laplacian, the edge weights of its underlying graph determines the matrix exactly. In order not to clutter the notations, from now on we may also refer to the underlying graph of matrix A as A itself if the context is clear.

The support graph preconditioning technique analyzes a matrix A and its preconditioner M in terms of their underlying graphs. By embedding the graph of A into the graph of M, the largest eigenvalue of $M^{-1}A$ can be bounded. Since a simple structure is easier to study, it often splits A and M into a sum of matrices, as $A = A_1 + \cdots + A_m$ and $M = M_1 + \cdots + M_m$. Each A_i and M_i are Laplacians themselves. Usually each A_i represents an edge of graph A with the entire edge weight, and M_i is a path of graph M where each edge may contain only a fraction of its entire edge weight. We say the edge A_i is *supported* by the path M_i if there exists a finite number τ_i, such that $\tau_i M_i - A_i$ is positive semidefinite. If every edge of A is supported by a path of M, then $\lambda_{max}(M^{-1}A) \leq \tau$, where $\tau = \max_i \tau_i$. In fact, τ is a central concept in the support theory called *support number* or simply *support.* It is specified by graph embedding properties such as congestion and dilation, which is given by what is called *congestion-dilation* lemma [24, 16, 14] in support graph preconditioning theory. The smallest generalized eigenvalue of $M^{-1}A$ is bounded in the same way by exchanging the roles of A and M. Thus the condition number of the preconditioned system is bounded. If a support graph M is a subgraph of A, we have the following lemma.

Lemma II.4 (Condition number bound of support graph preconditioning) *Suppose A and M are symmetric diagonally dominant M-matrices. M is a support graph of A, and the underlying graph of M is a subgraph of the underlying graph of A. Let δ be the largest dilation over all edges of A, and c be the largest congestion over all edges of M. Then,*

$$\kappa(M^{-1}A) \leq \delta \cdot c.$$

Proof This lemma is easy to be proved by using Congestion-Dilation Lemma. Here we prove it in a pure algebraic way. Without loss of generality, assume A and M are

Laplacians. For Laplacian matrices we have,

$$x^TAx = \sum_{A_{ij}\neq 0} |A_{ij}|(x_i - x_j)^2 = \sum_{(i,j)\in E_A} \omega_{ij}(x_i - x_j)^2,$$
$$x^TMx = \sum_{M_{ij}\neq 0} |M_{ij}|(x_i - x_j)^2 = \sum_{(i,j)\in E_M} \omega_{ij}(x_i - x_j)^2.$$

Where ω_{ij} is the absolute value of the off-diagonal entry $A_{ij}, i, j = 1, \cdots, n$. Notice that M is a subgraph of A, i.e., $E_M \subseteq E_A$. Thus,

$$\frac{x^TAx}{x^TMx} = \frac{\sum\limits_{(i,j)\in E_A} \omega_{ij}(x_i - x_j)^2}{\sum\limits_{(m,n)\in E_M} \omega_{mn}(x_m - x_n)^2} = 1 + \frac{\sum\limits_{(i,j)\in E_A\setminus E_M} \omega_{ij}(x_i - x_j)^2}{\sum\limits_{(m,n)\in E_M} \omega_{mn}(x_m - x_n)^2}.$$

It is obvious that,

$$\frac{x^TAx}{x^TMx} \geq 1. \tag{2.1}$$

For every edge $(i, j) \in E_A$, there exists a support path $p_{ij} \in E_M$. Suppose the path has $\rho + 1$ edges and consists of the following nodes $x_i, x_{s_1}, \cdots, x_{s_\rho}, x_j$, i.e., the dilation of (i, j) is $\delta_{ij} = \rho + 1$. We have,

$$(x_i - x_j)^2 = \left[(x_i - x_{s_1}) + (x_{s_1} - x_{s_2}) + \cdots + (x_{s_\rho} - x_j)\right]^2 \leq \delta_{ij} \sum_{(k,l)\in p_{ij}} (x_k - x_l)^2.$$

Then,

$$\begin{aligned}\sum_{(i,j)\in E_A} \omega_{ij}(x_i - x_j)^2 &\leq \sum_{(i,j)\in E_A} \omega_{ij}\delta_{ij} \sum_{(k,l)\in p_{ij}} (x_k - x_l)^2 \\ &= \sum_{(i,j)\in E_A} \sum_{(k,l)\in p_{ij}} \omega_{ij}\delta_{ij}(x_k - x_l)^2 \\ &= \sum_{(k,l)\in E_M} \sum_{\substack{\forall (i,j)\in E_A \\ s.t.\ (k,l)\in p_{ij}}} \omega_{ij}\delta_{ij}(x_k - x_l)^2.\end{aligned}$$

The relationship above gives,

$$
\begin{aligned}
\frac{x^T A x}{x^T M x} &= \frac{\displaystyle\sum_{(i,j)\in E_A} \omega_{ij}(x_i - x_j)^2}{\displaystyle\sum_{(m,n)\in E_M} \omega_{mn}(x_m - x_n)^2} \\
&\le \frac{\displaystyle\sum_{(k,l)\in E_M} \sum_{\substack{\forall (i,j)\in E_A \\ s.t.\ (k,l)\in p_{ij}}} \omega_{ij}\delta_{ij}(x_k - x_l)^2}{\displaystyle\sum_{(m,n)\in E_M} \omega_{mn}(x_m - x_n)^2} \\
&= \frac{\displaystyle\sum_{(k,l)\in E_M} \omega_{kl}(x_k - x_l)^2 \sum_{\substack{\forall (i,j)\in E_A \\ s.t.\ (k,l)\in p_{ij}}} \frac{\omega_{ij}}{\omega_{kl}}\delta_{ij}}{\displaystyle\sum_{(m,n)\in E_M} \omega_{mn}(x_m - x_n)^2} \\
&\le \max_{(k,l)\in E_M} \sum_{\substack{\forall (i,j)\in E_A \\ s.t.\ (k,l)\in p_{ij}}} \frac{\omega_{ij}}{\omega_{kl}}\delta_{ij} \\
&\le \delta \cdot \max_{(k,l)\in E_M} \sum_{\substack{\forall (i,j)\in E_A \\ s.t.\ (k,l)\in p_{ij}}} \frac{\omega_{ij}}{\omega_{kl}} \\
&\le \delta \cdot \max_{(k,l)\in E_M} c_{kl} \\
&\le \delta \cdot c.
\end{aligned}
\tag{2.2}
$$

By the results of (2.1) and (2.2), and the definition of the condition number, we have,

$$
\kappa(M^{-1}A) = \frac{\lambda_{\max}(M^{-1}A)}{\lambda_{\min}(M^{-1}A)} = \frac{\displaystyle\max_{\|x\|=1} \frac{x^T A x}{x^T M x}}{\displaystyle\min_{\|x\|=1} \frac{x^T A x}{x^T M x}} \le \frac{\delta \cdot c}{1} = \delta \cdot c. \qquad \blacksquare
$$

II.4. The second order elliptic finite element problem

The finite element method is one of the most popular and effective numerical techniques for solving partial differential equations (PDEs), especially over complex domains. The finite element discretization of PDEs often gives a large sparse linear system. The associated coefficient matrix is symmetric positive definite for the second order self-adjoint elliptic boundary value problems. This linear system can be solved by iterative methods such as PCG.

The equation of the second order elliptic boundary value problem is,

$$-\bigtriangledown \cdot (K(\mathbf{x}) \bigtriangledown u) + q(\mathbf{x})u = f(\mathbf{x}) \text{ in } \Omega, \qquad u(\mathbf{x}) = 0 \text{ on } \partial\Omega. \tag{2.3}$$

where Ω is a bounded open set of $\mathbb{R}^d, d = 2, 3$. $K(\mathbf{x})$ is a $d \times d$ symmetric matrix that is uniformly bounded positive definite in Ω, i.e. there are positive constants γ, Γ, such that $\gamma|u|^2 \leq u^T K(\mathbf{x})u \leq \Gamma|u|^2$ for all $u \in \mathbb{R}^d$ and for all $x \in \Omega$. $q(\mathbf{x})$ is nonnegative and is bounded as $0 \leq q(\mathbf{x}) \leq Q$. For simplicity, we also assume $f(\mathbf{x})$ is square integrable in Ω, i.e., $f(\mathbf{x}) \in L_2(\Omega)$. Notice that the use of homogeneous Dirichlet boundary condition only simplifies the following presentation, more complicated boundary conditions can be imposed without any problem.

A weak formulation of this problem is, find $u \in H_0^1(\Omega)$, such that

$$a(u, v) = (f, v), \ \forall v \in H_0^1(\Omega). \tag{2.4}$$

where

$$a(u, v) = \int_\Omega K(\mathbf{x}) \bigtriangledown u \cdot \bigtriangledown v + q(\mathbf{x})uvdx, \quad (f, v) = \int_\Omega fvdx.$$

According to Galerkin finite element procedure, we are looking for an approximate solution $u_h \in V_h$, where V_h is a finite dimensional subspace of $H_0^1(\Omega)$. Typically, V_h is a space of piecewise polynomials associated with a mesh $\mathcal{T}$ of Ω. The finite element approximation of problem (2.4) is to find $u_h \in V_h$, such that

$$a(u_h, v) = (f, v), \ \forall v \in V_h. \tag{2.5}$$

Choose a suitable basis $\{\phi_i\}, i = 1, \cdots, N$ for V_h. Let $u_h = \sum_{i=1}^{N} U_i \phi_i$, then (2.5) gives a large sparse linear system of equations for $U = [U_1, \cdots, U_N]^T$,

$$AU = b. \tag{2.6}$$

where $A_{ij} = a(\phi_i, \phi_j)$ and $b_i = (f, \phi_i)$. Since $a(u, v)$ is symmetric and coersive, the discretized matrix A is SPD. Thus, (2.6) can be solved by PCG. Note that, in general A is not diagonally dominant (see, e.g., [8]). For example, in Appendix A, we show that the matrix obtained from linear triangular element discretization of the Poisson problem is SPD in general. The readers can become familiar with the above finite element formulation and discretization procedure by reading any standard textbook (e.g. [32]).

CHAPTER III

CONSTRUCT M-MATRIX PRECONDITIONERS

The largest class of matrices that the support graph preconditioning technique can apply to is the symmetric diagonally dominant matrices. This chapter extends the support graph preconditioning to the SPD matrices arising from the second order elliptic finite element problems. First we review other works in this area; then we present our approaches to transform these SPD matrices into symmetric diagonally dominant M-matrices, starting from the simple Poisson model problem, incrementally to the full fledged second order elliptic boundary value problem.

III.1. Other works and proposal

One direction of extension of support graph preconditioning is the finite element problems. The basic idea is to approximate the stiffness matrix A (which is symmetric positive definite for the second order self-adjoint elliptic problems) by a symmetric diagonally dominant M-matrix A' first, then construct support graph preconditioner M for A' in the usual way. In the end, M is used as a preconditioner for A. Thus, the important issue is to keep the condition number $\kappa(A'^{-1}A)$ as low as possible.

For Poisson problems, Boman et al. [17] gives a general algebraic approach. Their approach is general, in the sense, that it applies to any order Lagrange elements, any number of Gauss quadrature points; and it allows jump in the conductivity coefficient. They write the stiffness matrix in the form of $A = B^T J^T DJB$, where B is a node-arc incidence matrix (a block of B represents a *star* graph), J is a block diagonal matrix related to the inverse Jacobians and the gradient matrix of the reference element, D is a diagonal matrix related to the conductivity coefficient, quadrature weights and the determinant of Jacobians. For a well-shaped mesh, J is well-conditioned; if in addition, the jump in coefficient occurs only along element boundaries, then D is well-conditioned as well. By choosing a diagonal matrix $\tilde{D}$, let $\tilde{J} = D^{1/2} J \tilde{D}^{-1/2}$, such that $\kappa(\tilde{J})$ is minimized or close to minimized, they use the M-matrix $A' = B^T \tilde{D} B$ to precondition A. And $\kappa(A'^{-1}A) \leq \kappa(\tilde{J})^2$ is well bounded. The limitation, however, is that their approach applies to isotropic problems only.

The idea of another approach by Avron et al. [7] is to split the elements into two subsets $E(t)$ and $\overline{E}(t)$. Let $A_E = \sum_{e \in E(t)} A_e$ and $A_{\overline{E}} = \sum_{e \in \overline{E}(t)} A_e$, then the global stiffness matrix is $A = A_E + A_{\overline{E}}$. $E(t)$ consists all elements whose element matrix A_e is approximable by a diagonally dominant matrix L_e, $\overline{E}(t)$ consists all the rest. t is a threshold parameter: if $\kappa(L_e^{-1}A_e) \leq t$, then $e \in E(t)$; otherwise $e \in \overline{E}(t)$. The global approximate matrix L for A_E is constructed as $L = \sum_{e \in E(t)} \alpha_e L_e$, where $\{\alpha_e\}$'s are scaling factors, they are chosen such that $\kappa(L^{-1}A_E) \leq \max_{e \in E(t)} \kappa(L_e^{-1}A_e)$. Since L is a symmetric, diagonally dominant matrix, it can be approximated by a support graph preconditioner M. In the end, the original global matrix A will be preconditioned by $A' = \gamma M + A_{\overline{E}}$, and γ is another scaling factor such that $\kappa(A'^{-1}A) \leq \kappa(M^{-1}A_E)$. The major operation of constructing L_e for elements in $E(t)$ is *column scaling*, that basically is to construct a weighted *clique* graph. Using the result due to van der Sluis [43], the constructed L_e approximates A_e within a factor of $\sqrt{n_e}$ of the optimal scaling (where n_e is the dimension of A_e). The catch, however, is that even for the optimal scaling the condition number of $L_e^{-1}A_e$ can be large.

The solver of Avron et al. [7] is general in that, it has no requirement of meshes, and it can deal with anisotropic problems. It divides elements into two subsets $E(t)$ and $\overline{E}(t)$, the matrix of one subset is approximated by a symmetric diagonally dominant matrix L, the matrix of the other subset is left as is (either due to distorted element or due to anisotropy). So the preconditioner totally embraces everything of the ‘bad’ elements. However, the benefit of easy factorization of support graph M which is constructed based on L diminishes due to the addition of $A_{\overline{E}}$. When the problem becomes ill-conditioned, more and more elements would be thrown into $\overline{E}(t)$, the cost of Cholesky factorization of the preconditioner becomes close to that of the original matrix; eventually, a direct solver takes over. As a practical approach this may regarded as having a grace transition from an iterative solver to a direct solver.

In contrast of these two approaches, which are more algebraic and abstract, our approach is more geometric and intuitive. Our idea is to use element level transformations to construct symmetric diagonally dominant M-matrices that can be used to approximate the coefficient matrix. The basic procedure is to approximate the element gradient matrix by taking the gradients along chosen edges, whose unit vectors form a new coordinate system. For Lagrangian elements, the rows of the element gradient matrix in this new coordinate system are scaled edge vectors, thus a diagonally dominant symmetric semidefinite M-matrix can be generated to approximate the element stiffness matrix. Depending on the element type, one or more such coordinate systems are required to obtain a global nonsingular M-matrix. In terms of graph, this can be seen that the element matrix, which is usually a *clique*, is approximated by a union of *star* graphs. We show that this technique of element coordinate transformations applies to a variety of first order Lagrangian elements. Combination of this technique and other techniques enables us to construct an M-matrix preconditioner for a wide range of second order elliptic problems even with higher order elements.

In order to prevent cluttering our coordinate transformation idea, in the following, we start from a simple Poisson model problem and then progressively to more difficult problems.

III.2. Poisson model problem

By setting $K(\mathbf{x}) = 1, q(\mathbf{x}) = 0$ in (2.3), we obtain the Poisson model problem. Let's consider Lagrangian elements. Assume the mesh $\mathcal{T}$ is in d dimensional space $\mathbf{x} = \{x_1, \cdots, x_d\}$. Let $\{\phi_i^e\}$'s be the local nodal bases for an element e, and n_e the number of nodal bases. Then the solution on element e is given as,

$$u_e = \boldsymbol{\phi}^e \cdot \mathbf{u}^e = \sum_{i=1}^{n_e} u_i^e \phi_i^e.$$

where u_i^e is the nodal solution at node i. The element stiffness matrix and element load vector are,

$$A_e(i,j) = \int_e \nabla \phi_i^e \cdot \nabla \phi_j^e d\mathbf{x}, \quad b_e(i) = \int_e f \phi_i^e d\mathbf{x}.$$

The global stiffness matrix A and the load vector b are assembled, respectively, as,

$$A = \sum_{e \in \mathcal{T}} A_e, \qquad b = \sum_{e \in \mathcal{T}} b_e.$$

If we denote the element gradient matrix of e as,

$$B_e = \begin{bmatrix} \phi_{1,x_1}^e & \phi_{2,x_1}^e & \cdots & \phi_{n_e,x_1}^e \\ \phi_{1,x_2}^e & \phi_{2,x_2}^e & \cdots & \phi_{n_e,x_2}^e \\ \vdots & & & \\ \phi_{1,x_d}^e & \phi_{2,x_d}^e & \cdots & \phi_{n_e,x_d}^e \end{bmatrix}.$$

then,

$$A = \sum_{e \in \mathcal{T}} A_e = \sum_{e \in \mathcal{T}} \int_e B_e^T B_e d\mathbf{x}. \tag{3.1}$$

In general, A is SPD but not diagonally dominant, which prevents construction of a support graph preconditioner. In the following, we show that, by taking the gradients of the shape functions along our chosen edges, the element matrix A_e can be approximated by a Laplacian matrix. Assembling these approximate matrices results in a global diagonally dominant symmetric M-matrix that can be used to approximate A.

III.2.1. Triangular and tetrahedral elements

In two dimensional (2D) space, the simplicial element is the linear triangular element. The element gradient matrix B_e is,

$$B_e = \begin{bmatrix} \phi^e_{1,x} & \phi^e_{2,x} & \phi^e_{3,x} \\ \phi^e_{1,y} & \phi^e_{2,y} & \phi^e_{3,y} \end{bmatrix}.$$

where $\phi^e_{j,x}$ and $\phi^e_{j,y}$ denote the partial derivatives of shape function ϕ^e_j along x-axis and y-axis, respectively.

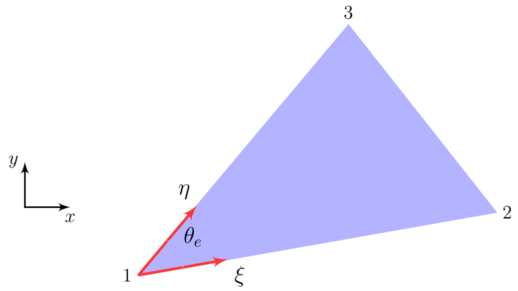


Fig. III.1. *A triangular element with a new coordinate system.*

Suppose we choose two adjacent edges with directions pointing away from their common node as shown in Fig. III.1, where the angle between them is θ_e. We use $\mathbf{l}_{ij}$ to denote the vector from node i to node j. Let l_{ij} be the lengths of $\mathbf{l}_{ij}$. Apparently, the unit vectors of $\mathbf{l}_{12}$ and $\mathbf{l}_{13}$ form a new coordinate system $\{\xi, \eta\}$ which conforms to the right hand rule. We define a new gradient matrix B'_e, whose rows are the partial derivatives of shape functions along these chosen edges,

$$B'_e = \begin{bmatrix} \phi^e_{1,\xi} & \phi^e_{2,\xi} & \phi^e_{3,\xi} \\ \phi^e_{1,\eta} & \phi^e_{2,\eta} & \phi^e_{3,\eta} \end{bmatrix} = \begin{bmatrix} l_{12}^{-1} & 0 \\ 0 & l_{13}^{-1} \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}.$$

Notice that each row of B'_e is a scaled edge vector, which is the key to construct an M-matrix. These two gradient matrices B_e and B'_e have the following relationship:

$$B'_e = \begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix} B_e = G_e B_e.$$

where G_e is the element Jacobian,

$$G_e = \frac{\partial(x,y)}{\partial(\xi,\eta)} = \begin{bmatrix} (x_2 - x_1)/l_{12} & (y_2 - y_1)/l_{12} \\ (x_3 - x_1)/l_{13} & (y_3 - y_1)/l_{13} \end{bmatrix} = \begin{bmatrix} \mathbf{l}_{12}^T/l_{12} \\ \mathbf{l}_{13}^T/l_{13} \end{bmatrix}.$$

Recall (3.1), we have matrix A assembled as,

$$A = \sum_{e \in \mathcal{T}} A_e = \sum_{e \in \mathcal{T}} \int_e B_e^T B_e d\mathbf{x}.$$

An approximation of A can be constructed as,

$$A' = \sum_{e \in \mathcal{T}} A'_e = \sum_{e \in \mathcal{T}} \int_e \omega_e B_e'^T B'_e d\mathbf{x}.$$

where ω_e is a scaling factor that is chosen in such a way that the condition number of $A'^{-1}A$ can be bounded elementwise. Since gradient matrix B'_e is constant, A' can be written as,

$$A' = \sum_{e \in \mathcal{T}} \omega_e B_e'^T B'_e \Delta_e,$$

where Δ_e denotes the area of element e. Note that A' is a symmetric diagonally dominant M-matrix, and it is more sparse than the original matrix A. Since A and

A' are SPD,

$$\lambda_{\min}(A'^{-1}A) \leq \frac{x^TAx}{x^TA'x} \leq \lambda_{\max}(A'^{-1}A),$$

for all nonzero x. To estimate the condition number of $A'^{-1}A$, we need to compute bounds on $x^TAx/x^TA'x$. Observe that,

$$x^TAx = \sum_{e\in\mathcal{T}} x^TA_ex = \sum_{e\in\mathcal{T}} \mu_e(x)x^TA'_ex,$$

where,

$$\mu_e(x) = \begin{cases} \dfrac{x^TA_ex}{x^TA'_ex}, & A_ex \neq 0. \\ \quad 1, & \text{otherwise.} \end{cases}$$

Notice that the above formula uses the fact that A_e and A'_e have the same null space. Therefore,

$$\min_{e\in\mathcal{T}} \mu_e(x) \sum_{e\in\mathcal{T}} x^TA'_ex \leq x^TAx \leq \max_{e\in\mathcal{T}} \mu_e(x) \sum_{e\in\mathcal{T}} x^TA'_ex,$$

which implies that,

$$\kappa(A'^{-1}A) = \frac{\max\limits_x \dfrac{x^TAx}{x^TA'x}}{\min\limits_x \dfrac{x^TAx}{x^TA'x}} \leq \max_x \left\{ \frac{\max\limits_{e\in\mathcal{T}} \mu_e(x)}{\min\limits_{e\in\mathcal{T}} \mu_e(x)} \right\}.$$

Considering that for $A_ex \neq 0$,

$$\frac{x^TA_ex}{x^TA'_ex} = \frac{1}{\omega_e}\frac{x^TB_e^TB_ex}{x^TB_e^TG_e^TG_eB_ex} = \frac{1}{\omega_e}\frac{y^Ty}{y^TG_e^TG_ey},$$

where $y = B_ex$. By choosing $\omega_e = 1/\lambda_{\min}(G_e^TG_e)$, we ensure that,

$$\frac{\lambda_{\min}(G_e^TG_e)}{\lambda_{\max}(G_e^TG_e)} \leq \frac{x^TA_ex}{x^TA'_ex} \leq 1.$$

Thus,

$$\frac{1}{\kappa(G_e^TG_e)} \leq \mu_e(x) \leq 1.$$

which gives an estimate for the spectral condition number of $A'^{-1}A$:

$$\kappa(A'^{-1}A) \leq \max_{e\in\mathcal{T}} \kappa(G_e^T G_e) = \max_{e\in\mathcal{T}} \kappa(G_e G_e^T).$$

Notice that $G_e^T G_e$ has the same eigenvalues as $G_e G_e^T$. The outer-product of the element Jacobians is,

$$G_e G_e^T = \begin{bmatrix} \mathbf{l}_{12}^T/l_{12} \\ \mathbf{l}_{13}^T/l_{13} \end{bmatrix} \begin{bmatrix} \mathbf{l}_{12}/l_{12} & \mathbf{l}_{13}/l_{13} \end{bmatrix} = \begin{bmatrix} 1 & \cos\theta_e \\ \cos\theta_e & 1 \end{bmatrix}. \tag{3.2}$$

It is obvious that a necessary condition for $G_e G_e^T$ to be singular is when $1-\cos^2\theta_e = 0$, i.e., $\sin\theta_e = 0$. This is impossible for a triangulation mesh, otherwise the area of a triangle becomes zero. The condition number of $A'^{-1}A$ can be written explicitly as,

$$\kappa(A'^{-1}A) \leq \max_{e\in\mathcal{T}} \kappa(G_e G_e^T) = \max_{e\in\mathcal{T}} \frac{1+|\cos\theta_e|}{1-|\cos\theta_e|}. \tag{3.3}$$

This indicates that the quality of the approximation can be improved by selecting a pair of edges for which $|\cos\theta_e|$ is as small as possible. For example, if $\theta_e = 90^o$, then the element matrix does not need transformation. For two dimensional good quality meshes, usually $\pi/8 \leq \theta_e \leq 7\pi/8$ (see, e.g., [48]), which implies that $\kappa(A^{-\prime}A) \leq 26$. In practice, it is observed that the best angle in an element usually satisfies $\pi/4 \leq \theta_e \leq 3\pi/4$. For example, for meshes used in our experiments, $\max_{e\in\mathcal{T}} \kappa(G_e G_e^T)$ is less than 8.

This procedure can be generalized in three dimensional (3D) space for the tetrahedral elements (see Fig. III.2). Suppose three chosen adjacent edges are the vectors $\mathbf{l}_{12}, \mathbf{l}_{13}, \mathbf{l}_{14}$, from which a new coordinate system $\{\xi, \eta, \zeta\}$ can be formed. Let Δ_e denote the volume of a tetrahedron e. We can approximate the gradient matrix B_e by

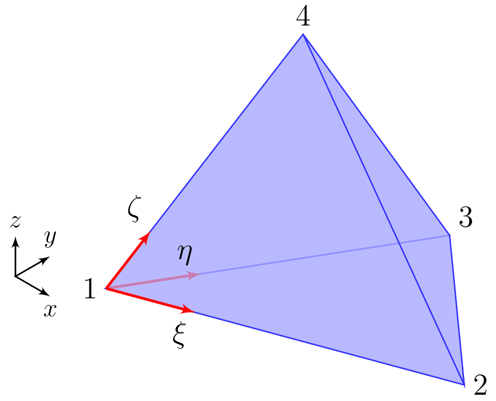


FIG. III.2. *A tetrahedral element with a new coordinate system.*

the following matrix B'_e,

$$B'_e = \begin{bmatrix} l_{12}^{-1} & 0 & 0 \\ 0 & l_{13}^{-1} & 0 \\ 0 & 0 & l_{14}^{-1} \end{bmatrix} \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

By coordinate transformation we have $B'_e = G_e B_e$, and

$$G_e = \frac{\partial(x,y,z)}{\partial(\xi,\eta,\zeta)} = \begin{bmatrix} \mathbf{l}_{12}^T/l_{12} \\ \mathbf{l}_{13}^T/l_{13} \\ \mathbf{l}_{14}^T/l_{14} \end{bmatrix}.$$

As what is done in the two dimensional case, the stiffness matrix A is approximated by the following M-matrix A',

$$A' = \sum_{e\in\mathcal{T}} \omega_e B_e'^T B'_e \Delta_e.$$

Again, take $\omega_e = 1/\lambda_{\min}(G_e G_e^T)$, it can be shown that the condition number of $A'^{-1}A$ is bounded as

$$\kappa(A'^{-1}A) \le \max_{e\in\mathcal{T}} \kappa(G_e G_e^T). \tag{3.4}$$

Expand the outer-product of the element Jacobians, we have,

$$G_e G_e^T = \begin{bmatrix} 1 & \cos\theta_e^{1,23} & \cos\theta_e^{1,24} \\ \cos\theta_e^{1,23} & 1 & \cos\theta_e^{1,34} \\ \cos\theta_e^{1,24} & \cos\theta_e^{1,34} & 1 \end{bmatrix}. \tag{3.5}$$

where $\theta_e^{i,jk}$ denotes the angle at node i that faces the edge jk in element e. Define $V(\theta_1, \theta_2, \theta_3)$ as,

$$V(\theta_1, \theta_2, \theta_3) \triangleq \sqrt{1 + 2\cos\theta_1 \cos\theta_2 \cos\theta_3 - \cos^2\theta_1 - \cos^2\theta_2 - \cos^2\theta_3}.$$

A necessary condition for $G_e G_e^T$ to be singular is when $V(\theta_e^{1,23}, \theta_e^{1,34}, \theta_e^{1,24}) = 0$. This is impossible for a tetrahedron, since the volume of a tetrahedron can be expressed as $\Delta_e = \frac{l_{12} l_{13} l_{14}}{6} V(\theta_e^{1,23}, \theta_e^{1,34}, \theta_e^{1,24})$, which would be zero under such condition. From (3.5) it is not difficult to get the inverse of $G_e G_e^T$. Using Gerschgorin theorem we can estimate,

$$\lambda_{\max}(G_e G_e^T) \le 3, \quad \lambda_{\max}((G_e G_e^T)^{-1}) \le \frac{5}{V^2(\theta_e^{1,23}, \theta_e^{1,34}, \theta_e^{1,24})}.$$

The condition number of $G_e G_e^T$ is bounded as,

$$\kappa(G_e G_e^T) \le \frac{15}{V^2(\theta_e^{1,23}, \theta_e^{1,34}, \theta_e^{1,24})}. \tag{3.6}$$

Applying (3.4) we get,

$$\kappa(A'^{-1} A) \le \max_{e \in \mathcal{T}} \frac{15}{V^2(\theta_e^{1,23}, \theta_e^{1,34}, \theta_e^{1,24})}. \tag{3.7}$$

Notice that the condition number bound of $A'^{-1}A$ is entirely determined by the mesh topology parameters, i.e., the angles between edges. An ideal case is when these three adjacent edges are perpendicular to each other, thus $\lambda_{\max}(G_e G_e^T) = \lambda_{\min}(G_e G_e^T) = 1$. The quality of the approximation $A' \sim A$ can be improved by selecting three adjacent edges for which $\kappa(G_eG_e^T)$ is as small as possible.

The relationships of (3.3) and (3.7) show that, the spectral distance between the constructed M-matrix A' and the original matrix A is bounded by the mesh characteristics. If the mesh is 'good' then the approximation is good. However, there is no universal mesh quality metric, since it is application dependent. What kind of mesh is a good mesh for our application? As many iterative methods have guaranteed convergence for symmetric diagonally dominant matrices, which may suggest that a mesh that produces a matrix that is spectrally close to a symmetric diagonally dominant matrix is a good mesh. From this point of view, our results may provide a quality metric for the simplicial elements — the condition number of the outer-product of a chosen element Jacobian $\kappa(G_eG_e^T)$. Usually the mesh quality metrics are based on geometric criteria, such as element volumes, aspect ratio, skew, angles, etc., the work by Knupp [31] pointed out that many geometric characteristics are actually embedded in the Jacobian matrix. Knupp used a weighted Jacobian matrix, which is nodally invariant, to define many quality metrics for the simplicial elements. What should be emphasized is that, the requirement on the mesh quality of (3.3) and (3.7) is not severe. For a triangular element, there can be three different Jacobian matrices, as the origin of the new coordinate system can anchor at three different vertex nodes; for a tetrahedral element, there are four different Jacobian matrices. In order to get a good approximation A', only one Jacobian G_e is required such that $\kappa(G_eG_e^T)$ is well bounded.

The results of (3.2) and (3.5) clearly say that the mesh quality is specified by some element angle(s). However, the angle relationship in 3D is pretty involved. In fact, by estimating $\kappa(G_eG_e^T)$, the 2D and 3D cases can be unified which gives a more intuitive interpretation of these bounds. The bound for the 3D case is estimated already in (3.6). For the 2D case, (3.2) can be estimated as,

$$\kappa(G_eG_e^T) = \frac{1+|\cos\theta_e|}{1-|\cos\theta_e|} = \frac{(1+|\cos\theta_e|)^2}{\sin^2\theta_e} \leq \frac{4}{V^2(\theta_e)}. \tag{3.8}$$

where $V(\theta) = \sin\theta$. Recall that the triangular area can be expressed as $\Delta_e = \frac{l_{12}l_{13}}{2}V(\theta_e)$. Then from (3.8) and (3.6) we may say that, the quality of a simplicial element is measured by a dimensionless volume $V(\cdot)$, which is also the determinant of the Jacobian matrix G_e. Actually $V(\cdot)$ can be seen as the element volume normalized by a scaled product of the chosen edges' length. If the 'ideal' element is the one with $V(\cdot) = 1$, then the more close $V(\cdot)$ is to 1, the better is the quality of the element. Finally, the mesh quality would be measured by the overall quality of its elements, which give the spectral bounds for $\kappa(A'^{-1}A)$.

TABLE III.1
The number of iterations of PCG using preconditioner A' for linear triangular elements.

# elements	822	3,387	13,282	52,968	212,461	848,231	3,392,190
# unknowns	444	1,755	6,766	26,732	106,720	425,102	1,698,059
# iterations	7	9	10	10	11	10	11

TABLE III.2
The number of iterations of PCG using preconditioner A' for linear tetrahedral elements.

# elements	1,592	6,125	24,642	96,655	385,192	1,535,257	6,132,234
# unknowns	407	1,309	4,683	17,093	64,983	251,283	985,304
# iterations	9	10	10	10	10	9	9

To conclude this section, we apply this coordinate transformation approach to the Poisson problem with Dirichlet boundary condition. In two dimensional space, we consider the classical unit square domain, and the triangular mesh is generated by

Triangle [38]. In three dimensional space, we consider a unit cube domain, and the tetrahedral mesh is generated by *Tetgen* [39]. Shown in Table III.2.1 are the number of iterations for the PCG method applied to the linear triangular elements, where the stopping criteria is the two-norm of the relative residual less than 10^{-6}. The results indicate that, using matrix $A' = \sum \omega_e B_e'^T B_e' \Delta_e$ as the preconditioner for the original stiffness matrix A, the number of iterations is small and is independent of the problem size. This confirms the theoretical analysis of the condition number $\kappa(A'^{-1}A)$. The same conclusion holds for the tetrahedral elements as shown in Table III.2.1.

III.2.2. Quadrilateral and hexahedral finite elements

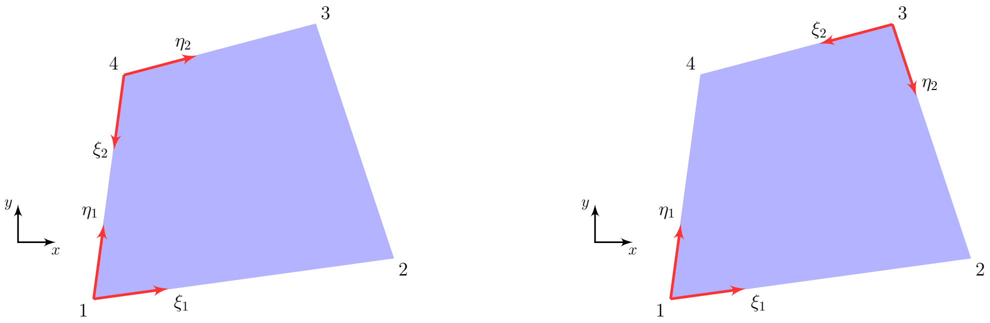


FIG. III.3. *Two cases of a quadrilateral with new coordinate systems.*

Though less flexible as triangular elements in many cases, quadrilateral elements often gives accurate results for problems that has certain directions such as problems in fluid dynamics. The simplest bases of a quadrilateral element are bilinear functions. In an element e, a bilinear function is generally quadratic along a line that is not aligned with an edge. This complication does not pose a big problem for our approach. As for the linear triangular element, we certainly can choose a pair of edges that defines a new coordinate system, and due to the property of Lagrangian shape functions, the derivatives of shape functions along these edges will form a gradient matrix whose rows are scaled edge vectors. An M-matrix A' can be constructed to approximate the stiffness matrix A. However, the problem is that the resulting A' can be singular. By examining the construction at the element level, we can see that one shape function has all zero derivatives in this new gradient matrix B'_e, which is the source of the singularity of A'. Suppose edges sharing node 1 are chosen, then the new gradient matrix B'_e is,

$$B'_e = \begin{bmatrix} l_{12}^{-1} & 0 \\ 0 & l_{14}^{-1} \end{bmatrix} \cdot \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}.$$

The gradient of shape function ϕ_3^e is zero along all the chosen edges, which may lead to the singularity of A'. To prevent this singularity, two pairs of edges are chosen to define two new coordinate systems (two choices are shown in Fig. III.3), and construct A' as,

$$A' = \sum_{e\in\mathcal{T}} A'_e = \sum_{e\in\mathcal{T}} \frac{1}{2}\left(\int_e \omega_{e,1} B'^T_{e,1} B'_{e,1} d\mathbf{x} + \int_e \omega_{e,2} B'^T_{e,2} B'_{e,2} d\mathbf{x}\right).$$

where,

$$\begin{aligned} B'_{e,1} &= G_{e,1}B_e, & B'_{e,2} &= G_{e,2}B_e, \\ G_{e,1} &= \frac{\partial(x,y)}{\partial(\xi_1,\eta_1)}, & G_{e,2} &= \frac{\partial(x,y)}{\partial(\xi_2,\eta_2)}, \\ \omega_{e,1} &= 1/\lambda_{\min}(G_{e,1}G_{e,1}^T), & \omega_{e,2} &= 1/\lambda_{\min}(G_{e,2}G_{e,2}^T). \end{aligned}$$

It is not difficult to get the condition number bound:

$$\kappa(A'^{-1}A) \le \max_{e\in\mathcal{T}} \max_{i\in\{1,2\}} \kappa(G_{e,i}G_{e,i}^T) = \max_{e\in\mathcal{T}} \max_{i\in\{1,2\}} \frac{1+|\cos\theta_e^i|}{1-|\cos\theta_e^i|}.$$

where θ_e^i is the angle between the ith pair of the chosen edges in element e. The strategy is to choose such θ_e^is that are as close to $\pi/2$ as possible. Experiments are conducted of the Poisson problem with Dirichlet boundary condition on a unit circle domain, and the quadrilateral mesh is generated by Gambit[1]. Table III.2.2 shows the number of iterations of the PCG using A' as preconditioner for the bilinear quadrilateral elements, and the stopping criteria is the two-norm of the relative residual less than 10^{-6}. This result shows that our M-matrix preconditioner is effective in keeping the number of iterations low and independent of the size of the problem.

TABLE III.3

The number of iterations of PCG using preconditioner A' for quadrilateral elements.

# elements	89	380	1,418	5,427	20,363	82,009	312,318
# unknowns	106	413	1,482	5,554	20,616	82,513	313,325
# iterations	10	10	10	10	10	10	10

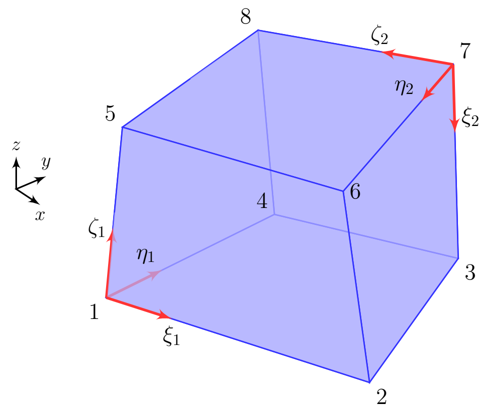


FIG. III.4. *A hexahedral element with new coordinate systems.*

This approach can easily be generalized to three dimensional hexahedral elements. To prevent the singularity of the constructed M-matrix, At least two pairs of new coordinate systems are needed, an example is shown in Fig. III.4. The choice

[1] Gambit 2.3.16, Fluent Inc.

on these new coordinate systems may depend on the application, and it may affect the work required for stiffness matrix assembly, the sparsity of the transformed M-matrix, the work required for Cholesky factorization and the convergence rate of the PCG algorithm. Numerical experimental results show that two coordinate systems are adequate enough to construct an effective M-matrix preconditioner, which is also the most sparse. Table III.2.2 shows the result of applying this strategy to the three dimensional Poisson problem with Dirichlet boundary condition, where two coordinate systems are selected in each element. The problem domain is a cylinder, with unit length for both radius and height. The mesh is generated by Gambit[2]. The PCG stopping criteria is the same like before.

TABLE III.4

The number of iterations of PCG using preconditioner A' for hexahedral elements.

# elements	445	3,800	28,360	217,080
# unknowns	636	4,543	31,122	227,714
# iterations	14	13	13	13

The generalization to other types of Lagrangian elements does not pose any difficulty. The required number of coordinate systems for an element depends on the element type and applications. And at least, two or more such coordinate systems are needed for the non-simplicial elements. While using more than one coordinate system, the condition number $\kappa(A_e'^{-1}A_e)$ is bounded by the combination of the topology parameters of several simplicial elements that are embedded in a non-simplicial element. This may suggest that a quality metric for a non-simplicial element can be defined by the element quality metrics for the simplicial elements embedded in the

[2] Gambit 2.3.16, Fluent Inc.

non-simplicial element, such as $\max_{i=1,2} \kappa(G_{e,i} G_{e,i}^T)$ for the quadrilateral elements. This is actually resonant to the suggestion by Knupp [31], who suggested that the non-simplicial elements may need multiple Jacobian matrices in definitions of quality metrics.

III.2.3. Graph interpretation of matrix transformations

Inspect the element gradient matrix B'_e in the new coordinate system. It is apparent that its rows are actually scaled edge vectors $u_{\langle ij \rangle}$'s, which has the following form,

$$u_{\langle ij \rangle} = a_{ij} e_{\langle ij \rangle} = a_{ij} \begin{bmatrix} \vdots \\ 1 \\ \vdots \\ -1 \\ \vdots \end{bmatrix}.$$

where $e_{\langle ij \rangle}$ is the unit edge vector and a_{ij} is the edge weight. Notice that all entries of $u_{\langle ij \rangle}$ are zeros except the ith and jth. It is easy to see that $u_{\langle ij \rangle} u_{\langle ij \rangle}^T$ is a Laplacian matrix. So, basically we use $A'_e = \omega_e B'_e B_e'^T$, a Laplacian matrix, to approximate the element stiffness matrix A_e. If an element matrix A_e is viewed as a graph, whose vertices are labeled from 1 to n_e, and whose edges are the nonzero off-diagonal entries, then A_e is a clique in general. Then the constructed matrix A'_e can be seen as a star graph embedded in this clique. The spectral distance between A'_e and A_e is bounded by $\kappa(G_e G_e^T)$, where G_e is the element Jacobian of the new coordinate system, whose axes are along the star graph edges. As shown in previous sections, at the element level, one star graph is used to approximate the graph of simplicial elements, whereas two or more star graphs are needed for other types of Lagrangian elements. Also notice that, this kind of approximation not only transforms the original stiffness matrix A into a diagonally dominant M-matrix, it also sparsifies A.

III.3. Elliptic problems with variable coefficients

The preconditioning approach described earlier can be applied to a wider class of elliptic problems. Now let's consider problem (2.3) with $K(\mathbf{x})$ being SPD and $q(\mathbf{x}) = 0$. Assume $K(\mathbf{x})$ is uniformly bounded positive definite in Ω, i.e. there are positive constants γ, Γ, such that $\gamma |u|^2 \leq u^T K(\mathbf{x}) u \leq \Gamma |u|^2$ for all $u \in \mathbb{R}^d$ and for all $\mathbf{x} \in \Omega$. Appropriate choices of $K(\mathbf{x})$ can be used to model a wide variety of domain properties including inhomogeneity and anisotropy. In the following, our approach is illustrated by using linear triangular elements. A piecewise linear finite element method gives,

$$A = \sum_{e \in \mathcal{T}} B_e^T D_e B_e \Delta_e,$$

where B_e is the element gradient matrix as given before, D_e is obtained by evaluating $K(\mathbf{x})$ at some quadrature point of element e or the average of the integration of $K(\mathbf{x})$. Using the transformation described in Section III.2.1, we obtain an alternative expression for matrix A,

$$A = \sum_{e \in \mathcal{T}} B_e'^T G_e^{-T} D_e G_e^{-1} B_e' \Delta_e,$$

which is used to derive the following approximation of A,

$$A' = \sum_{e \in \mathcal{T}} \omega_e B_e'^T D_e' B_e' \Delta_e,$$

where D_e' is a positive diagonal matrix that is chosen to minimize the approximation error and ω_e is the scaling factor. It is easy to see that A' is a symmetric M-matrix. Take $\omega_e = 1/\lambda_{\min}(D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2})$, the quality of the approximation is determined by the following bound

$$\kappa(A'^{-1}A) \leq \max_{e\in\mathcal{T}} \kappa(D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2}).$$

which should be minimized by appropriate choice of D_e' in each element.

III.3.1. Inhomogeneous domains

Discontinuity in the domain properties is common in a wide variety of applications such as heat transfer through composite materials, multiphase flows of viscous liquids, and groundwater modeling. Other applications where the coefficients have large discontinuities are electrical networks, semiconductors, and electromagnetics modeling. For problems involving different isotropic materials, the discontinuity is restricted to the interface between the materials. One can use a mesh that conforms to these interfaces such that elements do not straddle different materials. In this case, $D_e = d_e I$, where d_e is a scalar and I is the identity matrix. By choosing $D_e' = D_e$, as shown in the following, we guarantee that the quality of the preconditioner is no different from that of an isotropic problem. This holds even for arbitrarily ill-conditioned scenarios.

$$\begin{aligned} \kappa(A'^{-1}A) &\leq \max_{e\in\mathcal{T}} \kappa(D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2}) \\ &= \max_{e\in\mathcal{T}} \kappa(D_e^{1/2} G_e D_e^{-1} G_e^T D_e^{1/2}) \\ &= \max_{e\in\mathcal{T}} \kappa(G_e G_e^T). \end{aligned}$$

III.3.2. Anisotropic domains

In an anisotropic domain, $K(\mathbf{x})$ varies continuously over the domain. The eigenvectors and eigenvalues of $K(\mathbf{x})$ give the orthogonal directions and their corresponding magnitudes of anisotropy, respectively, at the point $\mathbf{x}$. For the coordinate transformation, one should choose the edges closely aligned with these eigenvectors.

At the element level, each block matrix D_e is SPD, therefore, $\exists P_e$, s.t. $D_e = P_e \Lambda_e P_e^T$, where Λ_e is a positive diagonal matrix with the eigenvalues of D_e (the eigenvalues are bounded below by γ and above by Γ) as the diagonal entries, and P_e is an orthonormal matrix, whose columns are the corresponding eigenvectors. Consider the case when every element e has two adjacent edges aligned with the eigenvectors of D_e, whose unit vectors form the new coordinate system. The transformation matrix G_e^T is identical to the eigenvector matrix of P_e, leading to the following simplification,

$$A = \sum_{e \in \mathcal{T}} B_e'^T G_e^{-T} (G_e^T \Lambda_e G_e) G_e^{-1} B_e' \Delta_e = \sum_{e \in \mathcal{T}} B_e'^T \Lambda_e B_e' \Delta_e,$$

which indicates that A is a symmetric diagonally dominant M-matrix. From this result, intuitively, we may speculate that if a mesh is generated according to the anisotropic characteristics of the domain, then our approach would work perfectly well. A justification of this speculation is given in the following.

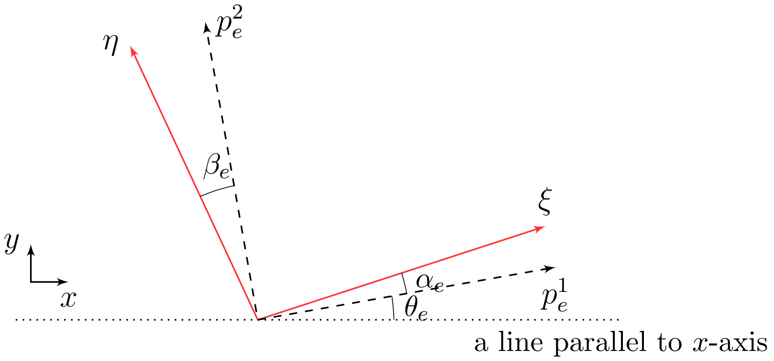


FIG. III.5. *Mesh edges not well aligned with the anisotropic principal axes.*

Analyzing a full fledged 3D case is an extremely involved task and it is easy to get lost due to the number of parameters involved. For simplicity, let's analyze a two dimensional case. Suppose (ξ, η) is our chosen coordinate system in element e. Let

$P_e = [p_e^1, p_e^2]$, where $p_e^i, i = 1, 2$ are the column vectors. P_e can be written as,

$$P_e = \begin{bmatrix} \cos\theta_e & -\sin\theta_e \\ \sin\theta_e & \cos\theta_e \end{bmatrix}, \tag{3.9}$$

where θ_e is the angle between p_e^1 and x-axis. Without loss of generality, let $\Lambda_e = diag(1, \delta_e)$, where $\gamma/\Gamma \leq \delta_e \leq 1$. Suppose the mesh edges don't align with the anisotropic axes perfectly, and α_e, β_e are the angles between ξ and p_e^1, and η and p_e^2 respectively (positive in anti-clockwise direction), as shown in Fig. III.5. Notice that P_e is an orthonormal matrix, some simple calculations show that,

$$D_e^{-1} = P_e \begin{bmatrix} 1 & 0 \\ 0 & 1/\delta_e \end{bmatrix} P_e^T,$$

$$G_e = \begin{bmatrix} \cos(\theta_e + \alpha_e) & \sin\theta_e + \alpha_e) \\ -\sin(\theta_e + \beta_e) & \cos(\theta_e + \beta_e) \end{bmatrix},$$

$$G_e D_e^{-1} G_e^T = \begin{bmatrix} \cos^2\alpha_e + \sin^2\alpha_e/\delta_e & \sin\alpha_e\cos\beta_e/\delta_e - \cos\alpha_e\sin\beta_e \\ \sin\alpha_e\cos\beta_e/\delta_e - \cos\alpha_e\sin\beta_e & \sin^2\beta_e + \cos^2\beta_e/\delta_e \end{bmatrix}.$$

Let $D_e'^{-1} = diag(G_e D_e^{-1} G_e^T)$. After diagonal scaling,

$$D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2} = \begin{bmatrix} 1 & F_e \\ F_e & 1 \end{bmatrix}.$$

where $F_e = F_{n,e}/F_{d,e}$, and

$$F_{n,e} = \sin\alpha_e \cos\beta_e - \delta_e \sin\beta_e \cos\alpha_e,$$

$$F_{d,e} = \sqrt{(\delta_e \cos^2\alpha_e + \sin^2\alpha_e)(\delta_e \sin^2\beta_e + \cos^2\beta_e)}.$$

The eigenvalues of $D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2}$ are $\lambda_e = 1 \pm F_e$. The condition number of $A'^{-1}A$ can be written as,

$$\begin{aligned}\kappa(A'^{-1}A) &= \max_{e\in\mathcal{T}} \kappa(D_e'^{1/2} G_e D_e^{-1} G_e^T D_e'^{1/2}) \\ &= \max_{e\in\mathcal{T}} \frac{1+|F_e|}{1-|F_e|} \qquad (3.10) \\ &= \max_{e\in\mathcal{T}} \frac{F_{d,e}+|F_{n,e}|}{F_{d,e}-|F_{n,e}|}.\end{aligned}$$

Notice that $|F_{n,e}| \le F_{d,e}$, $\kappa(A'^{-1}A)$ also can be estimated as,

$$\begin{aligned}\kappa(A'^{-1}A) &= \max_{e\in\mathcal{T}} \frac{(F_{d,e}+|F_{n,e}|)^2}{F_{d,e}^2 - F_{n,e}^2} \\ &\le \max_{e\in\mathcal{T}} \frac{4F_{d,e}^2}{F_{d,e}^2 - F_{n,e}^2}.\end{aligned} \qquad (3.11)$$

From the analysis of (3.10) and (3.11) we obtain the following conclusions:

- If the domain is isotropic, i.e., $\delta_e = 1$,

$$F_{n,e} = \sin\alpha_e \cos\beta_e - \sin\beta_e \cos\alpha_e = \sin(\alpha_e - \beta_e),$$

$$F_{d,e} = \sqrt{(\cos^2\alpha_e + \sin^2\alpha_e)(\sin^2\beta_e + \cos^2\beta_e)} = 1.$$

thus, (3.10) gives,

$$\kappa(A'^{-1}A) = \max_{e\in\mathcal{T}} \frac{1+|\sin(\alpha_e-\beta_e)|}{1-|\sin(\alpha_e-\beta_e)|}.$$

In fact , this recovers the result of (3.3) in Section III.2.1.

- If mesh edges align perfectly with the dominant anisotropic axis, i.e. $\alpha_e = 0$,

  $$F_{n,e} = -\delta_e \sin\beta_e,$$

  $$F_{d,e} = \sqrt{\delta_e(\delta_e \sin^2\beta_e + \cos^2\beta_e)}.$$

  thus, (3.11) gives,

  $$\begin{aligned}\kappa(A'^{-1}A) &= \max_{e\in\mathcal{T}} \frac{4\delta_e(\delta_e\sin^2\beta_e+\cos^2\beta_e)}{\delta_e\cos^2\beta_e} \\ &\le \max_{e\in\mathcal{T}} \frac{4(\delta_e\sin^2\beta_e+\cos^2\beta_e)}{\cos^2\beta_e}.\end{aligned}$$

  recall $\gamma/\Gamma \le \delta_e \le 1$, thus,

  $$\kappa(A'^{-1}A) \le \frac{4}{\cos^2\beta_e}.$$

  In this case, the condition number of $A'^{-1}A$ is well bounded independent of γ and Γ.

- In general, (3.11) gives,

  $$\kappa(A'^{-1}A) \le \max_{e\in\mathcal{T}} \frac{4(\delta_e\cos^2\alpha_e+\sin^2\alpha_e)(\delta_e\sin^2\beta_e+\cos^2\beta_e)}{\delta_e(\cos\alpha_e\cos\beta_e+\sin\alpha_e\sin\beta_e)^2}. \qquad (3.12)$$

since $\gamma/\Gamma \leq \delta_e \leq 1$,

$$\begin{aligned}\kappa(A'^{-1}A) &\leq \max_{e\in\mathcal{T}} \frac{4(\cos^2\alpha_e + \sin^2\alpha_e)(\sin^2\beta_e + \cos^2\beta_e)}{\delta_e(\cos\alpha_e\cos\beta_e + \sin\alpha_e\sin\beta_e)^2} \\ &\leq \frac{4\Gamma}{\gamma}\max_{e\in\mathcal{T}}\frac{1}{\cos^2(\alpha_e - \beta_e)}.\end{aligned}$$

thus, the condition number of our preconditioning system is small if the ratio of Γ and γ is small.

Actually, if the mesh is adapted to the coefficients, i.e. $\alpha = \max_{e\in\mathcal{T}}\max\{\alpha_e, \beta_e\}$ is adjusted to the anisotropic ratio γ/Γ, our approach is anticipated to work well even when γ/Γ approaches zero. Let $\gamma/\Gamma \to 0$, and $\alpha \to 0$, from (3.12) we can get,

$$\begin{aligned}\kappa(A'^{-1}A) &\leq \max_{e\in\mathcal{T}} \frac{4(\delta_e\cos^2\alpha_e + \sin^2\alpha_e)(\delta_e\sin^2\beta_e + \cos^2\beta_e)}{\delta_e(\cos\alpha_e\cos\beta_e + \sin\alpha_e\sin\beta_e)^2} \\ &\leq \max_{e\in\mathcal{T}} \frac{4(\delta_e\cos^2\alpha_e + \sin^2\alpha_e)}{\delta_e(\cos\alpha_e\cos\beta_e + \sin\alpha_e\sin\beta_e)^2} \\ &= \max_{e\in\mathcal{T}} O\left(\frac{4(1+(\alpha_e)^2/\delta_e)}{(1+\alpha_e\beta_e)^2}\right) \qquad (\alpha_e, \beta_e \to 0) \\ &= O\left(\frac{\alpha^2}{\gamma/\Gamma}\right).\end{aligned}$$

This shows that if $\gamma/\Gamma \to 0$ and $\frac{\alpha^2}{\gamma/\Gamma} \to O(1)$, then $\kappa(A'^{-1}A)$ is well bounded. This means that a small γ/Γ can be compensated by well aligned meshes. In practice, when $\gamma/\Gamma \to 0$, the mesh is usually refined accordingly, such that α approaches zero. Experimental results of the support graph preconditioners for the anisotropic problems in Chapter IV show the effectiveness of our approach.

III.3.3. Notes on quadrilateral and hexahedral elements

There is no difficulty to apply the above approach to the quadrilateral and hexahedral elements when solving problems with variable coefficients. Due to the fact that these elements usually need more quadrature points to obtain the stiffness matrix, it seems that when constructing M-matrix by evaluating $K(\mathbf{x})$ at the centroid of an element e may not be so "accurate". However, as a preconditioner, this is not a problem at all. Another point is, in three dimensional space, it is difficult or impossible to construct a pure tetrahedral mesh such that in each tetrahedron we can find three adjacent edges that are well aligned with the anisotropic axes; this certainly is not a problem for a hexahedral mesh. In this sense, quadrilateral and hexahedral elements are good candidates for solving problems with inherent directions. In real problems, depending on the application a hybrid mesh may serve well.

III.4. Diffusion-reaction problems

Consider the general diffusion-reaction problem of (2.3), with uniformly bounded SPD matrix $K(\mathbf{x})$ and $0 \leq q(x) \leq Q$. The linear system of (2.6) is resulted after standard Galerkin discretization. Here A consists of two parts: $A = A_1 + A_0$, where A_1 is the stiffness matrix, and A_0 is the mass matrix. According to what has been discussed so far, we already have a good M-matrix preconditioner A_1' for A_1,

$$\tilde{c}_0 \leq \frac{u^T A_1 u}{u^T A_1' u} \leq 1,$$

where $\tilde{c}_0$ depend only on mesh properties (such as element angles, edge alignment) and domain properties (such as anisotropy). A well known fact about mass matrix is that, in a certain sense it is a scaled identity matrix (see, e.g. [20]), therefore, its diagonal matrix approximates it very well. Let $A_0' = diag(A_0)$, then,

$$\widehat{c}_0 \leq \frac{u^T A_0 u}{u^T A_0' u} \leq \widehat{c}_1, \quad \forall u \in V_h,$$

where the constant $\widehat{c}_0, \widehat{c}_1$ depends only on the element type, dimension d and the order of elements. For example, $\widehat{c}_0 = 1/2,\ \widehat{c}_1 = 1 + d/2$ for linear simplicial elements (see [47, 34]). Since A_0' is positive, combining these two matrices together gives an M-matrix $A' = A_1' + A_0'$, which is used to approximate A. Since $u^T A u = u^T A_1 u + u^T A_0 u$, we have,

$$\tilde{c}_0 u^T A_1' u + \widehat{c}_0 u^T A_0' u \leq u^T A u \leq u^T A_1' u + \widehat{c}_1 u^T A_0' u, \quad \forall u \in V_h.$$

so,

$$\min\{\tilde{c}_0, \widehat{c}_0\} u^T (A_1' + A_0') u \leq u^T A u \leq \max\{1, \widehat{c}_1\} u^T (A_1' + A_0') u, \quad \forall u \in V_h.$$

i.e.,

$$\min\{\tilde{c}_0, \widehat{c}_0\} \leq \frac{u^T A u}{u^T A' u} \leq \max\{1, \widehat{c}_1\}, \quad \forall u \in V_h.$$

Thus,

$$\kappa(A'^{-1} A) \leq \frac{\max\{1, \widehat{c}_1\}}{\min\{\tilde{c}_0, \widehat{c}_0\}}. \tag{3.13}$$

that is again independent of the size of the problem. Notice that other strategies, such as using a lumped mass matrix to approximate the mass matrix, would achieve a similar result.

III.5. Extension to higher order finite elements

From finite element theory, it is known that if the solution u of (2.4) is smooth, then the errors in the L_2-norm of the solution is of order $O(h^{p+1})$ as $h \to 0$, where h is the maximum diameter of elements and p is the highest order of the finite element basis.

Our idea on solving higher order finite element problem is to use the lower order problem to approximate it. Here we especially borrow some results from two-level methods (see [13, 9]).

The two-level methods are considered in the context of hierarchical bases. With every node, we associate a basis function with a polynomial of degree at most p such that the set of the basis functions is linearly independent. Let these basis functions span the finite dimensional space V_h. V_h can be decomposed as the direct sum $V_h = V_h^{(1)} \bigoplus V_h^{(2)}$, where $V_h^{(1)} \bigcap V_h^{(2)} = 0$. $V_h^{(1)}$ is spanned by the set of basis functions $\{\phi_1, \cdots \phi_{N_v}\}$ associated with the vertex nodes, which is a complete set of linear (or multi-linear) functions in V_h. $V_h^{(2)}$ is spanned by the rest of basis functions $\{\phi_{N_v+1}, \cdots, \phi_N\}$. The final assembled matrix obtained by the finite element discretization thus has a 2×2 block form,

$$A = \begin{bmatrix} A_1 & C \\ C^T & A_2 \end{bmatrix}, \tag{3.14}$$

where

$$\begin{aligned} A_1 &= \{a(\phi_i, \phi_j)\}, \qquad i, j = 1, \cdots, N_v, \\ A_2 &= \{a(\phi_i, \phi_j)\}, \qquad i, j = N_v + 1, \cdots, N, \\ C &= \{a(\phi_i, \phi_j)\}, \qquad i = 1, \cdots, N_v, \ j = N_v + 1, \cdots, N. \end{aligned}$$

Note that A_1, A_2 are SPD and have spectral numbers $\kappa(A_1) = O(h^{-2})$ and $\kappa(A_2) = O(1)$ as $h \to 0$ (see [13, 9]). Also notice that the order of matrix A_1 is very small compared to A. For example, for simplicial elements, the vertex nodes are $v = O(N/p^d)$, where N is the total number of nodes, d is the dimension. For a regular triangulation, we have the following lemma:

Lemma III.1 (see [13, 9]) *The spectral condition number of*

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}^{-1} \begin{bmatrix} A_1 & C \\ C^T & A_2 \end{bmatrix}. \tag{3.15}$$

is bounded from above by $\frac{1+\varsigma}{1-\varsigma}$*, where* $0 < \varsigma < 1$*. The parameter* ς *depends on the geometry properties of the mesh and the boundedness of* $K(\mathbf{x})$ *and* $q(\mathbf{x})$*, and it is independent of the problem size.*

One strategy is to use the diagonal block matrix $[A_1, 0; 0, A_2]$ to approximate the original matrix (3.14). To construct an M-matrix preconditioner for A, an M-matrix preconditioner for A_1 and A_2 is needed. Employing the approach described in previous sections, an M-matrix A_1' can be constructed to approximate A_1. For the higher order part A_2, a trivial diagonal preconditioning is used (other strategies to construct M-matrix approximation for A_2 can be explored). Let $A_2' = diag(A_2)$, then $\kappa(A_2'^{-1}A_2) = O(1)$. Thus, we have,

$$\begin{aligned} &\kappa\left(\begin{bmatrix} A_1' & 0 \\ 0 & A_2' \end{bmatrix}^{-1} \begin{bmatrix} A_1 & C \\ C^T & A_2 \end{bmatrix}\right) \\ \leq\ & \kappa\left(\begin{bmatrix} A_1' & 0 \\ 0 & A_2' \end{bmatrix}^{-1} \begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}\right) \cdot \kappa\left(\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}^{-1} \begin{bmatrix} A_1 & C \\ C^T & A_2 \end{bmatrix}\right) \\ =\ & (\max\{\kappa(A_1'^{-1}A_1), \kappa(A_2'^{-1}A_2\}) \cdot \frac{1+\varsigma}{1-\varsigma} \\ =\ & O(1). \end{aligned}$$

In fact, the idea above can be applied to standard Lagrangian elements without any difficulty. The two spaces $V_h^{(1)}$ and $V_h^{(2)}$ can be constructed by linear transformation on the original basis. Let's consider a typical element e that has q nodes, where the first v $(v \leq q)$ are vertex nodes. Since each polynomial of degree at most p is a unique linear combination of these higher order Lagrangian basis functions, in particular, we can represent v independent linear basis functions at the vertex nodes. Thus, we have,

$$\begin{bmatrix} \boldsymbol{\psi}_1^e \\ \boldsymbol{\psi}_2^e \end{bmatrix} = W^e \begin{bmatrix} \boldsymbol{\phi}_1^e \\ \boldsymbol{\phi}_2^e \end{bmatrix} = \begin{bmatrix} W_{11}^e & W_{12}^e \\ 0 & I \end{bmatrix} \begin{bmatrix} \boldsymbol{\phi}_1^e \\ \boldsymbol{\phi}_2^e \end{bmatrix},$$

where $\boldsymbol{\phi}^e = [\boldsymbol{\phi}_1^e, \boldsymbol{\phi}_2^e]^T = [\phi_1^e, \cdots \phi_v^e, \phi_{v+1}^e, \cdots, \phi_q^e]^T$ are the Lagrangian basis, and $\boldsymbol{\psi}^e = [\boldsymbol{\psi}_1^e, \boldsymbol{\psi}_2^e]^T = [\psi_1^e, \cdots \psi_v^e, \psi_{v+1}^e, \cdots, \psi_q^e]^T$ are the new basis with the first v functions being linear. Then, let $\boldsymbol{\psi}_1^e$ span $V_{h,e}^{(1)}$ and $\boldsymbol{\psi}_2^e$ span $V_{h,e}^{(2)}$. After ordering the vertex nodes first globally, we have,

$$\boldsymbol{\psi} = W\boldsymbol{\phi},$$

where, $\boldsymbol{\psi} = [\psi_1, \cdots, \psi_{N_v}, \psi_{N_v+1}, \cdots, \psi_N]^T$, $\boldsymbol{\phi} = [\phi_1, \cdots, \phi_{N_v}, \phi_{N_v+1}, \cdots, \phi_N]^T$, and W denotes the set of all element transformation matrices $\{W^e\}$ in the global node ordering (the vertex nodes are ordered first). $V_h^{(1)}$ is spanned by basis functions $\{\psi_1, \cdots, \psi_{N_v}\}$, and $V_h^{(2)}$ is spanned by the rest of the basis functions $\{\psi_{N_v+1}, \cdots, \psi_N\}$. The final assembled matrices obtained by these two sets of basis functions have the following relationship:

$$A_\psi = a(\boldsymbol{\psi}, \boldsymbol{\psi}) = a(W\boldsymbol{\phi}, W\boldsymbol{\phi}) = Wa(\boldsymbol{\phi}, \boldsymbol{\phi})W^T = WA_\phi W^T.$$

Thus solving the original linear system $A_\phi x = b$ is equivalent to solving the new linear system $A_\psi x^\psi = b^\psi$, where $x^\psi = W^{-T}x,\ b^\psi = Wb$ (Note that the inverse matrix W is not needed in the actual implementation, since x is obtained from x^ψ). The matrix A_ψ has the 2×2 block structure of (3.14).

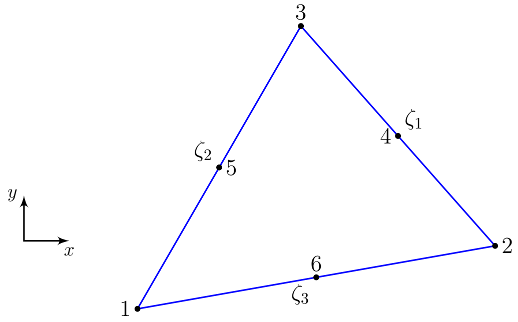


Fig. III.6. *P2 master element.*

Now we give an example of solving standard triangular quadratic elements by this approach. The standard basis functions of the master quadratic triangular element (see Fig. III.6) are:

$$\begin{aligned}
&\phi_1^e = \zeta_1(2\zeta_1 - 1), \quad &&\phi_2^e = \zeta_2(2\zeta_2 - 1), \quad &&\phi_3^e = \zeta_3(2\zeta_3 - 1),\\
&\phi_4^e = 4\zeta_2\zeta_3, &&\phi_5^e = 4\zeta_1\zeta_3, &&\phi_6^e = 4\zeta_1\zeta_2.
\end{aligned}$$

where $\{\zeta_i\},\ i = 1, 2, 3$ are the natural coordinates. The linear transformation used is,

$$W^e = \begin{bmatrix} 1 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 1 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

By $\boldsymbol{\psi^e} = W^e\ \boldsymbol{\phi^e}$, a new set of basis functions $\{\psi_i^e\},\ i = 1, \cdots, 6$ is obtained, where the first three are actually the shape functions of the corresponding linear element, and the last three are unchanged. This new set functions is, in fact, the standard hierarchical basis for the quadratic element:

$$\begin{aligned} &\psi_1^e = \zeta_1, && \psi_2^e = \zeta_2, && \psi_3^e = \zeta_3, \\ &\psi_4^e = 4\zeta_2\zeta_3, && \psi_5^e = 4\zeta_1\zeta_3, && \psi_6^e = 4\zeta_1\zeta_2. \end{aligned}$$

In this new basis $\{\psi_i\}$s (where the vertex nodes are ordered first), we take the preconditioner A' as,

$$A' = \begin{bmatrix} A_1' & 0 \\ 0 & diag(A_2) \end{bmatrix}.$$

Notice that A_1' is the transformed matrix for the stiffness matrix A_1 of the corresponding linear elements in the new basis.

Table III.5

The number of iterations of PCG using preconditioner A' for quadratic elements.

# elements	215	822	3,344	13,256	53,036	212,093
# unknowns	400	1,583	6,565	26,269	105,579	423,204
size of A_1	93	381	1,611	6,507	26,272	105,556
# iterations	20	22	21	23	22	21

We applied the approach described above to the unit square Poisson model problem with Dirichlet boundary condition. The quadratic triangular mesh is generated by *Triangle*. The PCG stopping criteria is the same as before. Experimental results are reported in Table III.5, which shows that the number of iterations is almost unchanged with varying problem sizes. Notice that the size of A_1 is only about a quarter of the number of unknowns, which means the M-matrix preconditioner A' is extremely sparse. In Chapter IV, we show that our support graph preconditioner built on top of this M-matrix approximation outperforms the incomplete Cholesky factorization even with less number of nonzeros in the factor.

CHAPTER IV

MODIFIED DOMAIN PARTITIONED SUPPORT GRAPH PRECONDITIONERS

This chapter presents applications of the support graph preconditioning technique to the second order elliptic finite element problems. First, the strategy of applying support graph preconditioners to the finite element problems is given. Then, the current status of support graph preconditioning is reviewed. This is followed by the presentation of a new variant of the support graph preconditioning technique called modified domain partitioned support graph preconditioners.

IV.1. Support graph preconditioning for FEM problems

Our strategy of applying support graph preconditioning technique to a linear system (2.6) derived from finite element problems is a two-step preconditioning: first, generate a symmetric diagonally dominant M-matrix A' for A using the techniques outlined in Chapter III; then, a support graph preconditioner M constructed via A' is used to the original matrix A. According to the methodology of Chapter III, the condition number $\kappa(A'^{-1}A)$ does not depend on the size of the problem and it can be denoted as a constant C_M. Now let c be the congestion and δ the dilation of the support graph M after embedding the graph of A'. By Lemma II.4, the condition number of the preconditioned linear system $M^{-1}Ax = M^{-1}b$ can be estimated as,

$$\kappa(M^{-1}A) \;=\; \kappa(M^{-1}A'A'^{-1}A) \leq \kappa(M^{-1}A') \cdot \kappa(A'^{-1}A) \leq C_M \cdot \delta \cdot c. \qquad (4.1)$$

The condition number is controlled by the product of the support graph's congestion and dilation, which in turn affects the number of iterations of the PCG algorithm. The smaller the condition number, the larger the number of edges in the support graph is required. A dense support graph not only increases the complexity of factorization, it also increases the work per iteration of the PCG algorithm. Depending on the application, support graph preconditioner should be constructed by balancing the overhead of setting up the preconditioner and the work required by the iterations, and the later also can be improved by balancing the number of iterations and the work required per iteration.

IV.2. The state-of-the-art in support graph preconditioning

The first support graph preconditioners were proposed by Vaidya [42]. Vaidya designed a family of preconditioners based on spanning trees in graphs. His first class of preconditioners are maximum-weight spanning tree preconditioners. They are easy to factor without generating any *fill-in* . However, they don't perform well, as for some large sized problems the preconditioned conjugate gradient method even converges slower than the one without preconditioning. His second class of preconditioners are the maximum-weight spanning tree augmented by adding extra edges. Specifically, the algorithm splits the maximum-weight spanning tree into t connected components of almost the same size, then adds the heaviest edge between every pair of subtrees if there is an edge connecting them in the original graph. The condition number of the preconditioned system is improved, resulting in a better convergence rate. However, this is accompanied with an increase in the cost of factorization and the work required in each iteration step. To reduce *fill-in* in factors, a high quality of sparse direct solver is required. The theoretically optimal value of t is around $N^{0.25}$ which gives a total solution time of $O(N^{1.75})$, and $O(N^{1.2})$ for a planar graph. Recently Spielman and Teng [40] gives a near linear time algorithm for solving linear systems by support graph preconditioners, however, their algorithm is quite complex and has not been implemented yet. Implementation and performance analysis of Vaidya's preconditioners was reported by Chen and Toledo [18]. The matrices of their experiments are obtained from finite difference discretization of the second order elliptic PDEs. They showed that the cost and convergence of these preconditioners depends almost only on the nonzero structure of the matrix, but not on its numerical values. Also, these preconditioners are almost oblivious to the boundary conditions and robust for difficult problems. In particular, these preconditioners outperform incomplete Cholesky factorization based preconditioners for anisotropic problems in two dimensional space. For three dimensional problems, in general, Vaidya's preconditioners are much slower than the incomplete Cholesky preconditioners.

Extension to symmetric diagonally dominant positive definite matrices was also mentioned by Vaidya and implemented and analyzed later on by Boman et al.[15]. Now the concept of a maximum-weight spanning tree is extended to a construction called a *maximum-weight basis*. Every symmetric diagonally dominant matrix A can be written in the form of $A = UU^T$, where each column of U is either a positive edge vector, a negative edge vector or a vertex vector. The edges of U can be used to define a structure called a *matroid* – an independent subset of these edges. The goal is to seek a preconditioner $M = VV^T$ for A, where the columns of V are a subset of the columns of U, such that $\kappa(M^{-1}A)$ is small and at the same time M is easy to factor. Constructing a maximum-weight basis is a process of constructing a matroid with maximum weight, in a very similar way (though a little bit complicated) of constructing maximum-weight spanning tree which, in fact, is its special case.

Another important support graph preconditioners, called support trees, were proposed by Gremban et al. [25, 24] They view the matrix-vector multiplication as a process of information 'mixing'. Thus, a matrix is a communication network. The support tree M is required to approximate the topological properties of A, and it is *not* necessarily a subgraph of the original graph A. A support tree is constructed by partitioning the nodes of A hierarchically. At the top level, a root node of M is formed, which represents the original graph. Subsequently, at each level of the hierarchy, a child node is added (to M) for each subgraph of the parent node. The edge between a child and its parent is assigned by a proper weight, such that the communication of the subgraph to the outside world is preserved in average. The final level of subgraphs are the original nodes of A. Thus, each node of M defines a subgraph of A, and each edge of M connects subgraphs at two consecutive hierarchical levels; the root corresponds to the entire graph A, while leaves are the nodes of A. A balanced tree resulted if the subgraphs are roughly the same size at each level of partitioning. For regular grids, a node of M usually has a branching factor of 2^d in d dimensional space. Thus, M is a binary tree for a one dimensional mesh, a quadtree for a two dimensional mesh and a octree for a three dimensional mesh. By assigning appropriate weights for edges of M, M approximates the communication properties of A. Since M is a balanced tree, these preconditioners parallelize well. One of the drawbacks of this approach, however, is the increased system size. They basically solve an expanded system. For example, the number of nodes of M can be twice the number of nodes of A for a two dimensional problem. Another major drawback is, the condition number bounds are available only for matrices whose edge weights are more or less uniform, which is almost equivalent to say, that the support tree approach applies only to the Poisson problems with constant conductivity. In their experiments, they used the matrices derived from regular grid discretization with constant coefficients. One thing worth to mention is that Gremban et al. also presented a technique to transform a problem of solving a linear system with symmetric diagonally dominant matrix to a linear system with symmetric diagonally dominant M-matrix. Though the new M-matrix *doubles* the size both of vertices and edges of the original matrix.

In the context of solving finite element problems, Wang and Sarin [46] proposed domain partitioned support graph preconditioners (DPSG). DPSG is a variant of Vaidya's preconditioners, since it uses subgraph to construct the support graph. Compared to the convectional Vaidya's preconditioners [18], DPSG preconditioners eliminate the recursive procedure of splitting a tree; instead they use a graph partitioning algorithm to partition the domain first, then generate the maximum weighted spanning tree in each domain. Thus the size of the subgraph components is well balanced. In addition, the domain partition approach has inherent parallelism, which can be explored to construct a parallel support graph preconditioner. DPSG sees the graph of the matrix A the same as the finite element mesh, and the graph of transformed M-matrix A' is a subset of the mesh. DPSG forms the subgraph components by partitioning the finite element mesh. To construct the support graph M, first the mesh is divided into t subdomains of nearly equal size, such that each element belongs to one subdomain; then, all the interface edges, i.e., the edges neighboring different subdomains, are included in the support graph M, and are virtually identified as the *super root* of the support graph M; finally, a maximum weight spanning tree is generated in each subdomain starting from the super root, and the edges of the trees are included in the support graph M. The performance study of DPSG for two dimensional finite element problems showed a good potential. A drawback of DPSG is that the support graph construction is involved with the finite element mesh. In this work, we propose a modified partitioned support graph preconditioners (MDPSG) that does not involve the finite element mesh and can be served as a general black box algorithm. In chapter V, we apply MDPSG to various finite element problems and compare its performance with incomplete factorization based preconditioners.

IV.3. MDPSG

MDPSG preconditioners apply to matrices from both finite difference and finite element methods. Since the focus of this work is the finite element problems, MDPSG preconditioners are introduced and tested in the context of the finite element problems.

IV.3.1. MDPSG algorithm

Experimental results showed the effectiveness of our earlier scheme DPSG [46] for solving two dimensional elliptic problems discretized by triangular elements. The domain partition approach of DPSG has inherent parallelism, which is performance friendly on the multi-core or multiple processors. However, the requirement of DPSG to include all interface edges between subdomains in the support graph may have a backward side-effect: it may force selection of these interface edges during element coordinate transformation when constructing a symmetric diagonally dominant M-matrix A' (see Chapter III for detail), such that the final support graph is a well-compartmented connected graph. A care is also needed when dealing with anisotropic problems. Since all the interface edges are included in the support graph, the domain should be partitioned in a way that minimizes the number of edges with small weights.

In this dissertation, we propose a new scheme called MDPSG. The main departure of MDPSG from DPSG is that it partitions the graph of the already transformed M-matrix A' directly. Thus, MDPSG applies to any symmetric diagonally dominant M-matrix, either from finite difference methods or finite element methods. The procedure of constructing the MDPSG support graph M follows the same steps of DPSG but with some subtle differences:

1. Divide the graph A' into t subdomains of nearly equal size. Each node belongs

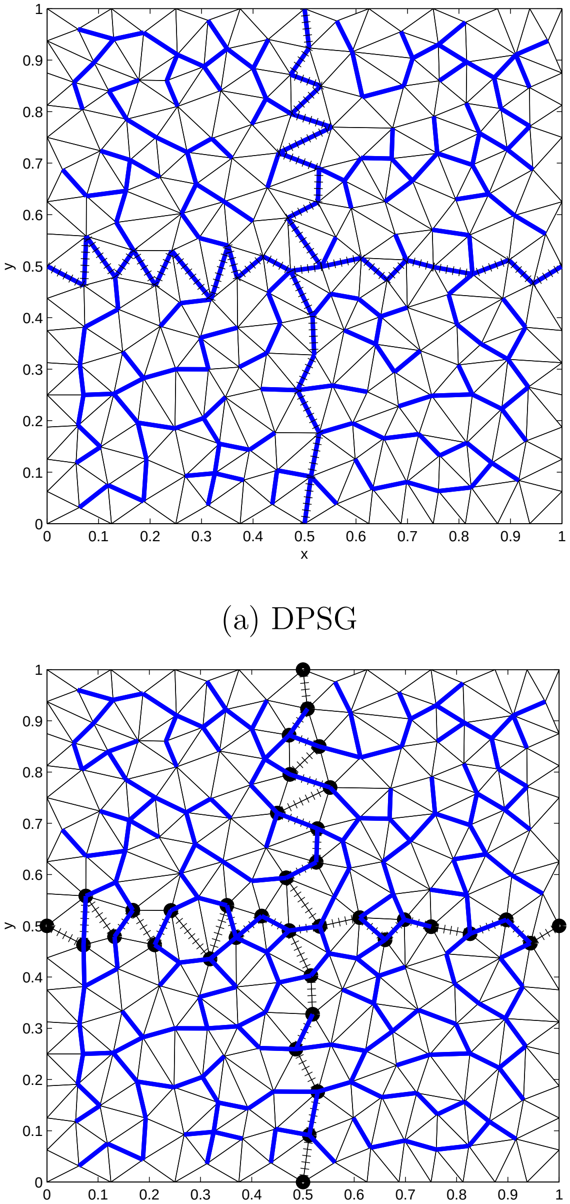
to one subdomain.

2. Each subdomain is augmented by neighboring nodes that are connected to the nodes of the subdomain, i.e. the interface nodes participate in more than one augmented subdomains.

3. A maximum weight spanning tree is generated within each augmented subdomain. The edges of the tree are included in the support graph M.

For finite element problems, constructing MDPSG is entirely decoupled from the process of constructing M-matrix preconditioner A', thus the quality of the preconditioner A' is not compromised in any way. This decoupling makes it a perfect general black box algorithm for all diagonally dominant M-matrices. Another advantage of MDPSG over DPSG is that, it is a true maximum weight spanning graph. Since for any edge $e \in A'$, there is a support path $p \in M$, such that all edges in p have weight at least as heavy as that of e. Thus, no special care is needed when dealing with anisotropic problems.

These two variations of domain partitioned support graph preconditioners are illustrated in Fig. IV.1. The domain is a Poisson problem defined on a two dimensional unit square with Dirichlet boundary condition and discretized by a triangular mesh (generated by *Triangle*). The triangular mesh is geometrically partitioned into 2-by-2 subdomains along x and y directions. The interfaces of these subdomains are distinguished by the barbed edges. The blue edges are the edges of the support graphs: Fig. IV.1(a) is the graph of a DPSG preconditioner and Fig. IV.1(b) is the graph of a MDPSG preconditioner (the nodes at interfaces are noted by thick dot markers).



(a) DPSG

(b) MDPSG

Fig. IV.1. *Graphs of domain partitioned support graph preconditioners.*

IV.3.2. Analysis of MDPSG

We use M_t to denote a MDPSG preconditioner that forms t subgraph components of an n-by-n symmetric diagonally dominant M-matrix A. M_t partition the graph of A into t components $G_1, \cdots, G_t$, each has $O(n/t)$ nodes and $O(n/t)$ edges (A is sparse, the number of edges is in the order of $O(n)$). For any edge $e \in A$, we can find an augmented component $\tilde{G}_i, i \in [1, \cdots, t]$ that contains both of the end nodes of e. The induced subgraph of M_t by the nodes of $\tilde{G}_i$ includes a maximum weight spanning tree of $\tilde{G}_i$, and a support path can be found to support e such that all edges along this path are at least as heavy as e. The support paths in $\tilde{G}_i$ have length at most $O(n/t)$, and each edge in $\tilde{G}_i$ supports at most $O(n/t)$ paths, i.e. both of the congestion and dilation of M_t are in the order of $O(n/t)$. By Lemma II.4, the condition number $\kappa(M_t^{-1}A)$ is bounded by $O(n^2/t^2)$.

As a concrete example, let's consider the two dimensional unit square Poisson problem aforementioned (with constant coefficients $K(\mathbf{x}) \equiv 1$). The mesh is geometrically partitioned along x and y directions to form subdomains. As the mesh refined, for instance, the number of unknowns is quadrupled, the number of subdomains is doubled along each direction to maintain the quality of the preconditioners. The left column of Table IV.1 shows that the number of PCG iterations (asymptotically) remains unchanged when the number of unknowns n quadrupled, as long as the number of subdomains t is quadrupled as well. This follows the condition number bound $O(n^2/t^2)$ obtained above. For such a regular partition, each subdomain is well-shaped, and the diameter of the subdomains is pretty uniform. Notice however, a high quality geometric partition is possible only for a simple domain, such as, a unit square. As a general support graph preconditioner, MDPSG partitions the graph A directly and does not rely on the mesh information. Shown in the right column of

Table IV.1 is the result of MDPSG using *Metis* [29] to partition the graph A. We can see, that the quality of MDPSG is not affected by using a general graph partition algorithm at all, it may be even better due to the high quality of the latter.

TABLE IV.1

Number of iterations of PCG by preconditioner MDPSG with almost fixed n/t. This is a 2D unit square Poisson problem with Dirichlet boundary condition. PCG stops when the relative residual norm below 10^{-6}.

n	Regular geometric partitions		Metis partitions	
	$t = k \times k$	# iter	t	# iter
3,228	8×8	53	64	45
13,099	16×16	57	256	49
52,625	32×32	66	1,024	55
211,362	64×64	76	4,096	63
846,728	128×128	76	16,384	67

In general, the cost of *Metis* partition is $O(n \log n)$ (for a moderate number of partitions, the cost is $O(n)$). The cost of finding maximum weight spanning trees is $O(n \log n)$. Thus, the construction of M_t is not significant. For the factorization, the node elimination can proceed as follows. The root of the maximum weight spanning tree in each subdomain will be eliminated last. The order of elimination for other nodes is according to the node degree. The leaf nodes are eliminated first, then the nodes with degree of 2, and so forth. For the bounded graph, as what is obtained from the finite element problems, the number of *fill-in* is proportional to the number of nodes when the graph is shrinked as a connected graph with t nodes. The elimination of this shrinked graph can take advantage of the graph property, such as planarity. In summary, the factorization cost of M_t follows the Vaidya's augmented spanning tree preconditioners, $O(n + t^6)$ work and $O(n + t^4)$ space required for a general graph A, and $O(n + t^{1.5})$ work and $O(n + t \log t)$ space required when A is a planar (see [14] for detail). The parameter t should be chosen to balance the factorization cost and the PCG iteration cost. A large number t results in a smaller condition number and hence the high convergence rate, at the expense of large *fill-in* in M_t's factors. For the extreme cases, M_1 gives a maximum spanning tree preconditioner, the Cholesky factorization can proceed without any *fill-in*, while PCG converges the most slowly; M_n is the original matrix itself, the Cholesky factorization is the most expensive one, while PCG converges in only one step (a direct solver in fact).

For convenience, an alternative notation MDPSG-s (where s is an integer) is used in Chapter V. MDPSG-s refers to MDPSG that has subdomains of size s, that is, (approximately) s number of nodes are allocated for each subdomain. For instance, MDPSG-1 corresponds to M_n, while MDPSG-n corresponds to M_1.

CHAPTER V

EXPERIMENTAL RESULTS

This chapter presents the experimental results of applying the MDPSG preconditioners to various second order elliptic finite element problems. The methodology of performance comparison and experimental setup are introduced first. The effectiveness of the MDPSG preconditioners compared to the incomplete factorization based preconditioners is presented next. Finally, the parallel potential of these preconditioners is discussed.

V.1. Methodology

The performance of MDPSG preconditioners is compared to that of incomplete factorization based preconditioners. Both incomplete factorization based preconditioning and support graph preconditioning share the same idea of reducing the *fill-in* in the factors. Incomplete factorization based preconditioners sparsify the factors during the factorization, that is – the factorization is *incomplete*; whereas, support graph preconditioners sparsify the original matrix first and then factorize *completely.* Since the incomplete factorization based preconditioning is widely recognized as an effective method for iterative sparse linear solvers, and it has a striking parity with support graph preconditioning, our MDPSG experiments are conducted in parallel with incomplete factorization based preconditioners.

The incomplete factorization based preconditioners are also called ILU (Incomplete LU factorization), which are constructed in the factored form $M = \tilde{L}\tilde{U}$, with $\tilde{L}$ and $\tilde{U}$ being lower and upper triangular matrices. For symmetric case, these preconditioners are called ICC (Incomplete Cholesky), with the factored form $M = \tilde{L}\tilde{L}^T$.

ILU falls into two categories: threshold based ILUT and structure based ILU(k). In ILUT (see [36]), a *fill-in* is permitted only if it is larger than a specified value, and an upper limit may be placed on the number of entries permitted in a row. Thus, the locations of *fill-in* of ILUT are determined during numeric factorization dynamically. In contrast, ILU(k) has two phases. In the symbolic factorization phase, the locations of *fill-in* are determined. Each *fill-in* is assigned a level, a *fill-in* is permitted only if its level is not greater than k. Usually the level of *fill-in* is defined recursively from the level of its parents in the Gaussian elimination process. A common scheme is that all nonzero entries of the original matrix are assigned 0 level, and zero entries are assigned ∞ level. The level of a *fill-in* is the minimum value of the current level and the sum of the levels of its two parent entries incremented by one (see, e.g., [37] for detail). In the numerical factorization phase the factors are constructed according to the symbolic factorization. Since the parameter for ILUT should be tuned to a particular application, we only consider ILU(k) here, in particular the ICC(k).

One of the difficulties of comparing different algorithms is the implementation problem. Different implementations can give totally different results. Both for efficiency and objective comparison, whenever possible we use third party software that have good reputation and popularity. Our driver program is coded using library of Portable Extensible Toolkit for Scientific Computation (*PETSc*) [10, 11, 12], which nowadays is a very popular computation suit for the scalable (parallel) solution of scientific applications modeled by partial differential equations. *PETSc* has implemented data structures and routines that provide the building blocks for solving large scale applications both on parallel and serial computers. Many Krylov subspace methods and preconditioners are available in *PETSc* with superior performance compared to most implementations. In addition, *PETSc* provides a uniform access to external packages composed of popular direct solvers and preconditioners. In particular, we use *PETSc*’ external package *MUMPS* [4, 5, 6] for factorization of MDPSG and *BlockSolve95* [28] as the parallel implementation of ICC(0). *MUMPS* is the public domain package called MUltifrontal Massively Parallel sparse direct Solver. It is a multifrontal code primarily intended for unsymmetric and symmetric positive definite systems, it can also be used to solve many indefinite problems. *MUMPS* has interface to several pivot orderings, such as the approximate minimum degree (AMD) ordering [2], an approximate minimum degree fill-in (AMF) ordering [3], and the nested dissection (ND) ordering provided by *Metis* [30] etc. *BlockSolve95* was developed in the 90’s, and it is a general purpose, scalable parallel software library for the solution of sparse linear system. *BlockSolve95* implements ILU(0) and requires the matrix being symmetric in structure. It uses a parallel coloring algorithm that allows for the efficient computation of matrix ordering and scalable performance.

Table V.1

Code components of PCG solvers with different preconditioners.

components	PCG solver		
	MDPSG (serial, parallel)	ICC(k) (serial)	Parallel ICC(0) (parallel)
assemble	our C++ code		
M-matrix transformation	our C++ code	-	-
partition	Metis 4.0	-	-
support graph	our C++ code	-	-
factorization	MUMPS 4.7.3 (invoked through PETSc)	PETSc ICC	BlockSolve95 (invoked through PETSc)
iterative solver	PETSc CG		

- means not applicable.

Our experiments involve three preconditioners in the context of iterative solve PCG: the support graph preconditioner MDPSG, the incomplete Cholesky factorization ICC(k) and the parallel ICC(0). The finite element assembling, M-matrix transformation, and support graph formation are done by our own C++ code; all other parts use third party software. The code components of these solvers are summarized in Table V.1. Notice that we use ICC(k) that comes with *PETSc*.

V.2. Setup

We conduct experiments on two different platforms. One is an Intel(R) Pentium 4 2.4GHz single processor machine with 2GB RAM, that runs Ubuntu 6.06 Linux operating system, and the compiler is GCC 4.03. All our two dimensional experiments are conducted on the Pentium 4 machine, while three dimensional experiments and parallel experiments are conducted on Hydra – a 640-processor IBM 1.9GHz Power5+'s Cluster system, that runs AIX 5.3 operating system, the compiler is IBM XL C/C++ 8.0. Each node is a symmetric multi-processor (SMP) system, that is, with 16 Power5+ processors and a 32GB of shared memory (only about 25GB are available to user processing). All parallel programs employ the message-passing interface (MPI) library.

Table V.2 lists descriptions of some notations that will be used in the report tables and/or figures. Also notice that the preconditioner name is often used to refer the PCG solver with that preconditioner if the context is clear. In all experiments, the PCG iterations start with initial zero solution. Considering the large size of the linear system and the ill-conditioned problems to be solved, all PCG solvers use the following stopping criteria: the relative residual norm is below 10^{-10}, i.e. $\|r\|/\|r_0\| < 10^{-10}$, where $r = b - Ax$ is the residual and $\| \cdot \|$ is the two-norm. In order to evaluate the

TABLE V.2
Notations used in the tables.

notation	description
n	number of unknowns
p	number of processors
#iter	number of PCG iterations
nnzFac	number of nonzeros of Cholesky or Incomplete Cholesky factor
$\|e\|/\|e_0\|$	the relative solution error in two-norm
M-trx	time (in second) of construct M-matrix during assembling
part	time (in second) of subdomain partition of MDPSG
sg	time (in second) of forming support graphs of MDPSG
fac	time (in second) of Cholesky or Incomplete Cholesky factorization
iter	time (in second) of PCG iterations
time	total time (in second) of a PCG solver
δ	the smallest anisotropy value (the largest anisotropy value is fixed at 1)
rcm	the reverse Cuthill-McKee (RCM) ordering of the nodes of a graph
nd	the nested dissection (ND) ordering of the nodes of a graph

accuracy of a PCG solver, for linear system (2.6), a random solution x is generated and the right hand side is obtained by $b = Ax$; and the solution accuracy is evaluated against this random solution.

Notice that the production term $q(x)$ in (2.3) does not pose any difficulty for the element level M-matrix construction, and it only increases the diagonal dominance of the matrices, which is usually a good feature for any iterative solve. Therefore, $q(x) = 0$ is assumed in all the experiments.

V.3. Two dimensional problems

All two dimensional (2D) serial experiments are conducted on the Pentiums 4 Linux machine. The representative problems considered are: a simple Poisson problem, an inhomogeneous problem and an anisotropic problem.

V.3.1. Poisson problem

The first problem considered is a simple smooth Poisson problem discretized by linear triangular elements (generated by *Triangle*) on a 2D unit square domain $[0, 1] \times [0, 1]$. The coefficients $K(x, y)$ of (2.3) is a scalar function $k(x, y) = 10^{-6}+x^2+y^2$. Two types of boundary condition are considered: one is with the Dirichlet boundary condition at four walls; the other is a mixed boundary condition, where the Neumann boundary condition is at the upper wall and the Dirichlet boundary condition is at all other three walls. The performance of different PCG solvers for these two cases are shown in Table V.3 and Table V.4. Note that the factorization of MDPSG by *MUMPS* uses AMF ordering by default for all our experiments (several orderings of *MUMPS* had been tried, it seems that AMF consistently gives small number of of *fill-in* in the factors for the 2D problems; for 3D problems, auto selection of the ordering sometimes gives less *fill-in* but not consistent). These results show that the performance of ICC is more hindered by Neumann boundary condition, whereas MDPSG is less sensitive. ICC(1) is faster than ICC(0), but ICC(0) is more robust with respect to the node ordering (for the case when ICC(1) with ND ordering fails, *PETSc* outputs convergence reason -8 indicating the indefiniteness of the preconditioner). The ICC with RCM ordering has better performance over the one with ND ordering. MDPSG is robust and it outperforms ICC as the problem size becomes large. When the problem size become too large, where is also the range that a direct solver cannot solve, MDPSG can gracefully increase the parameter s and solve it. Also notice that the performance of MDPSG is not sensitive to the value of s.

Table V.5 shows the performance characteristics of ICC(0) and MDPSG: ICC(0) is more efficient per iteration due to the smaller number of nonzeros of its factor,

Table V.3

Total time (in second) for a 2D unit square Poisson problem with Dirichlet boundary condition.

n	ICC(0)		ICC(1)		MDPSG			
	nd	rcm	nd	rcm	20	15	10	5
26272	1.14	0.71	1.10	0.52	4.00	3.75	3.06	2.59
52625	3.21	1.99	3.61	1.37	8.45	7.96	6.34	5.58
105556	9.75	5.85	13.34	4.06	18.02	16.53	14.18	11.90
211362	29.98	17.59	35.46	12.34	38.77	35.48	30.55	25.84
423329	93.10	48.60	X	36.04	85.41	77.06	64.85	56.91
846728	289.76	161.79	X	109.43	179.65	162.53	145.33	128.25
1693377	916.35	516.23	X	352.51	386.15	352.46	309.70	266.27
3387812	2612.28	1458.74	X	1003.52	819.82	767.35	708.39	-

X means divergence due to the indefiniteness of the preconditioner, - means too large to be solved.

however, the number of iterations of ICC(0) does not scale well with the problem size; MDPSG has more cost per iteration due to the large number of nonzeros of its factor, but the number of iterations almost unchanged when the problem size increases. Under the same PCG stopping criteria, in general, the solution accuracy ($\|e\|/\|e_0\|$, where $e = \tilde{x} - x$, $\tilde{x}$ is the solution of a PCG solver, x is the true solution) of MDPSG is higher than that of ICC(0), especially for the large size problems. A typical performance breakdown of ICC(0) and MDPSG is shown in Table V.6. With much smaller preconditioner overhead, ICC(0) has to take a longer iteration time compared to MDPSG. The only overhead of ICC(0) is the incomplete factorization, which takes a very small portion of the overall time. The overhead of MDPSG has four parts: M-matrix transformation, subdomain partitioning, support graph formation and factorization. The overhead of MDPSG takes a quite portion of the overall time, especially the factorization as the problem size grows. The balance between

Table V.4

Total time (in second) for a 2D unit square Poisson problem with mixed boundary condition.

n	ICC(0)		ICC(1)		MDPSG			
	nd	rcm	nd	rcm	20	15	10	5
26391	1.30	0.81	1.28	0.59	4.14	3.77	3.11	2.64
52799	3.74	2.36	3.85	1.67	8.59	7.67	6.80	5.61
105797	11.73	7.17	X	5.09	18.65	16.75	14.20	12.33
211714	35.58	21.13	43.73	14.48	38.69	36.30	30.18	24.81
423819	112.45	63.90	X	43.46	82.52	77.78	67.00	56.07
847432	347.85	195.15	X	132.20	178.64	161.48	143.37	123.69
1694345	1128.25	627.12	X	426.69	379.41	358.68	301.62	270.64
3389156	3409.64	1942.62	X	1303.16	852.07	785.82	713.15	-

X means divergence due to the indefiniteness of the preconditioner, - means too large to be solved.

factorization and iteration is the vantage point of support graph preconditioning technique that chooses the best of the iterative and direct solve. In practice, the balance of the overhead and the iteration time of MDPSG is application dependent, and is affected by factors such as hardware constrains, implementations of factorization and iterative solve. As a rule of thumb, 10 $\sim$ 50 is a good range of s for two dimensional problems.

Another observation from Table V.5 and Table V.6 is that the iteration implementation of ICC(0) is much efficient than that of MDPSG, considering the number of nonzeros of the factor and the number of iterations (a more clear example on this issue is given in Section V.3.2). Under the same CG implementation of *PETSc*, the poor iteration performance of MDPSG is due to the inefficiency of MUMPS's solution solve phase, which somehow is not very good to be used within an iterative process.

Table V.5

Performance characteristics of ICC(0) and MDPSG for a 2D unit square Poisson problem with mixed boundary condition.

n	ICC(0) rcm				MDPSG 10			
	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$
26391	1.05×10^{5}	174	0.81	7.26×10^{-9}	3.88×10^{5}	37	3.11	6.26×10^{-11}
52799	2.10×10^{5}	245	2.36	1.25×10^{-8}	8.58×10^{5}	41	6.80	7.46×10^{-11}
105797	4.22×10^{5}	340	7.17	2.18×10^{-8}	1.94×10^{6}	41	14.20	1.35×10^{-10}
211714	8.45×10^{5}	469	21.13	4.71×10^{-8}	4.16×10^{6}	43	30.18	9.79×10^{-11}
423819	1.69×10^{6}	648	63.90	7.66×10^{-8}	9.09×10^{6}	46	67.00	7.79×10^{-11}
847432	3.39×10^{6}	901	195.15	9.87×10^{-8}	1.99×10^{7}	48	143.37	8.76×10^{-11}
1694345	6.77×10^{6}	1263	627.12	1.35×10^{-7}	4.37×10^{7}	47	301.62	1.19×10^{-10}
3389156	1.35×10^{7}	1763	1942.62	2.00×10^{-7}	9.43×10^{7}	49	713.15	9.33×10^{-11}

Table V.6

Typical Performance breakdown of ICC(0) and MDPSG for a 2D unit square Poisson problem with mixed boundary condition.

n	ICC(0) rcm		MDPSG 10				
	fac	iter	M-trx	part	sg	fac	iter
26391	0.05	0.76	0.08	0.42	0.08	0.41	2.12
52799	0.11	2.25	0.17	0.86	0.16	0.86	4.75
105797	0.24	6.93	0.32	1.79	0.32	1.92	9.85
211714	0.53	20.60	0.53	3.68	0.64	4.16	21.17
423819	1.19	62.71	1.03	7.46	1.35	9.43	47.73
847432	2.64	192.52	2.25	16.80	2.73	21.68	99.91
1694345	5.97	621.15	4.50	38.50	5.57	53.48	199.57
3389156	13.01	1929.61	11.82	70.70	11.38	137.63	481.62

Our own PCG implementation with MUMPS shows a similar performance.

V.3.2. Inhomogeneous problem

For inhomogeneous problems, i.e. the coefficient $K(\mathbf{x})$ has a jump across different materials, our strategy is to make sure that the edges of finite element mesh do not straddle different materials, then in principle there is no performance degradation of support graph preconditioners after M-matrix transformation (see Section III.3 for detail). Under this strategy, experiments show that the performance of both ICC and MDPSG of an inhomogeneous problem is comparable to that of its homogeneous counterpart.

In the following, we demonstrate that MDPSG with a factor of almost the same or even less number of nonzeros compared to that of ICC(0) still can outperforms the latter. The problem considered is a unit square Poisson problem, $K(x, y)$ is a scalar function $k(x, y)$ that has a jump across the domain:

$$k(x,y) = \begin{cases} \mu, & \text{for } 0 < x \leq 0.5; \\ 1.0, & \text{for } x > 0.5. \end{cases}$$

without loss of generality, assume $0 < \mu \leq 1.0$. The upper wall is specified a Neumann boundary condition, all other three walls are Dirichlet boundary condition. The domain is discretized by quadratic triangular elements using *Triangle*. For higher order elements, our support graph preconditioning strategy is to use lower order elements to approximate the higher order elements. In this example, a support graph preconditioner for the linear triangular elements is constructed, and is used to precondition the quadratic elements (see Section III.5 for detail).

Table V.7 shows the comparison between ICC(0) RCM and MDPSG-10 for the problem with $\mu = 10^{-3}$. We add an additional column for MDPSG, that uses our own

Table V.7

*Comparison of ICC(0) and MDPSG-*10 *for a 2D inhomogeneous unit square Poisson problem (discretized by quadratic triangular elements, with $\mu = 10^{-3}$).*

n	ICC(0) rcm			MDPSG-10			MDPSG-10 (AMD-LDL)		
	#iter	time	$\frac{\|e\|}{\|e_0\|}$	#iter	time	$\frac{\|e\|}{\|e_0\|}$	#iter	time	$\frac{\|e\|}{\|e_0\|}$
26402	166	1.03	3.83×10^{-8}	78	4.83	2.02×10^{-9}	78	0.76	2.02×10^{-9}
52782	219	2.78	1.60×10^{-7}	81	10.22	2.23×10^{-10}	81	1.65	2.23×10^{-10}
105847	298	8.16	9.87×10^{-8}	82	21.02	1.81×10^{-9}	82	3.72	1.81×10^{-9}
211647	418	24.45	2.93×10^{-7}	91	46.70	2.15×10^{-10}	91	8.37	2.15×10^{-10}
423587	586	73.82	3.20×10^{-7}	94	98.04	1.99×10^{-10}	94	18.39	3.64×10^{-10}
847811	805	230.02	7.11×10^{-7}	99	219.07	6.01×10^{-10}	99	42.62	6.01×10^{-10}
1694028	1136	662.96	8.11×10^{-7}	105	446.89	2.15×10^{-9}	105	88.82	2.15×10^{-9}

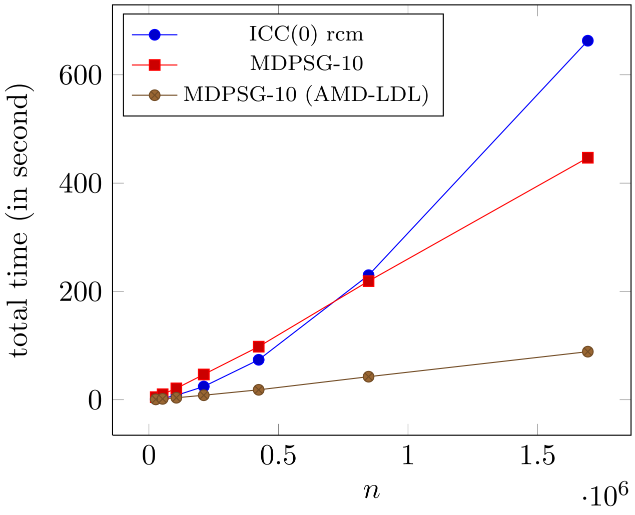


Fig. V.1. *Total time vs. number of unknowns of ICC(0) and MDPSG (a 2D inhomogeneous unit square Poisson problem discretized by quadratic triangular elements, with $\mu = 10^{-3}$).*

Table V.8

*Iteration performance of ICC(0) and MDPSG-*10 *for a 2D inhomogeneous unit square Poisson problem (discretized by quadratic triangular elements, with $\mu = 10^{-3}$).*

n	ICC(0) rcm			MDPSG-10			MDPSG-10 (AMD-LDL)		
	nnzFac	#iter	iter	nnzFac	#iter	iter	nnzFac	#iter	iter
26402	1.63×10^5	166	0.95	9.13×10^4	78	4.20	7.93×10^4	78	0.31
52782	3.26×10^5	219	2.62	2.04×10^5	81	8.91	1.80×10^5	81	0.71
105847	6.57×10^5	298	7.81	4.44×10^5	82	18.21	3.93×10^5	82	1.57
211647	1.32×10^6	418	23.71	9.99×10^5	91	40.83	8.76×10^5	91	3.78
423587	2.64×10^6	586	72.27	2.17×10^6	94	85.78	1.93×10^6	94	8.60
847811	5.28×10^6	805	226.60	4.68×10^6	99	192.80	4.31×10^6	99	20.45
1694028	1.06×10^7	1136	655.71	1.02×10^7	105	396.27	9.42×10^6	105	44.63

PCG solve and Tim Davis's a simple Cholesky solver called *LDL* with an ordering routine called *AMD* ([19]). MDPSG-10 outperforms ICC(0) RCM for a large size problem (notice that MDPSG-10 with AMD-LDL always outperforms ICC(0) RCM for all problems in the table), which is also drawn in Figure V.1. As mentioned before, the intention of this work is to demonstrate the performance of algorithms of MDPSG and ICC, not much on the implementation. The result here shows that MDPSG-10 scales well with the problem size and for large size problems MDPSG-10 outperforms ICC(0). For the implementation issue, let's see the iteration performance data shown in Table V.8. It is obvious that, the performance of PETSc's PCG solve with MUMPS (for factorization) is very poor compared to that with ICC(0). For example, in the row of the problem size of 847811, the number of nonzeros of the factors of MDPSG-10 and ICC(0) are comparable, the number of iterations of ICC(0) is almost 8 times the number of iterations of MDPSG-10, but the iteration time of them are quite similar. However, notice that the iteration time of MDPSG-10 with AMD-LDL is reasonable. Due to the fact that our emphasis is on the algorithms, and MUMPS is basically a parallel direct solver which is convenient for our parallel experiments, so we stick to our implementation choice for MDPSG in this work. The reader, however, should be aware of the fact, that MDPSG can perform much better with a better implementation.

V.3.3. Anisotropic problem

In order to apply the support graph preconditioning technique to the anisotropic problems, the usage of an adaptive mesh is proposed. The idea is that, the mesh edges should be well aligned with the anisotropic axes (see Section III.3 for detail). Consider an anisotropic problem defined in a ring-shape domain that has the inner and outer radius of 2 and 3 unit respectively, the strong anisotropic axis is along the circle direction with anisotropy value 1, while the weak anisotropic axis is along the radius direction with anisotropy value δ, the boundary condition is Dirichlet. The domain is discretized by $\sqrt{N} \times \sqrt{N}$ nodes along both circle and radius directions, where N is the total number of mesh nodes. Though the grid is regularly formed, the indices of nodes are randomly permuted. We tested MDPSG with both triangular and quadrilateral discretizations varying the problem size and anisotropy ratio, the results obtained are similar for both elements.

Shown in Table V.9 is the total time of different PCG solvers with quadrilateral discretization and a fixed smallest anisotropy value $\delta = 10^{-3}$. Also listed in this table is the unpreconditioned CG solver. Compared to CG, ICC(0) does not provide much speed up as the problem size grows, while ICC(1) totally fails (*PETSc* outputs convergence reason -8 for ICC(1) indicating the indefiniteness of the preconditioner). On the other hand, MDPSG may be not impressive for small size problem, but scales well as the problem size increases. Table V.10 shows the typical performance char-

Table V.9

Total time (in second) of different solvers for a 2D ring-shape domain anisotropic problem (discretized by bi-linear quadrilateral elements, the smallest anisotropy value $\delta = 10^{-3}$).

n	CG	ICC(0)		ICC(1)		MDPSG		
		nd	rcm	nd	rcm	10	20	30
39600	6.25	4.49	2.26	X	X	6.20	7.77	9.68
159200	132.30	62.23	34.62	X	X	30.43	39.62	46.37
358800	503.17	224.05	131.95	X	X	75.70	97.31	116.31
638400	1281.72	561.11	328.85	X	X	144.88	177.67	215.60
998000	2462.07	1108.63	654.14	X	X	232.98	284.49	342.66
1437600	4219.64	1880.73	1135.07	X	X	357.79	436.36	527.83

X means divergence due to the indefiniteness of the preconditioner.

Table V.10

Comparison of ICC(0) and MDPSG-30 for a 2D ring-shape domain anisotropic problem (discretized by bi-linear quadrilateral elements, the smallest anisotropy value $\delta = 10^{-3}$).

n	ICC(0) rcm				MDPSG-30			
	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$
39600	1.97×10^{5}	204	2.26	9.93×10^{-9}	3.85×10^{5}	65	9.68	1.10×10^{-10}
159200	7.95×10^{5}	390	34.62	1.85×10^{-8}	1.90×10^{6}	67	46.37	1.51×10^{-10}
358800	1.79×10^{6}	574	131.95	2.10×10^{-8}	4.60×10^{6}	70	116.31	1.28×10^{-10}
638400	3.19×10^{6}	761	328.85	2.63×10^{-8}	8.71×10^{6}	71	215.60	1.06×10^{-10}
998000	4.99×10^{6}	945	654.14	3.51×10^{-8}	1.49×10^{7}	71	342.66	1.14×10^{-10}
1437600	7.18×10^{6}	1125	1135.07	5.55×10^{-8}	2.15×10^{7}	76	527.83	1.04×10^{-10}

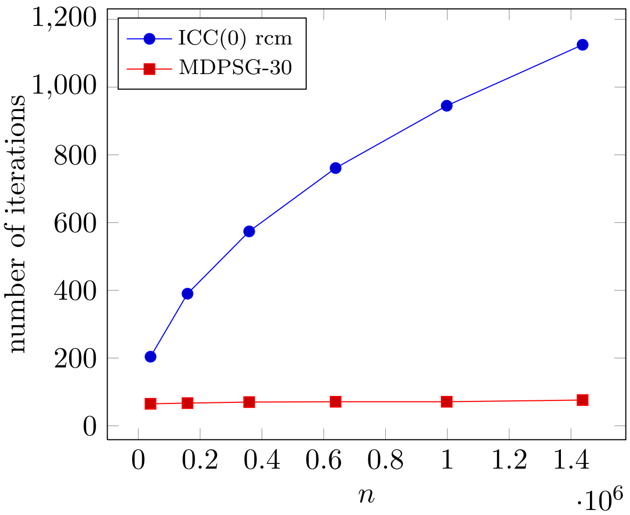


FIG. V.2. *Number of iterations vs. number of unknowns of ICC(0) and MDPSG-*30 *(a 2D ring-shape domain anisotropic problem discretized by quadrilateral mesh, $\delta = 10^{-3}$).*

acteristics of ICC(0) and MDPSG-30. Again, MDPSG gives more accurate solution under the same stopping criteria, and the number of iterations stays more or less the same for a fixed s. The number of iterations of ICC(0) grows proportionally to the square root of the problem size as shown in Figure V.2, this actually follows the asymptotic analysis of ICC(0).

Another observation as shown in Table V.11 is that for a fixed size problem, as the anisotropic ratio increases, i.e. the illness of the problem increases, the number of iteration of ICC(0) also grows. Whereas for MDPSG, the anisotropy may pose difficulty at first, but as the anisotropic problem becomes more like a one dimensional problem, the maximum weight spanning graph of MDPSG turns out to be very effective approximation, which results in a smaller number of iterations. A similar result is also observed in three dimensional space as shown later.

Table V.11

Comparison of ICC(0) and MDPSG for a 2D ring-shape domain anisotropic problem (discretized by quadrilateral elements, the size of the linear system is $n = 1437600$).

δ	ICC(0) rcm				MDPSG-30			
	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$
10^{-1}	7.18×10^{6}	454	464.64	4.56×10^{-7}	1.96×10^{7}	47	350.62	2.40×10^{-10}
10^{-2}	7.18×10^{6}	689	699.47	2.53×10^{-7}	1.95×10^{7}	80	554.40	9.01×10^{-11}
10^{-3}	7.18×10^{6}	1125	1135.07	5.55×10^{-8}	2.15×10^{7}	76	527.83	1.04×10^{-10}
10^{-4}	7.18×10^{6}	1394	1404.38	1.17×10^{-7}	2.15×10^{7}	45	343.35	5.58×10^{-10}
10^{-5}	7.18×10^{6}	1659	1669.66	1.46×10^{-7}	2.11×10^{7}	35	279.27	3.73×10^{-9}
10^{-6}	7.18×10^{6}	2097	2207.67	2.19×10^{-7}	2.11×10^{7}	25	222.73	1.66×10^{-8}
10^{-7}	7.18×10^{6}	2268	2387.78	2.58×10^{-7}	2.08×10^{7}	23	210.51	1.08×10^{-8}

V.4. Three dimensional problems

All three dimensional (3D) serial experiments are conducted on a single node of Hydra (only one processor with maximum available memory 25 GB is used). The condition number of a linear system in 3D is much smaller compared to its 2D counterpart of the same size. ICC(0) performs extremely well as shown in Figure V.3, which shows the total time of different PCG solves for a model Poisson problem ($K(x, y, z) \equiv 1$) with various number of unknowns. The domain is discretized by tetrahedral elements using *Tetgen* [39], with Neumann condition specified at the upper wall and Dirichlet boundary condition at the other walls. Table V.12 is a comparison between ICC(0) RCM and MDPSG-40. These results show that, the difficulty of MDPSG is similar to that of a direct solver in 3D. The large number of *fill-in* during factorization usually kills the performance, which also increases the cost per iteration of the PCG solve. Recall the performance analysis of MDPSG in Section IV.3.2, by balancing the work of factorization and iteration, the theoretical optimal value of t is around $N^{1/4}$ which gives the work complexity of $O(N^{1.2})$ for a 2D problem and $O(N^{1.75})$ for a 3D problem. Since the work complexity of ICC(0) is $O(N^{1.5})$, there is no advantage for MDPSG in solving a 3D Poisson problem. Similar performance of support graph preconditioners for three dimensional problems is also observed by other authors, such as [18, 7].

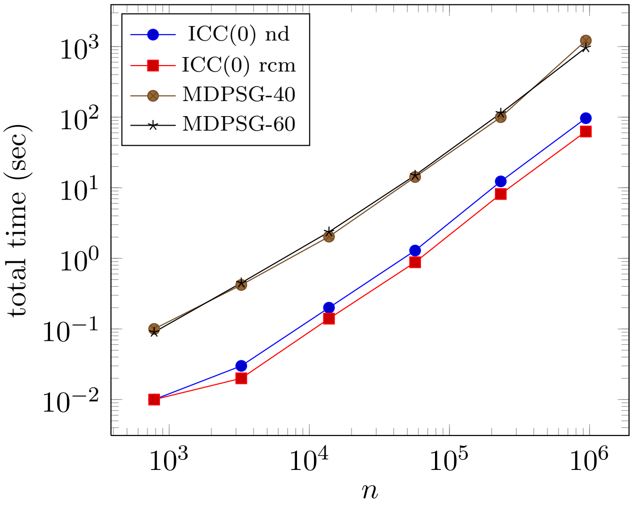


FIG. V.3. *Total time vs. number of unknowns of different PCG solves for a 3D Poisson problem on a unit cube with mixed boundary condition (discretized by tetrahedral elements).*

Despite of the disadvantage as observed above, for difficult problems, MDPSG preconditioners are much more effective than ICC(k). Consider an anisotropic problem on a 3D-ring-shape domain defined as a 2D ring extruded in the vertical direction, see Figure V.4. The inner radius of the domain is 1 unit, the outer radius is 2 unit, and the height is 1 unit. The strong anisotropy are 1 along both the circle direction and the vertical direction, the weak anisotropy is δ $(0 < \delta \leq 1)$ along the radius direction. The domain is discretized with nodes ratio $10 : 50 : 1$ along the circle direction, the radius direction and the vertical direction respectively. Assume that, the number of nodes along the radius direction is larger in order to have enough resolution

TABLE V.12

Comparison between ICC(0) and MDPSG for a 3D Poisson problem on a unit cube with mixed boundary condition (discretized by tetrahedral elements).

n	ICC(0) rcm				MDPSG 40			
	nnzFac	#iter	fac	iter	nnzFac	#iter	fac	iter
781	5.49×10^3	17	0.01	0.00	1.04×10^4	48	0.01	0.07
3269	2.48×10^4	25	0.01	0.01	9.12×10^4	55	0.04	0.30
13815	1.09×10^5	36	0.03	0.11	8.94×10^5	56	0.23	1.38
56865	4.62×10^5	54	0.18	0.71	7.79×10^6	63	2.09	9.58
232369	1.92×10^6	83	1.02	7.13	6.91×10^7	72	33.16	58.00
940390	7.83×10^6	129	5.97	56.60	6.02×10^8	80	731.36	453.93

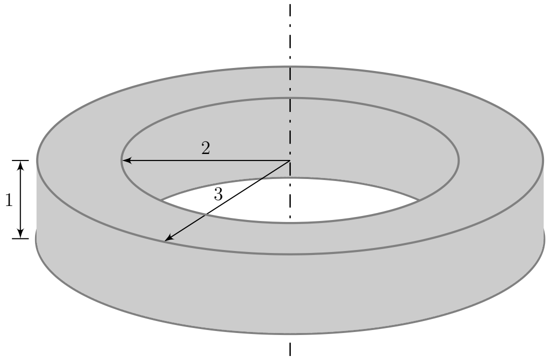


FIG. V.4. *A 3D-ring-shape domain.*

Table V.13
Anisotropic problem of a 3D-ring-shape domain discretized by hexahedral elements (Dirichlet boundary condition, $\delta = 10^{-5}$).

n	ICC(0) rcm		ICC(0) nd		MDPSG 50	
	#iter	time	#iter	time	#iter	time
152460	804	61.38	8411	653.12	36	46.05
392000	994	268.54	12234	3140.30	47	149.56
773670	1334	693.00	17529	9700.59	53	343.27
1608204	2011	3198.46	-	-	59	931.66

- means wall time limit of 6:30:00 hr is exceeded.

Table V.14
Anisotropic problem of a 3D-ring-shape domain discretized by hexahedral elements (Dirichlet boundary condition, number of unknowns $n = 773670$).

δ	ICC(0) rcm				MDPSG 50			
	nnzFac	#iter	time	$\frac{\lVert e \rVert}{\lVert e_0 \rVert}$	nnzFac	#iter	time	$\frac{\lVert e \rVert}{\lVert e_0 \rVert}$
10^{-1}	1.01×10^{7}	157	87.90	6.31×10^{-9}	1.02×10^{8}	93	491.95	1.04×10^{-10}
10^{-2}	1.01×10^{7}	215	128.52	1.63×10^{-8}	1.01×10^{8}	69	412.91	5.28×10^{-10}
10^{-3}	1.01×10^{7}	267	142.59	8.14×10^{-8}	1.04×10^{8}	65	378.49	1.62×10^{-9}
10^{-4}	1.01×10^{7}	572	340.94	9.55×10^{-7}	1.04×10^{8}	61	371.48	6.43×10^{-9}
10^{-5}	1.01×10^{7}	1334	693.00	3.88×10^{-6}	1.01×10^{8}	53	343.27	2.94×10^{-8}
10^{-6}	1.01×10^{7}	2310	1196.36	1.01×10^{-5}	1.02×10^{8}	39	314.46	1.54×10^{-8}
10^{-7}	1.01×10^{7}	2889	1483.82	2.22×10^{-5}	9.94×10^{7}	33	280.99	1.40×10^{-8}

to capture the small scale phenomena. Hexahedral elements are constructed using this regular grid (the indices of nodes are randomly permuted). Table V.13 shows the result for various number of unknowns with fixed $\delta = 10^{-5}$. ICC(0) with RCM ordering always outperforms the one with ND ordering. The number of iterations of ICC(0) increases as the problem size increases, while MDPSG-50 keeps its number of iterations almost unchanged. Table V.14 shows the performance comparison of ICC(0) and MDPSG for a fixed number of unknowns with δ varying. When δ is near 1, ICC(0) outperforms MDPSG-50; however, when the anisotropy ratio grows, i.e., the problem becomes more ill-conditioned, the number of iterations of ICC(0) RCM increases, so is its solver time; whereas MDPSG-50 quickly gives a solution with high accuracy. These results resemble that of the 2D anisotropic problems. The result of ICC(1) is not shown in the tables, since it totally fails for this problem.

V.5. Parallelization

The parallel performance of MDPSG is compared to that of parallel ICC(0) provided by *BlockSolve95*. *BlockSolve95* is a scalable parallel software for solving large sparse linear system. It uses a parallel coloring algorithm that allows for the efficient computation of matrix ordering and scalable performance. The problem to be solved is the same anisotropic problem defined on a 2D ring-shape domain considered early. Parallel performance of the problem with fixed number of unknowns $n = 1437600$ is shown in Table V.15. The result shows that for a fixed-sized problem, BlockSolve95 has a very good parallel performance when the number of processors is smaller than 32 (larger size problem may be needed for a larger number of processors), while the parallelization of MDPSG is not so good. From the performance breakdown as shown in Table V.16, we can see that, the initial three phases of MDPSG, i.e., the M-matrix transformation, the subdomain partition and the support graph formation, parallelize well, thanks to the domain partition feature; while the parallelization of the factorization and the PCG iteration is not very good. The implementation is, no doubt, a factor for this poor performance. However, the direct solve in general does not parallelize well, and the triangular solve is usually the bottleneck of parallelization, is a fact.

Table V.15

Parallel performance of the 2D anisotropic problem of a ring-shape domain with fixed-size $n = 1437600$ *(Dirichlet boundary condition, with* $\delta = 10^{-5}$*).*

p	BlockSolve95				MDPSG 10			
	nnzFac	#iter	time	$\frac{\Vert e\Vert}{\Vert e_0\Vert}$	nnzFac	#iter	time	$\frac{\Vert e\Vert}{\Vert e_0\Vert}$
1	2.16×10^{7}	12565	6324.14	1.44×10^{-7}	4.02×10^{7}	23	115.09	5.73×10^{-9}
2	2.16×10^{7}	8969	2550.83	1.27×10^{-7}	4.34×10^{7}	22	87.42	2.02×10^{-9}
4	2.16×10^{7}	9097	1248.62	1.20×10^{-7}	4.26×10^{7}	22	59.06	1.54×10^{-9}
8	2.16×10^{7}	9160	671.68	1.15×10^{-7}	4.36×10^{7}	22	48.20	1.95×10^{-9}
16	2.16×10^{7}	9068	372.47	1.53×10^{-7}	4.29×10^{7}	22	46.39	4.89×10^{-9}
32	2.16×10^{7}	8938	369.25	3.94×10^{-7}	4.29×10^{7}	22	46.32	4.89×10^{-9}

Despite all these disadvantages, for a difficult anisotropic problem like this, especially for a large problem size, MDPSG can still perform well. A better criteria about the parallel performance of an algorithm is its scalability. For a fixed size problem MDPSG does not scale well with respect to the number of processors; however, since the algorithm itself scales well with the problem size, the scalability of MDPSG can still be very competitive to other algorithms, such as BlockSolve95. Table V.17 shows the performance comparison, the problem size is doubled when the number of processors is doubled. We can see MDPSG-10 is much faster, and scales at least as well as the *BlockSolve95*. (the result holds at least up to 32 processors). This result is

TABLE V.16

Parallel performance breakdown of the 2D anisotropic problem of a ring-shape domain with fixed-size $n = 1437600$ (Dirichlet boundary condition, with $\delta = 10^{-5}$).

p	BlockSolve95		MDPSG 10				
	fac	iter	M-trx	part	sg	fac	iter
1	41.15	6282.99	3.08	19.22	5.87	20.97	65.95
2	30.92	2519.91	1.26	9.89	3.04	16.85	56.38
4	15.34	1233.28	0.64	4.47	1.50	12.73	39.72
8	7.57	664.12	0.33	2.22	0.77	11.10	33.78
16	3.71	368.76	0.21	1.11	0.41	11.34	33.32
32	3.68	365.57	0.18	1.11	0.40	11.28	33.35

TABLE V.17

Parallel performance of the 2D anisotropic problem of a ring-shape domain discretized by quadrilateral elements (Dirichlet boundary condition, with $\delta = 10^{-5}$).

p	n	BlockSolve95				MDPSG 10			
		nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$	nnzFac	#iter	time	$\frac{\|e\|}{\|e_0\|}$
1	89400	1.34×10^{6}	1655	38.99	1.28×10^{-8}	1.72×10^{6}	19	5.71	1.93×10^{-10}
2	178928	2.68×10^{6}	3231	79.13	2.09×10^{-8}	4.08×10^{6}	19	10.38	3.14×10^{-10}
4	358800	5.38×10^{6}	7295	193.22	3.49×10^{-8}	8.89×10^{6}	19	12.43	4.83×10^{-10}
8	717408	1.08×10^{7}	5924	204.25	7.24×10^{-8}	1.94×10^{7}	21	23.10	9.16×10^{-10}
16	1437600	2.16×10^{7}	9068	372.47	1.53×10^{-7}	4.29×10^{7}	22	46.39	4.89×10^{-9}
32	2876415	4.31×10^{7}	13453	1224.41	2.03×10^{-7}	9.46×10^{7}	23	98.30	4.75×10^{-9}

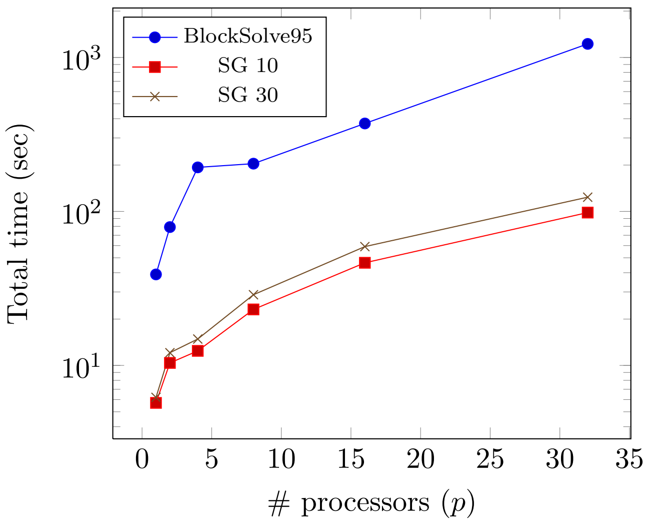


Fig. V.5. *Total time of solving 2D ring-shape domain anisotropic problem with* $\delta = 10^{-5}$*(the number of unknowns is proportional to the number of processors).*

also drawn in Figure V.5. The parallelization of MDPSG can be improved if the subdomain structure taken into consideration during factorization and triangular solve, as demonstrated in our earlier work [46]. Future investigation of MDPSG with other packages that explore this respect should expect a good coarse grain parallelism.

CHAPTER VI

CONCLUSION

This work presents element level transformation techniques to transform SPD matrices arising from the finite element discretization of the second order elliptic boundary problems into symmetric diagonally dominant M-matrices that are suitable for construction of the support graph preconditioners. The central idea of these techniques lies in the coordinate transformation. In terms of graph, basically it can be seen as using a star graph or a union of star graphs to approximate the original graph at the element level. Thus, the spectral bound for the M-matrix approximation $\kappa(A'^{-1}A)$ for arbitrary Lagrangian elements relates to the spectral bound for the simplicial elements. The resulted condition number $\kappa(A'^{-1}A)$ is bounded by the geometric characteristics of the mesh elements, which is actually specified by the spectral number of the outer-product of the Jacobians — $\max_e \kappa(G_e G_e^T)$, where G_e is the Jacobian of the new coordinate system of element e. This actually may provide a mesh quality metric, that can be used to measure how good the matrix A can be approximated by a symmetric diagonally dominant M-matrix A'.

Based on that, a variant of Vaidya's support graph preconditioner called MDPSG is proposed. Experiments are performed along with incomplete Cholesky preconditioners ICC(k) in solving various second order elliptic finite element problems. Like incomplete factorization based preconditioning, support graph preconditioning technique lies between direct and iterative methods. Both methods share the same idea of reducing the *fill-in* of the factors. Unlike incomplete factorization based preconditioners, which reduce the *fill-in* by sparsifying the factors, support graph preconditioners sparsify the matrix first and then factorize completely. The different order of factorization and sparsification results in very different solvers. Experimental results show that, denser factors do not guarantee more effectiveness for ICC(k), while denser support graph increases the effectiveness monotonically for MDPSG. With good ordering, ICC(k) can be very effective and memory efficient; whereas, the relative higher overhead of constructing MDPSG makes it not suitable for easy problems. ICC(k) is sensitive to node ordering and boundary conditions, while MDPSG is almost oblivious to both (notice that the ordering in the factorization phase can affect the factorization cost and iterative cost of MDPSG, but not its convergence rate). MDPSG preconditioners are robust and superior for solving ill-conditioned problems, including large size problems, inhomogeneous problems and anisotropic problems. When the product of congestion and dilation is held constant is held constant, the number of iterations of MDPSG does not change, thus, it scales well with the problem size. In addition, the domain partition feature provides inherent parallelism. Initial results show a good potential of parallelization and scalability of the MDPSG preconditioners.

Our main contribution can be summarized as, we extend the support graph preconditioning technique to the the second order elliptic finite element problems, and propose a new variant of support graph preconditioners called MDPSG which can be used as a general black box algorithm for all symmetric diagonally dominant M-matrices.

Further work is needed in the following aspects. First, the element level transformation technique for the higher order elements has a catch. The condition number of $\kappa(A'^{-1}A)$ depends on the boundness of the coefficient $K(\mathbf{x})$, this may pose difficulty for solving anisotropic problems with high anisotropy ratio. Second, though MDPSG is effective for solving certain difficult problems in three dimensional space, it faces the similar challenge that a direct solver faces. That is, the large number of *fill-in* during factorization usually kills the performance, which also increases the cost per iteration of the PCG solve. Can we design a new scheme of support graph preconditioner that is immune from this challenge? Third, though in practice MDPSG may be effective and scalable compared to other algorithms, the parallelization of the support graph preconditioners in general also faces the same challenge that a direct solver faces. Can we design a new scheme that applies to a wide range of problems, and in the mean time can be as parallelizable as the support tree scheme? Finally, the SPD matrices from the second order elliptic finite element problems, as considered in this work, are known to be the largest class of matrices which the support graph technology can apply to. The next natural step would be to extend the support graph preconditioning technique to a more general class of matrices.

REFERENCES

[1] Patrick Amestoy, Timothy Davis, and Iain Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[2] ——, *Algorithm 837: AMD, an approximate minimum degree ordering algorithm*, ACM Transactions on Mathematical Software, 30 (2004), pp. 381–388.

[3] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent, *A MUltifrontal Massively Parallel sparse direct Solver: Users' guide. MUMPS version 4.7.* Availalble at http://graal.ens-lyon.fr/MUMPS/. Accessed September 2008.

[4] ——, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM Journal on Matrix Analysis and Applications, 23 (2001), pp. 15–41.

[5] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer Methods in Applied Mechanics and Engineering, 184 (2000), pp. 501–520.

[6] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Computing, 32 (2006), pp. 136–156.

[7] Haim Avron, D. Chen, G. Shklarski, and S. Toledo, *Combinatorial preconditioners for scalar elliptic finite-elements problems.* Submitted to SIAM Journal on Matrix Analysis and Applications.

[8] O. Axelsson and V. A. Barker, *Finite element solution of boundary value problems: Theory and computation*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 2001.

[9] Owe Axelsson and Ivar Gustafsson, *Preconditioning and two-level multigrid methods of arbitrary degree of approximation*, Mathematics of Computation, 40 (1983), pp. 219–242.

[10] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang, *PETSc users manual*, Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, Argonne, IL, 2004.

[11] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang, *Portable, extensible toolkit for scientific computation.* Available at http://www.mcs.anl.gov/petsc. Accessed September 2008.

[12] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith, *Efficient management of parallelism in object oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Germany, 1997, Birkhäuser Press, pp. 163–202.

[13] R. Bank and T. Dupont, *Analysis of a two-level scheme for solving finite element equations*, Technical Report CNA-159, The University of Texas, Austin, TX, 1980.

[14] Marshall Bern, John R. Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo, *Support-graph preconditioners*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 930–951.

[15] Erik G. Boman, Doron Chen, Bruce Hendrickson, and Sivan Toledo, *Maximum-weight-basis preconditioners*, Linear Algebra and Its Applications, 11 (2004), pp. 695–721.

[16] Erik G. Boman and Bruce Hendrickson, *Support theory for preconditioning*, SIAM Journal on Matrix Analysis and Applications, 25 (2004), pp. 694–717.

[17] Erik G. Boman, Bruce Hendrickson, and Stephen Vavasis, *Solving elliptic finite element systems in near-linear time with support preconditioners.* To appear in SIAM Journal on Numerical Analysis.

[18] Doron Chen and Sivan Toledo, *Vaidya's preconditioners: Implementation and experimental study*, Electronic Transactions on Numerical Analysis, 16 (2003), pp. 30–49.

[19] Tim Davis, *Research and software development in sparse matrix algorithms.* Availalble at http://www.cise.ufl.edu/research/sparse. Accessed September 2008.

[20] Howard C. Elman, David J. Silvester, and Andrew J. Wathen, *Finite elements and fast iterative solvers: With applications in incompressible fluid dynamics*, Oxford University Press, Oxford, 2005.

[21] Alan George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.

[22] Alan George and Joseph W. H. Liu, *A fast implementation of the minimum degree algorithm using quotient graphs*, ACM Transactions on Mathematical Software (TOMS), 6 (1980), pp. 337–358.

[23] Alan George and Joseph W-H. Liu, *Computer solution of large sparse positive definite systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[24] Keith D. Gremban, *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems*, Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Oct. 1996.

[25] K. D. Gremban, G. L. Miller, and M. Zagha, *Performance evaluation of a parallel preconditioner*, in 9th International Parallel Processing Symposium Proceedings, Santa Barbara, CA, Apr. 1995, IEEE Computer Society, pp. 65–69.

[26] Louis A. Hageman and David M. Young, *Applied Iterative Methods*, Academic Press, New York, NY, 1981.

[27] Michael T. Heath, Esmond Ng, and Barry W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Review, 33 (1991), pp. 420–460.

[28] Mark T. Jones and Paul E. Plassmann, *Blocksolve95 users manual: Scalable library software for the parallel solution of sparse linear systems*, Technical Report ANL-95/48, Argonne National Laboratory, 1995.

[29] George Karypis, *METIS - family of multilevel partitioning algorithms.* Availalble at http://glaros.dtc.umn.edu/gkhome/views/metis. Accessed September 2008.

[30] George Karypis and Vipin Kumar, *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed Computing, 48 (1998), pp. 96–129.

[31] Patrick M. Knupp, *Algebraic mesh quality metrics*, SIAM Journal on Scientific Computing, 23 (2001), pp. 193–218.

[32] Stig Larsson and Vidar Thomee, *Partial Differential Equations with Numerical Methods (Texts in Applied Mathematics)*, vol. 45, Springer, Berlin, 1st ed., 2003. Corr. 2nd printing 2005.

[33] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan, *Generalized nested dissection*, SIAM Journal on Numerical Analysis, 16 (1979), pp. 346–358.

[34] Keith Miller, *On the mass matrix spectrum bounds of wathen and the local moving finite elements of baines*, SIAM Journal on Numerical Analysis, 29 (1992), pp. 89–106.

[35] James M. Ortega, *Numerical Analysis: A Second Course*, SIAM Edition, Philadelphia, PA, 1990.

[36] Yousef Saad, *ILUT: A dual threshold incomplete LU factorization*, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402.

[37] ——, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, PA, 2nd ed., 2003.

[38] Jonathan Richard Shewchuk, *A two-dimensional quality mesh generator and delaunay triangulator*. Availalble at http://www.cs.cmu.edu/~quake/triangle.html. Accessed September 2008.

[39] Hang Si, *A quality tetrahedral mesh generator and three-dimensional delaunay triangulator: Users's manual of version 1.4.1*. Availalble at http://tetgen.berlios.de/. Accessed September 2008.

[40] Daniel Spielman and Shang-Hua Teng, *Nearly linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceed-

ings of the 44th Annual IEEE Symposium on Foundations of Computer Science, Cambridge, MA, Oct. 2003, IEEE Computer Society, pp. 416–427.

[41] Lloyd N. Trefethen and III David Bau, *Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[42] P. 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, Oct. 1991.

[43] A. van der Sluis, *Condition numbers and equilibration of matrices*, Numerische Mathematik, 14 (1969), pp. 14–23.

[44] Henk A. van der Vorst, *Krylov subspace iteration*, Computing in Science & Engineering, 2 (2000), pp. 32–37.

[45] ——, *Efficient and reliable iterative methods for linear systems*, Journal of Computational and Applied Mathematics, 149 (2002), pp. 251–265.

[46] Meiqiu Wang and Vivek Sarin, *Parallel support graph preconditioners*, in Proceedings of the International Conference on High Performance Computing (HiPC). 13th International Conference, Bangalore, India, December 18-21, 2006., vol. 4297 of Lecture Notes in Computer Science, Springer-Verlag, Berlin, Dec. 2006, pp. 387–398.

[47] Andrew. J. Wathen, *Realistic eigenvalue bounds for the Galerkin mass matrix*, IMA Journal of Numerical Analysis, 7 (1987), pp. 449–457.

[48] X. Wu, M. S. Downes, T. Goktekin, and F. Tendick, *Adaptive nonlinear finite elements for deformable body simulation using dynamic progressive meshes*,

in EG 2001 Proceedings, A. Chalmers and T.-M. Rhyne, eds., vol. 20, Boston, MA, 2001, Blackwell Publishing, pp. 349–358.

[49] Jinchao Xu and Ludmil Zikatanov, *A monotone finite element scheme for convection-diffusion equations*, Mathematics of Computation, 68 (1999), pp. 1429–1446.

APPENDIX A

TRIANGULAR ELEMENT AND M-MATRIX

At element level, let matrix D_e denotes the variable coefficient, the element stiffness matrix for the general Poisson problem can be written as,

$$\begin{aligned} A_e &= \int_{\Omega_e} B_e^T D B_e d\Omega_e \\ &= \frac{1}{J_d^2} \int_{\Omega_e} \begin{bmatrix} y_2 - y_3 & x_3 - x_2 \\ y_3 - y_1 & x_1 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix} D_e \begin{bmatrix} y_2 - y_3 & y_3 - y_1 & y_1 - y_2 \\ x_3 - x_2 & x_1 - x_3 & x_2 - x_1 \end{bmatrix} dxdy \end{aligned} \tag{A.1}$$

where $J_d = 2\Delta_e$, with Δ_e being the area of element e. For the classic Poisson problem, i.e., D_e is an identity matrix, we have,

$$A_e = \frac{1}{2J_d} \begin{bmatrix} d_1^2 & -d_1 d_2 \cos\theta_3 & -d_1 d_3 \cos\theta_2 \\ -d_1 d_2 \cos\theta_3 & d_2^2 & -d_2 d_3 \cos\theta_1 \\ -d_1 d_3 \cos\theta_2 & -d_2 d_3 \cos\theta_1 & d_3^2 \end{bmatrix} \tag{A.2}$$

where d_i is the length of the edge facing vertex i, and θ_i is the angle facing the vertex i, for $i = 1, 2, 3$. Apparently, if any θ_i is greater than $\pi/2$, then it is possible that the global stiffness matrix may not be an M-matrix. There is a condition concerning the mesh characteristics for the P_1's stiffness matrix to be an M-matrix, for example, is given in Xu's paper [49]. If D_e presents, then things become more complicated. In general, the global stiffness matrix A is SPD.

APPENDIX B

NOTES ON METIS

The graph partitioning problem is to partition the vertices of a graph in k roughly equal parts, such that the edge cut among different parts is minimized. The graph partitioning problem is NP-complete. However, many algorithms have been developed that find reasonably good partition. One of them is the multilevel k-way partitioning scheme [30], which is implemented in the software package – *Metis*. *Metis* is a highly regarded library for partitioning unstructured graphs. Compared to other graph partitioning algorithms, it not only runs faster but gives a high quality partition. Many public domain softwares use it either to partition domain, or to compute fill-reducing ordering of sparse matrices. The *Metis* algorithms compute a k-way partitioning of a graph $G = (V, E)$ in $O(|E|)$ time, under the assumption that $k \log k$ is less than $|E|$, which is usually the case. Our MDPSG algorithm also employs *Metis* to partition the graph into subdomains. However, for a large number of partitions k, *Metis* is likely run in $O(n \log n)$ time, where $|V| = n$ and $k \sim O(n)$. For a large k, *Metis* also encounters difficulty: it requires a huge memory and runs slow.

The algorithms in *Metis* are based on multilevel graph partitioning. Multilevel partitioning algorithms first reduce the size of graph by collapsing vertices and edges, partition the smaller graph, and then uncoarsen it to construct a partition for the original graph. Notice that if the partition number is large, then the advantage of coarsening and uncoarsening procedure will not be fully taken. In our applications, the exact partition of k is not required, instead it requires that the number of nodes in each partition should below a specified threshold s. This flexibility enables us to solve the *Metis*' huge memory consumption problem and improve its efficiency. The basic idea is to use a hierarchical scheme: partition the graph into an K_i number of partitions at level i, such that the number of nodes in each partition is less than N_i; then split the graph into K_i separate graphs and for each separate graph go to the next level of partition. At each level, K_i is less than or equal to K_δ, where K_δ is the assumed upper limit of partition number that *Metis* is deemed to run efficiently. And N_i decreases monotonically at each level; at the last level it should be less or equal to s. For reference, this hierarchical calling of *Metis* is noted as Hierarchy-Metis.

Table B.1

Performance of Hierarchy Metis and Metis.

$\|V\|$	$\|E\|$	Hierarchy-Metis		Metis	
		# partition	time (sec)	# partition	time (sec)
1611	4677	162	0.02	162	0.02
3228	9445	324	0.05	323	0.06
6507	19204	652	0.11	651	0.14
13099	38821	1312	0.23	1310	0.37
26272	78160	2635	0.42	2628	0.96
52625	156906	5280	0.87	5263	2.61
105556	315371	10585	1.73	10556	7.62
211362	632138	21189	3.60	21137	35.95
423329	1267391	42436	7.23	-	-
846728	2536287	84897	17.09	-	-

- means not solvable.

Table B.1 shows the performance comparison between Hierarchy-Metis and Metis on the Pentium 4 machine *Wilkinson* with 2GB RAM. Here, we use $K_\delta = 256$, and $s = 10$. The result shows that Hierarchy-Metis not only runs faster than Metis, it also can solve problems with much large size. Experiments show that Hierarchy-Metis is not sensitive to the value of K_δ. For our applications the range of $200 \sim 1000$ is recommended.

VITA

Meiqiu Wang received her B.S. and M.S. in mechanics from Peking University in 1992 and 1995 respectively. She received her M.S. in computer science from the University of Houston in 2003. She entered the Ph.D. program at Texas A&M University in August 2003 and graduated in 2008. Her research interests include numerical algorithms, high performance computing, distributed and parallel computing, pattern recognition, machine learning and software development.

Ms. Wang may be reached at Department of Computer Science, c/o Dr. Vivek Sarin, Texas A&M University, College Station, TX 77843-3112. Her email address is mqwang@cs.tamu.edu.

This document was typeset in LaTeX by Meiqiu Wang.