Lecture Notes: Support Tree Preconditioners for Laplacian Matrices

Stephen Guattery
ICASE

April 22, 1997

Recall that we are considering solving

$$B^{-1}A\mathbf{x} = B^{-1}\mathbf{b}$$

using the Preconditioned Conjugate Gradient (PCG) algorithm. Last week we covered the use of embeddings to bound the spectral condition number of preconditioned systems when A and B are Laplacian matrices (recall that we are using the generalized definition of Laplacian from Keith Gremban's thesis). This week we will apply this background material to understand the intuition behind Gremban's *support tree preconditioners*, and his analysis of the quality of such preconditioners when used in PCG for Laplacians. The material in this lecture is drawn from Chapters 3 and 4 of Gremban's Ph.D. thesis unless otherwise noted.

1 Goals for Preconditioners

We can state two key goals of Gremban's work as follows:

1. Apply the concept of **support** in developing good preconditioners for Laplacians. How can one use the graph structure of the matrix A in developing the graph structure of preconditioner B such that A and B support each other?

2. The structure of B should make it easy to construct and to solve *in parallel.*

Gremban discusses an additional intuitive idea behind his work that, in my opinion, doesn't apply. He observes that convergence rates for some iterative algorithms depend on the diameter of the graph of the matrix A. The matrix multiplication step in these algorithms propagates information by passing it from a vertex to that vertex's neighbors; that information must be spread to the farthest parts of the graph and back before a solution can converge (see the example on p. 39 of the thesis). Gremban correctly notes that a good preconditioner will speed this process. However, he suggests that the graph structure of the preconditioner should reflect this by having short distances between all pairs of vertices. Thus, preconditioners should have small diameters.

However, it is the graph structure of the *inverse* of the preconditioner that affects the propagation of information. To see this, consider a line on n vertices. The best preconditioner for this line is itself. However, the line as preconditioner has diameter n, the same as the original system.

There are a number of topics to cover today:

- Support tree preconditioners.
- The Support Tree Conjugate Gradient (STCG) algorithm.
- Bounding the spectral condition number κ for support trees.
- Fast parallel solution of tree-structured matrices.

2 Building Support Tree Preconditioners

A support tree preconditioner is a Laplacian preconditioning matrix whose associated graph is a tree on a superset of the vertices of A (For simplicity of notation, I will use A to represent both the matrix and the graph $G(A)$ associated with that matrix). The tree is constructed so that it provides good support for A. Each vertex in the graph of the matrix becomes a leaf of the support tree. The internal vertices of the tree are added vertices.

How do we assign edges in the support tree? Note that every edge in the original graph has to be supported by a path in the tree. We want to keep congestion and dilation down, so intuitively we want to keep paths short, and to minimize the number of edges supported by long paths. We also want the tree to reflect the graph structure; tree edges should correspond to graph edges somehow.

This suggests the following method for constructing the support tree:

1. If the graph consists of a single vertex, return that vertex as the support tree. The weight of this vertex in the support tree is equal to the surplus in the diagonal entry for this vertex in the original matrix, which is the weight of any edges from that vertex to the zero Dirichlet boundary.
2. Otherwise, find a small edge separator S that gives a balanced (e.g., $\frac{1}{3}/\frac{2}{3}$) cut. Assign the root of the tree r to this separator.
3. Recursively build support trees for the components $G_1, G_2, \ldots, G_k$ left after removing the separator.

4. Add edges from the roots of the subtrees of the components to r. The weights of these edges are based on the weight of the edges in S and higher-level separators that are incident to G_i: The *frontier* of component G_i is the set of edges with one end in G_i, and the other in a different component. The *frontier weight* of G_i is the sum of the edge weights of its frontier. Let r_i be the root of the support tree for G_i; then the weight of the edge from r_i to r is the frontier weight of G_i. (Note: Gremban uses a different way of setting these weights in Chapter 5 of his thesis, which covers unstructured graphs. The best way of setting these weights is a possible area for further research.)

The changes above are described in graph terms; they are actually implemented in terms of constructing the Laplacian corresponding to the support tree.

An example of building a support tree is given in Figure 3.5 on p. 45 of the thesis.

3 How Does the Matrix Support the Support Tree?

Intuitively, it is clear that the support tree provides good support of the original matrix (we will prove this for a simple case below, and additional analyses are in Gremban's thesis). However, there are two obvious questions that may be bothering the reader at this point:

- How can the original matrix support the tree? They're not even on the same set of vertices!
- How can we use the support tree preconditioner in PCG?

These are very closely related questions. To answer them, we must consider what happens when we use Gaussian elimination on the internal vertices of the support tree; in particular, we need to study the resulting Schur complement K.

A few preliminary notational conventions are necessary: assume the original matrix A is $n \times n$, and that the construction of the support tree requires m additional vertices. We will denote the Laplacian of the support tree by T, an $(n+m) \times (n+m)$ matrix. For simplicity, we assume that we have numbered the vertices of A, and all the added vertices in T have numbers greater than n. Thus, the leaves of the support tree are numbered first, followed by the internal (non-leaf) vertices.

Now consider what happens if we perform Gaussian elimination on T. In particular, we will eliminate all the internal vertices by applying elimination to both the rows and columns. Row elimination can be represented by multiplying

T on the left by a matrix G (using Gremban's notation); column elimination is represented by multiplying by G^T on the right. It is easy to see that

$$G = \begin{pmatrix} I_n & H_{21} \\ 0 & H_{22} \end{pmatrix},$$

where I_n is the $n \times n$ identity matrix, and H_{21} and H_{22} are $n \times m$ and $m \times m$ respectively.

Eliminating the internal vertices of T produces a matrix $\hat{T}$:

$$\hat{T} = GTG^T = \begin{pmatrix} K & 0 \\ 0 & D \end{pmatrix},$$

where K is the $n \times n$ Schur complement and D is an $m \times m$ diagonal matrix. Gremban gives an interesting interpretation of the Schur complement in electrical terms, noting that (with respect to the remaining vertices) it is an equivalent circuit to the support tree. This gives some intuition about how A can be used to support T: A needs to support only the point-to-point conductances of all pairs of leaves of T. More detailed discussion of this analogy is beyond the scope of these notes.

The goal of the rest of this part of the lecture is to show that the STCG algorithm presented on p. 47 produces the same sequence of $\mathbf{x}^{(i)}$'s as does PCG when K is used as a preconditioner, and to use that fact in getting a bound on the spectral condition number of the preconditioned system.

3.1 Behavior of STCG

(**Notation:** For the rest of this section, we will be using vectors of both length n and length $n+m$. Vectors of length $n+m$ will be indicated by tildes; the $n+m$ vector $\tilde{\mathbf{y}}$ has two components. The first, $\mathbf{y}_1$, consists of the first n entries; $\mathbf{y}_2$ consists of the last m entries.)

Note that in PCG, the preconditioner B is used to modify the residual, which is also the gradient at the current $\mathbf{x}^{(i)}$. In particular, if $\mathbf{g}$ is the gradient, the value $\mathbf{h} = B^{-1}\mathbf{g}$ is used. STCG also works with the gradient $\mathbf{g}$. However, since T is a larger matrix, $\mathbf{g}$ must be extended to a longer vector $\tilde{\mathbf{g}}$ by adding m zeros. Likewise, the resulting $\tilde{\mathbf{h}}$ must be truncated:

$$\tilde{\mathbf{h}} = \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{pmatrix} = T^{-1}\tilde{\mathbf{g}} = T^{-1}\begin{pmatrix} \mathbf{g} \\ \mathbf{0} \end{pmatrix}.$$

The vector $\mathbf{h}_1$ is used as the modified residual/gradient.

We will show that $\mathbf{h}_1 = K^{-1}\mathbf{g}$, where K is the Schur complement left after eliminating the internal vertices of the support tree. To do this, we need two lemmas:

Lemma 3.1 *For* $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix}$, *let* $T^{-1}\tilde{\mathbf{x}} = \tilde{\mathbf{y}}$ *and* $\hat{T}^{-1}\tilde{\mathbf{x}} = \tilde{\mathbf{z}}$. *Then* $\mathbf{y}_1 = \mathbf{z}_1$.

Proof: Recall that $\hat{T} = GTG^T$. Noting that $\hat{T}^{-1} = (G^T)^{-1}T^{-1}G^{-1}$, we can perform the following calculation:

$$\tilde{\mathbf{z}} = \hat{T}^{-1}\tilde{\mathbf{x}} = (G^T)^{-1}T^{-1}G^{-1}\tilde{\mathbf{x}}.$$

As noted in Section 3, $G = \begin{pmatrix} I_n & H_{21} \\ 0 & H_{22} \end{pmatrix}$. Thus $G\tilde{\mathbf{x}} = \tilde{\mathbf{x}}$. Since G is invertible (the steps of Gaussian elimination can be done in reverse), this implies that $\tilde{\mathbf{x}} = G^{-1}\tilde{\mathbf{x}}$. Using these facts and multiplying both sides through by G^T gives

$$G^T\tilde{\mathbf{z}} = T^{-1}G^{-1}\tilde{\mathbf{x}} = T^{-1}\tilde{\mathbf{x}} = \tilde{\mathbf{y}}.$$

Since

$$G^T\tilde{\mathbf{z}} = \begin{pmatrix} \mathbf{z}_1 \\ H_{21}^T\mathbf{z}_1 + H_{22}^T\mathbf{z}_2 \end{pmatrix},$$

$\mathbf{z}_1 = \mathbf{y}_1$.
□

Lemma 3.2 *For* $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ \mathbf{0} \end{pmatrix}$, *let* $\hat{T}^{-1}\tilde{\mathbf{x}} = \tilde{\mathbf{z}}$. *Then* $\mathbf{z}_1 = K^{-1}\mathbf{x}$.

Proof: Recall that

$$\hat{T} = GTG^T = \begin{pmatrix} K & 0 \\ 0 & D \end{pmatrix},$$

so

$$\hat{T}^{-1} = \begin{pmatrix} K^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix}.$$

The lemma follows immediately.
□

Together, Lemmas 3.1 and 3.2 imply (modulo any numerical effects) that STCG using T will behave exactly the same as PCG using K in the sense that the sequence of $\mathbf{x}^{(i)}$'s (and hence the error norms) for both methods will be the same.

3.2 Bounding $\kappa(K^{-1}A)$: $\lambda_{\max}$

We now turn to the question of bounding the number of iterations needed for convergence of STCG. The results of the previous sections plus the material covered last week imply that $\kappa(K^{-1}A)$ provides an upper bound, and that $\kappa(K^{-1}A) = \sigma(A,K)\sigma(K,A)$. Unfortunately, K is almost certainly dense and messy to work with. However, the good news is that we can use T's support of A instead of $\sigma(A,K)$.

Since the support number is defined in terms of matrices of the same size, we define the matrix $\tilde{A}$ such that $\tilde{a}_{ij} = a_{ij}$ for $i \leq n$ and $j \leq n$, and $\tilde{a}_{ij} = 0$

otherwise. That is, the leading $n \times n$ submatrix of $\tilde{A}$ is A, and the rest of $\tilde{A}$ is zero.

The following lemma is a version of Lemma 4.9 in Gremban's thesis. Let $\Lambda(M)$ denote the set of eigenvalues of matrix M.

Lemma 3.3 *For B s.p.d., A symmetric positive semidefinite, and G invertible, $\lambda \in \Lambda\left(B^{-1}A\right)$ if and only if $\lambda \in \Lambda\left((GBG^T)^{-1}(GAG^T)\right)$.*

Proof: Let λ be an eigenvalue of $B^{-1}A$ with eigenvector $\mathbf{u}$. Then $B^{-1}A\mathbf{u} = \lambda\mathbf{u}$, and $A\mathbf{u} = \lambda B\mathbf{u}$. Hence we have

$$GA\mathbf{u} = \lambda GB\mathbf{u}.$$

Since G is invertible, G^T is also invertible, so there exists a $\mathbf{y}$ such that $G^T\mathbf{y} = \mathbf{u}$. Substituting gives

$$GAG^T\mathbf{y} = \lambda GBG^T\mathbf{y},$$

and we have proved one direction. The argument in the other direction is symmetric, and is left as an exercise.
□

We have the immediate corollary:

Corollary 3.4 $\Lambda\left(T^{-1}\tilde{A}\right) = \Lambda\left(\hat{T}^{-1}\tilde{A}\right)$.

Proof: This follows, since $\hat{T} = GTG^T$ (G is invertible as previously noted), and since

$$G\tilde{A}G^T = \begin{pmatrix} I_n & H_{21} \\ 0 & H_{22} \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} I_n & 0 \\ H_{21}^T & H_{22} \end{pmatrix} = \tilde{A}.$$

□

We now have enough tools to prove the following theorem:

Theorem 3.5 $\lambda_{\max}(K^{-1}A) = \lambda_{\max}(T^{-1}\tilde{A})$.

Proof: By Corollary 3.4, we can prove the theorem for $\lambda_{\max}(\hat{T}^{-1}\tilde{A})$. Recalling the structures of $\hat{T}^{-1}$ and $\tilde{A}$, we have

$$\hat{T}^{-1}\tilde{A} = \begin{pmatrix} K^{-1} & 0 \\ 0 & D^{-1} \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} K^{-1}A & 0 \\ 0 & 0 \end{pmatrix}$$

It is easy to show that the eigenvalues of this matrix are the n eigenvalues of $K^{-1}A$ plus 0 with multiplicity m; this is left as an exercise.
□
Thus, we can use T's support for $\tilde{A}$ in bounding $\kappa(K^{-1}A)$.

3.3 Bounding $\kappa(K^{-1}A)$: $\lambda_{\min}$

The situation for $\lambda_{\min}(K^{-1}A)$ is not quite as nice. In general, developing bounds for this eigenvalue requires proving properties about K and its graph, which in turn are used in proving properties of embeddings of K into A. This can require some fairly technical machinery. However, for certain model problems the analysis can be reasonable. We will cover that analysis for a path graph below; the analysis for regular grids is in Chapter 4 of Gremban's thesis. Analysis for more general cases is included in Chapter 5, but is beyond the scope of these lectures. A generalized bound for all Laplacians is an open question.

4 Analysis and Experimental Results

We now turn to some results covered in Gremban's thesis. We will start by giving an example of the analysis of a simple case, and then summarizing his results for regular grids. We follow this with a summary of some experimental results covered in the thesis.

4.1 An Example: Unweighted Paths

Let A be the Laplacian of the path graph P on $n = 2^k$ vertices. Assume that the two ends of the path have unit-weight edges to the zero boundary, though this won't affect the analysis: the same edges to boundary occur in A, T, and K [1]. The support tree T for this example will be a complete binary tree with n leaves and $2n-1$ vertices. The graph and its support tree for $k=3$ are shown in Figure 1 below. It is easy to see that the frontier weight of the subgraph induced by any subtree is either 1 or 2. Since we are interested in asymptotic bounds on the condition number, we set the weight of each edge in T to 1, which changes the numbers in the analysis by at most a small constant factor.

It is easy to show that $\sigma(A,T) = O(\log n)$. This involves embedding A into T. The dilation is clearly twice the height of the tree, which is the length of the path supporting the middle edge of the path. This is $2k = 2\log n$. We will show that the congestion of the embedding is 2, which will prove the claimed upper bound. We will show the result about the congestion using induction; for ease of presentation, we will cheat a little bit and use a description of the tree based on Figure 1 rather than formal mathematical notation. That figure lays the tree out from left to right; the path from the left-most leaf to the root will be the *left path*. The *right path* is similarly defined. Note that we can identify a left and right path for every subtree as well.

[1] Note that if there are many large-weight edges to the boundary, then the analysis could change, as edges could be supported by paths using only edges to and from the boundary. However, we will assume that is not the case.

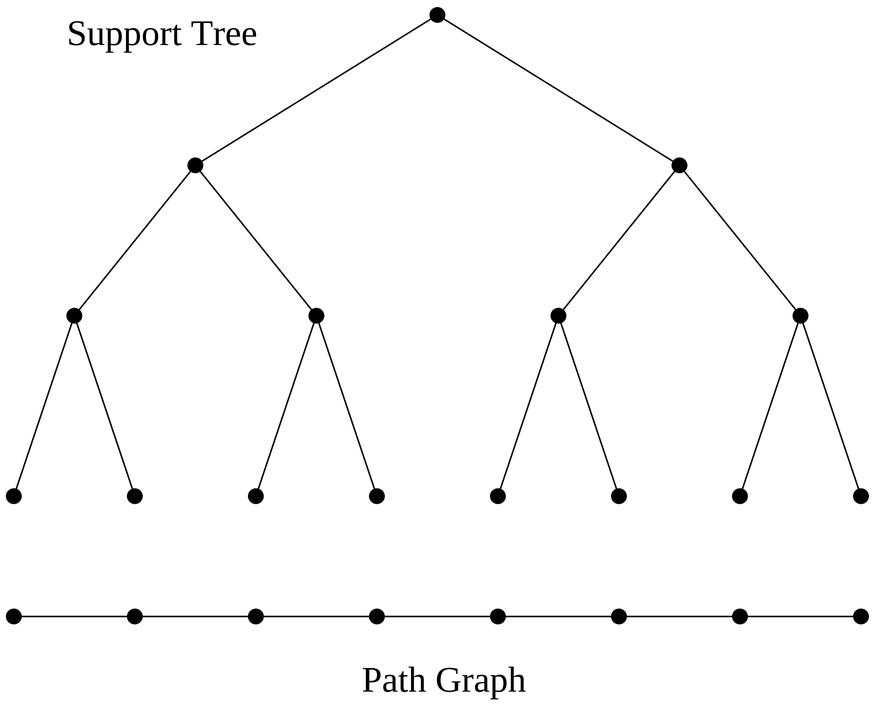

Figure 1: A Path and Its Support Tree

CLAIM: In any subtree S of the support tree T, the maximum congestion from supporting the subpath of P induced by the leaves of S is 2. Further, the congestion on the edges of the left path and the right path of S is 1.

To prove the claim, start with the base case consisting of a pair of leaves with a common parent. The congestion on the two edges (the left and right paths) is 1, which is also the maximum congestion. Now assume that we have proved the result for subtrees of height i and go to a subtree of height $i+1$. This joins two adjacent subtrees of height i; the induced subpath of P consists of the subpaths induced by the two smaller subtrees plus a single edge connecting the rightmost leaf of one of the smaller trees to the leftmost leaf of the other. To support this edge, we increase the congestion on the right path of the left tree by one, and the left path of the right tree by one (both of these paths are internal to the larger subtree). This edge also contributes a congestion of 1 to each edge from a root of one of the height i subtrees to the root of the height $i+1$ subtree. The claim clearly holds.

We have proved the following lemma:

Lemma 4.1 $\sigma(A,T) = O(\log n)$.

We now must address the question of embedding K into A. To start, consider the structure of K. Recall that K is the Schur complement that is left after the internal vertices of T are eliminated; the entries in K are independent of the order of elimination. If we consider a top-down elimination order (i.e., the root is eliminated first, then all vertices at distance 1 from the root, then all vertices at distance 2, etc.), an easy induction argument shows that K is a complete graph on the leaves of T: Let $T^{(i)}$ be the graph remaining after i elimination steps, and let $V_{\text{top}}^{(i)}$ be the vertices in $T^{(i)}$ whose parents have been eliminated. The induction hypothesis is that the subgraph induced by $V_{\text{top}}^{(i)}$ forms a clique. The base case is $T^{(1)}$ after the root is eliminated; its children will be connected by a fill edge. To see that the hypothesis holds after each subsequent elimination step, note that as each internal vertex is eliminated, each of its children gets fill edges to its sibling and all other vertices already in the clique. Note that this argument implies the claimed structure of K, which is $T^{(i)}$ when $V_{\text{top}}^{(i)}$ consists of the leaves of T.

The results of this elimination order are shown in Figure 2 for the case in which the top two levels of the tree have been eliminated. The figure also shows the weights of the edges in the clique.

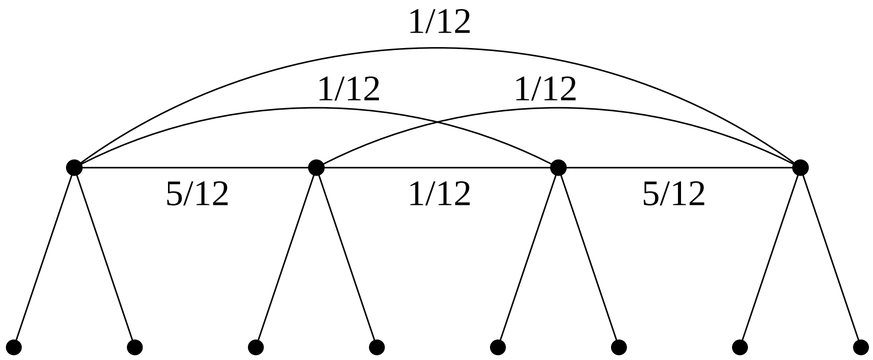

Figure 2: The Tree After Reducing Two Levels

The edges of K have the following interesting property: Let u and v be leaves of T such that the path between u and v passes through j internal vertices of T. Then the weight of edge (u, v) left after elimination is less than or equal to 2^{-j}. The proof of this fact in the thesis contains an error; and explanation of the error and a corrected proof are included in the Appendix (the exposition is long enough that it would disrupt the current argument if included here). The interested reader can check the details.

We now want to divide the edges of K into sets based on the number of internal vertices on the paths in T that connect their endpoints. What does a leaf-to-leaf (u, v) path through j internal vertices of T look like? It must run from u up to the least common ancestor of u and v, then back down to v. The number of edges on the path up must be the same as the number on the path down, so the least common ancestor must be $\frac{j+1}{2}$ levels higher than the leaves. Note that for a different number j' of internal nodes, a path will have its highest point at a different level. Note also that that any internal vertex w is the root of a subtree of T, and that all pairs of leaves of w's subtree have w as a least common ancestor. Thus the number of sets of edges from K is equal to the number of levels of internal vertices in T. Recall that the path P is on $n = 2^k$ vertices for some integer k, and that T has $2n-1$ vertices. A simple calculation shows that there are k sets of edges.

We will use this division of K into sets of edges to bound the support of A for K. The basic idea is to split A into k positive definite pieces, and then bound the support of A_i for K_i. As we showed last time, the maximum support taken over $1 \leq i \leq k$ will be the bound for the whole problem. The following lemma gives the details:

Lemma 4.2 $\sigma(K, A) = O(n)$.

Proof: Index the vertices in T by their heights above the leaves: leaves are at level 0, and T's root is at level k (note that this numbering of levels is opposite Gremban's level numbering when you look at the thesis; I think it's clearer this way). Based on the discussion above, we will associate the vertices at level i with the edges in K whose corresponding leaf-to-leaf paths contain exactly $2i-1$ internal vertices. We will denote this set by K_i; as noted above, the weights of the edges in K_i are bounded above by $\frac{1}{2^{2i-1}}$.

Divide A into k pieces as follows: for $1 \leq i \leq k$, $A_i = \frac{1}{2^{k+1-i}}A$. Note that these pieces add up to slightly less than A, but since we are embedding K into A, this works in our favor.

We now give a bound for $\sigma(K_i, A_i)$ (various points brought up in the analysis are illustrated in Figure 3 below for $i = 2$ and $n = 8$). The vertices at level i are the roots of subtrees of T. Consider any edge e in K_i: the endpoints of e must be leaves from a single such a subtree since their least common ancestor is at level i. We can be even more specific: Using the left-to-right layout convention described above, a subtree rooted at u on level i has a left child v_l and a right child v_r. Any edge in K_i that has u as the least common ancestor of its endpoints must have one end in the leaves of v_l's subtree, and the other in the leaves of v_r's subtree; this follows from the definition of least common ancestor.

These facts suggest that we can use the subtrees rooted at level i to partition the edges of K_i into disjoint sets. Since the weights of the edges in K_i and A_i are the same for each such subtree, we need only do the analysis for one. We now restrict our attention to one such subtree (this is shown by the heavier edges in Figure 3). Note that its leaves induce a subpath P_i of P; every edge of K_i for the subtree we are studying will be embedded into P_i.

We can now bound the congestion c_i and dilation d_i of the embedding of K_i into A_i. The length of P_i is one less than the number of leaves of the subtree,

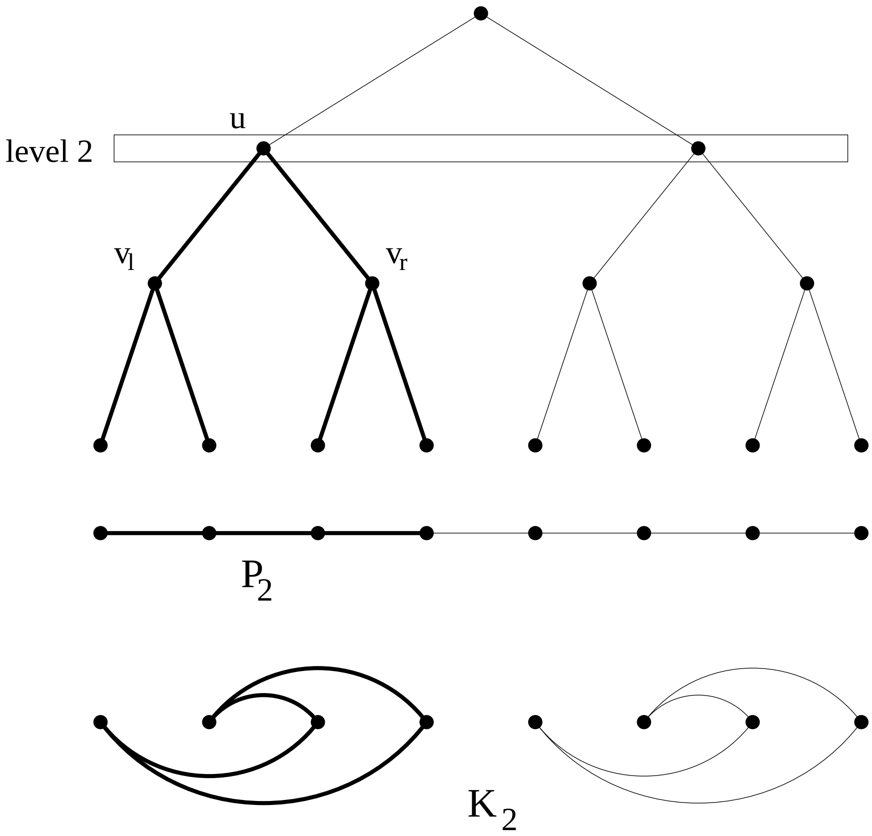

Figure 3: K_i and the Corresponding Subtrees for $i = 2$

or $2^i - 1$. Any edge of K_i must be embedded in the path between its endpoints, which must be within the leaves of a single subtree. Thus the maximum path length is the length of P_i, and the dilation d_i is less than 2^i.

Now consider the edges from K_i embedded into P_i; each one runs from a vertex in the left half of P_i to a vertex in the right half. Thus, each uses the middle edge in its embedding. K_i includes one such edge from each vertex on the left of the middle edge to each vertex on the right, for a total of $(2^{i-1})^2 = 2^{2i-2}$; the weight of each such edge is less than or equal to $\frac{1}{2^{2i-1}}$. The weight of the middle edge of P_i, which is part of A_i, is $\frac{1}{2^{k+1-i}}$. Recalling that the congestion of an edge e is the sum of the weights of the edges embedded using e divided by the weight of e, we have that c_i, the congestion of the middle edge when using index i is bounded as follows:

$$c_i = \frac{2^{2i-2}}{2^{2i-1}} \cdot 2^{k+1-i} = \frac{2^{k+1-i}}{2} = 2^{k-i}.$$

The support $\sigma(K_i, A_i)$ is bounded above by the product of the congestion times the dilation:

$$\sigma(K_i, A_i) \leq c_i d_i < 2^{k-i} \cdot 2^i = 2^k = n.$$

Note that this value is independent of i, and that it gives the bound in the lemma statement.
□

We can now prove the following theorem:

Theorem 4.3

$$\kappa(K^{-1}A) = O(n \log n).$$

Proof: This follows immediately from Lemmas 4.1 and 4.2, and from Theorem 3.5:

$$\kappa(K^{-1}A) = \frac{\lambda_{\max}(K^{-1}A)}{\lambda_{\min}(K^{-1}A)} \leq \sigma(A, T)\sigma(K, A) = O(n \log n).$$

□

4.2 Bounds for Rectangular Grids

We can think of the path graph from the preceding section as a regular grid in 1 dimension. Gremban goes on to generalize the result to regular rectangular meshes in higher dimensions. Let n be the length of a side of a grid in d dimensions; the grid will have size $N = n^d$. Then the spectral condition number of the grid is bounded above by $O(d^2 n \log n)$. Stated with respect to the size of the grid, the bound is $O(dN^{\frac{1}{d}} \log N)$. This result holds for any integer n.

We can compare the results in two dimensions with those for incomplete Cholesky factorization and modified incomplete Cholesky factorization: For incomplete Cholesky, the spectral condition number for the preconditioned rectangular grid in two dimensions is $O(n^2)$, or $O(N)$, which is the same as for the original problem. For modified incomplete Cholesky, the spectral condition number of the preconditioned system is $O(n)$, or $O(N^{\frac{1}{2}})$. (Gremban cites these results in Chapter 2, which I did not distribute.) However, spectral condition numbers alone do not define the performance of a preconditioning matrix. For example, as noted in a survey by Chan and van der Vorst, preconditioners formed by incomplete factorization can improve the distribution of eigenvalues, which speeds convergence. And, as Gremban notes, spectral condition number is only one factor in judging parallel performance. Thus Gremban also did some experiments to compare the running times of various methods.

4.3 Experimental Results

Gremban's experimental results are presented in Chapter 6, which I did not distribute. (They are also reported in Carnegie Mellon CS tech report CMU-CS-94-205, a shorter version of which appears in the Proceedings of the Ninth International Parallel Processing Symposium). However, a summary of what he found may suggest ideas for future areas of exploration.

In the experimental results, support trees were compared with Jacobi and incomplete Cholesky factorization preconditioners. The conjugate gradient methods using these are Diagonal Scaled Conjugate Gradient (DSCG) and Incomplete Cholesky Conjugate Gradient (ICCG) respectively. The experiments compared number of iterations needed for convergence and total running time until convergence of parallel implementations of these methods. The running time measurements did not include the time required to build the preconditioners.

Four sets of experiments were run: on 2D $n \times n$ rectangular meshes, on an irregular mesh corresponding to a cracked plate, on 3D $n \times n \times n$ meshes, and on $8 \times 8 \times n$ rectangular meshes. In general, the results showed that DSCG typically used the largest number of iterations. The number of iterations for STCG and ICCG usually crossed at some point, with STCG requiring fewer iterations as the problem sizes became large. This behavior was strongest for the 2D meshes, and weakest for the 3D meshes. In the 3D case, no crossing occurred for the problem sizes tested, and ICCG consistently used the fewest iterations. Gremban notes that the number of iterations required for the 3D problems is small, and speculates that this is because the ratio of problem size (volume) to diameter becomes large quickly in the 3D case. Results in this case are inconclusive.

As for running time, the benefits of fast parallel operation allowed STCG to converge in the shortest time for all example classes even for moderately large problems.

5 Parallel Calculations for Trees

Trees are easy to solve in parallel using a variant of a standard technique called *tree contraction*. Tree contraction allows the fast parallel solution of problems such as expression evaluation in cases where the graph of the data dependencies forms a tree. Usually tree contraction consists of two phases: a *rake* phase, where the leaves of the tree are processed, and a *contraction* phase, during which independent sets of nodes in long chains are processed. Since support trees do not have long chains (every subgraph is separated, which corresponds to every node having two or more children), we only need to use the rake phase to apply tree contraction to a support tree.

Gremban solves tree-structured matrices in two steps: first he forms the Cholesky factorization using the rake operation. This gives the upper and lower factors the same tree structure as the original matrix. He then uses the tree structure to do triangular solves. However, it is easier to illustrate the rake operation using Gaussian elimination, which solves the tree system directly.

Consider the following problem:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ -1 & -1 & 0 & 0 & 3 & 0 & -1 \\ 0 & 0 & -1 & -1 & 0 & 3 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

The graph for the matrix is a complete binary tree with unit edge weights and an edge from the root to the zero boundary. The numbering of the vertices is as shown in the following figure:

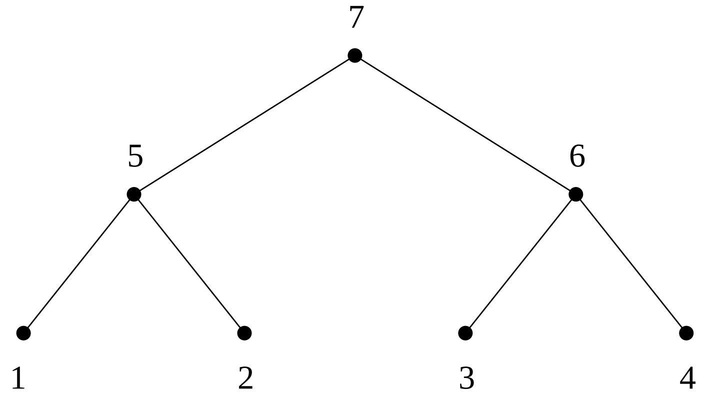

Figure 4: Numbering of Tree Vertices

Now consider what happens when we perform Gaussian elimination on all the leaves. The result is as follows:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & -1 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 + b_1 + b_2 \\ b_6 + b_3 + b_4 \\ b_7 \end{bmatrix}$$

This corresponds to a rake step; note that each vertex that is a parent of a leaf is updated independently. Since the degree of the tree is bounded, these updates involve a constant number of operations to update the right-hand side and to update the row in the matrix. Note that the remaining problem on vertices 5, 6, and 7 is a bounded-degree tree, so we can perform rake steps until we get to the root. Also note that, once the value of x_p of leaf l's parent p is known, the value of x_l can be computed with a single arithmetic operation.

This example is particularly simple, since none of the vertices other than the root has an edge to the boundary. Thus, all the pivots are 1. However, the existence of other edges to the boundary does not change the algorithm; it only changes the pivots. Updates and computations of x_i's can still be done in constant time.

Gremban points out other sources of parallelism, most notably the independence of subtree calculations. This provides an easy way to partition the problem.

A Edge Weights in the Reduced Tree

In Section 4 we claimed that the edges of K had the following property: For vertices u and v in T, let j be the number of internal vertices of T on the path between them. Then the weight of edge $(u, v) \in K$ after elimination is less than or equal to 2^{-j}. We now prove this.

Let's start by considering Gremban's argument for this, which I believe has a bug in it (all labels correspond to labels in Figure 5): We can eliminate the internal vertices of T in any order without changing the result; we will consider elimination from the root down, eliminating levels of the tree in order. The proof is inductive, with the inductive hypothesis that any edge in the partially reduced tree tree meets the weight bound specified above. The base case is the tree itself, which has unit weights.

Now consider elimination. When we eliminate vertex v, we create new edges between v's children and other neighbors of v. For example, consider the new edge between c_1 and u. It is easy to verify that eliminating v will create new matrix entries equivalent to adding an edge (u, c_1) with weight $\frac{w_{(v,u)}w_{(v,c_1)}}{d_v}$, where d_v is the diagonal entry for v (this is its generalized degree, hereafter referred to as degree). This relation holds for any such edge. How many vertices are on the path in T corresponding to such an edge? Since T is a tree, the only path from c_1 to u is a concatenation of the paths from c_1 to v, and from v to u. The number of internal vertices on this path is the sum of the number of internal vertices on each path plus 1 (for v). Note that, since we are eliminating from top down, v has two edges of weight 1 to its children, so the degree of v is greater than 2. Since the inductive hypothesis holds, and since the degree of v is greater than 2, it is easy to verify that the weights of the new edges are less than the bound. Consider our example edge and let there be j internal vertices on the path from v to u, and i internal vertices on the path from c_1 to v. Applying the inductive hypothesis and the fact that $d_v > 2$, we have the

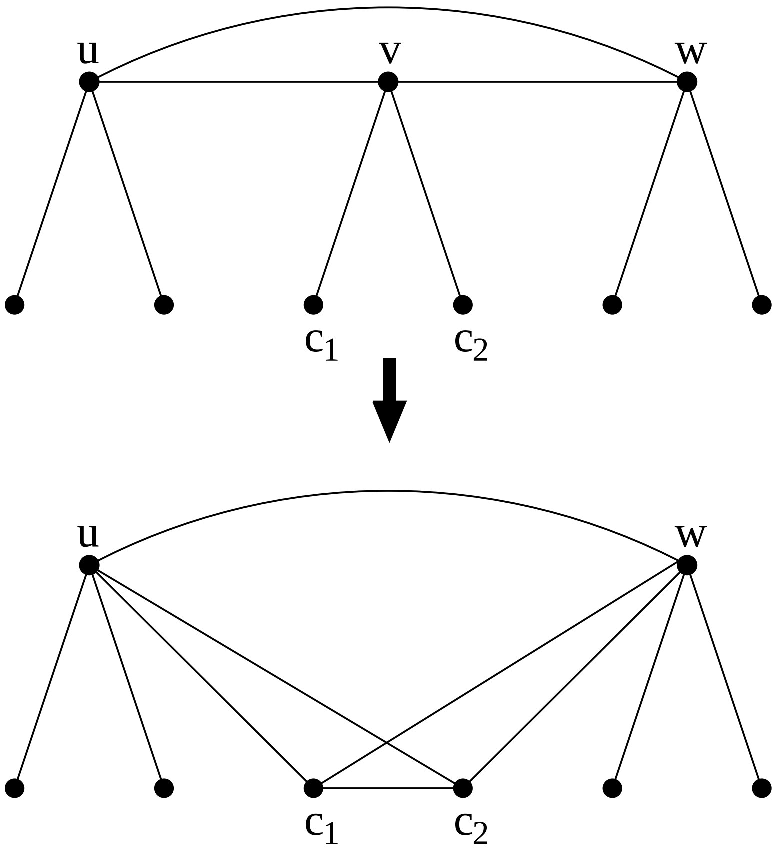

Figure 5: An Elimination Step in a Tree

following:

$$\frac{w_{(v,u)}w_{(v,c_1)}}{d_v} < \frac{2^{-j}2^{-i}}{d_v} < \frac{2^{-(j+i)}}{2} = 2^{-(j+i+1)}.$$

Since there are $j + i + 1$ internal vertices on the path from u to c_i, this gives the desired result. But...

What this argument ignores is that elimination can also increase the weight of edges that remain during the elimination. For example, edge (u, w)'s weight will be increased by $\frac{w_{(v,u)}w_{(v,w)}}{d_v}$. The inductive argument must account for these weights, which it currently does not. It may be possible to accommodate such increments by taking into account how much larger than 2 the degrees of the eliminated vertices are, but I found it easier to work from the bottom up to prove the desired property of K's edge weights. A proof follows; some details are left as exercises.

Lemma A.1 *Let T be an unweighted complete binary tree with 2^k leaves. After all internal vertices of T are eliminated, an edge (u, v) in the remaining clique on the leaves has weight $w_{(u,v)} \leq 2^{-j}$, where j is the number of internal vertices on the path in T between u and v.*

Proof: As in the argument in Lemma 4.2, we will index the levels of internal vertices from the leaves up. Leaves are at level 0, the root is at level k, and internal vertices have levels corresponding to their heights above the leaves. The argument is by induction on the internal levels, plus an additional case for the root. At the i^{th} step, the vertices at level i are eliminated. Each vertex v at level i is the root of a subtree of T; after the i^{th} step, the leaves of v's subtree plus v's parent form a clique. We will prove facts about the weights of the edges in these cliques. Figure 6 illustrates the elimination at the second induction step in a single subtree rooted at level 3.

Let v be an internal vertex at level $i < k$, let p be v's parent. As noted in previous arguments, the subtree rooted at v has a left subtree rooted at v's left child and a right subtree rooted at v's right child. Note that when v is eliminated, two types of new edges are created: those from leaves to p (dotted lines in Figure 6), and those between leaves originally in the left and right subtrees of v (dashed lines in Figure 6).

The induction hypothesis specifies the weights of new edges added or weight increments to existing edges as the non-root vertices are eliminated:

- New edges between leaves and vertices at level $i+1$ have weight $\frac{1}{2^{i+1}-1}$ and correspond to paths with i internal vertices.
- New edges between leaves have weight $\frac{1}{(2^i-1)(2^{i+1}-1)}$ and correspond to paths with $2i+1$ internal vertices. The same weight is added to any existing edges between leaves.

The base case is the elimination of vertices at level 1. The reader can check that the weights of the new edges are all $\frac{1}{3}$.

The key to proving that the induction hypothesis holds after an induction step is the degree of the eliminated vertex v. It has a single edge of weight 1 to its parent, and 2^i edges to leaves (these were originally the leaves of the subtree rooted at v). The weight of each such edge is $\frac{1}{2^i-1}$ by the induction hypothesis. Thus v's degree $d_v = 1 + \frac{2^i}{2^i-1} = \frac{2^{i+1}-1}{2^i-1}$. Recalling that the weights generated when v is eliminated are the product of two edges weights divided by d_v, we have two cases to consider. The new edges to v's parent p involve an edge of weight 1 and and edge of weight $\frac{1}{2^i-1}$; new edges between leaves (or the increment to existing such edges) involve two edges of weight $\frac{1}{2^i-1}$. It is left as an exercise to show that the product of the edge weights divided by d_v give the new weights as stated in the induction hypothesis. This completes the argument for elimination of vertices at levels less than k.

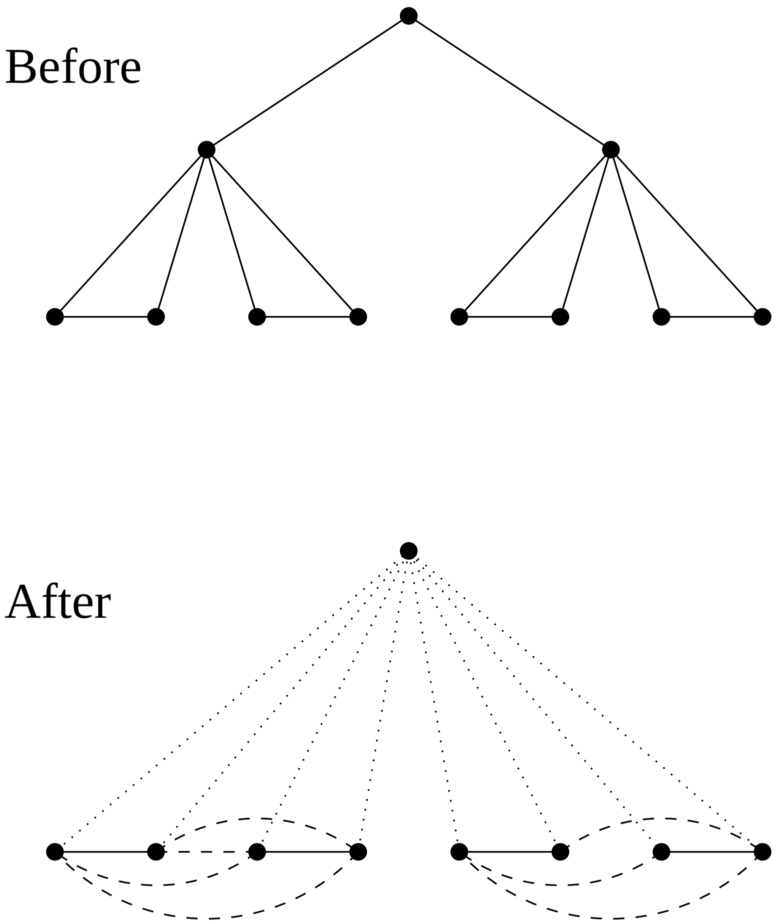

Figure 6: Elimination in a Single Subtree at Induction Step 2

The root r of the tree is at level k; the situation there is slightly different because the root has no parent. Thus, no edges to higher levels are created. Also, the degree at the root is just the product of the number of edges to the leaves times the weight of these edges. There are 2^k such edges; by the argument just completed, these edges have weight $\frac{1}{2^k-1}$. Thus $d_r = \frac{2^k}{2^k-1}$. The reader can verify that new edges between leaves formed at this level have weight $\frac{1}{2^k(2^k-1)}$; this is also the increment added to existing edges between leaves.

All that is left is to show that the original weight of an edge plus the sum of any increments it receives is bounded as per the lemma statement. Recall that a path with j internal vertices is between leaves of T that have their least common ancestor at level $i = \frac{j+1}{2}$. The corresponding edge in K is formed when the least common ancestor is eliminated. If the least common ancestor is the root r, then this level is k, and $j = 2k - 1$. As we've just shown, these edges have weight

$$\frac{1}{2^k(2^k - 1)} = \frac{1}{2^{2k} - 2^k} < \frac{1}{2^{2k-1}} = \frac{1}{2^j},$$

which obeys the bound.

If the least common ancestor is at some level $i < k$, we need to add up the initial weight plus all increments from the elimination of higher levels. The initial weight is $\frac{1}{(2^i-1)(2^{i+1}-1)}$; for each l such that $i < l < k$, it receives an increment of $\frac{1}{(2^l-1)(2^{l+1}-1)}$. The increment at level k is and additional $\frac{1}{2^k(2^k-1)}$. Noting that $j = 2i - 1$ for this case, we want to show that

$$\left(\sum_{l=i}^{k-1} \frac{1}{(2^l - 1)(2^{l+1} - 1)}\right) + \frac{1}{2^k(2^k - 1)} < \frac{1}{2^{(2i-1)}}.$$

This is straightforward and is left to the reader (Hint: note that $2^{-(2i-1)} = 2 \cdot 4^{-i}$, that $\frac{1}{(2^l-1)(2^{l+1}-1)} < 4^{-l}$, and $\frac{1}{2^k(2^k-1)} < 2 \cdot 4^{-k}$). This completes the proof of the lemma.
□