Performance Evaluation of a New Parallel Preconditioner

Keith D. Gremban   Gary L. Miller   Marco Zagha

School of Computer Science
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh PA 15213

Abstract

The linear systems associated with large, sparse, symmetric, positive definite matrices are often solved iteratively using the preconditioned conjugate gradient method. We have developed a new class of preconditioners, support tree preconditioners*, that are based on the connectivity of the graphs corresponding to the matrices and are well-structured for parallel implementation. In this paper, we evaluate the performance of support tree preconditioners by comparing them against two common types of preconditioners: diagonal scaling, and incomplete Cholesky. Support tree preconditioners require less overall storage and less work per iteration than incomplete Cholesky preconditioners. In terms of total execution time, support tree preconditioners outperform both diagonal scaling and incomplete Cholesky preconditioners.*

1 Introduction

The demand for increasingly accurate numerical simulation leads to increasingly large systems of linear equations. Systems with 100,000 equations are not unusual, and systems with more than 1,000,000 variables are desirable. We are interested in the subclass of linear systems with coefficient matrices that are sparse, symmetric, positive definite, diagonally dominant, and have non-positive off-diagonals. These matrices often arise from the discretization of second-order, self-adjoint, elliptic partial differential equations.

The method of conjugate gradients (CG) is a popular iterative method. The performance of CG can be improved by the use of a preconditioner, yielding the method of preconditioned conjugate gradients (PCG).

The best performance is achieved by multilevel preconditioners; some can achieve nearly optimal convergence rates and can be effectively parallelized [5][8]. However, they require *a priori* knowledge about the differential equation or the discretization process, which is often unavailable.

Diagonal scaling and incomplete Cholesky [10] preconditioners depend only on the coefficient matrix; we term these *a posteriori* preconditioners, since they can be constructed after the linear system is formulated. *A posteriori* preconditioners are the most general.

Diagonal scaling can be effectively parallelized, but yields little improvement in the convergence rate. The incomplete Cholesky preconditioners are effective at accelerating convergence, but require more computations per iteration and are more difficult to parallelize in general.

Therefore, there exists a need for effective, parallelizable, *a posteriori* preconditioners. The support tree preconditioners, to be introduced in the next section, are a step towards fulfilling this need.

In this paper, we evaluate the performance of the PCG method using support trees (STCG) by comparison with the performance using diagonal scaling (DSCG) and incomplete Cholesky preconditioning (ICCG). In all cases considered, we found that STCG yielded convergence rates competitive with, or superior to ICCG, and processing times superior to both ICCG and DSCG.

2 Support Trees

Consider linear systems of the form $Ax = b$, where the coefficient matrix A is a symmetric, diagonally dominant M-matrix.[1] We call these matrices *Laplacian matrices,* or *Laplacians.* A Laplacian A corresponds to an undirected graph $G = G(A)$ with weighted edges: every row/column of A corresponds to a node of G; if $a_{ij} = a_{ji} \neq 0$, then nodes i and j are connected with an edge of weight $|a_{ij}|$; if the extra diagonal weight $d_i \neq 0$, where $d_i = a_{ii} - \sum_{i \neq j} |a_{ij}|$, then node i has a self-loop of weight d_i.

Let A be a Laplacian matrix of order n, and let $G = G(A)$ be the corresponding graph. Let S_0 be an edge separator of G; that partitions G into two disconnected subgraphs G_0 and

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597, in part by NSF Grant CCR-9016641, and in part by the Pittsburgh Supercomputing Center (Grant ASC890018P).

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government.

1. A is an M-matrix if $A_{ij} \leq 0$ for $i \neq j$, A is nonsingular, and $A^{-1} \geq 0$ [10]

G_1 such that $|G_i| \geq \frac{n}{3}$. Now, continue separating recursively until only singleton sets of nodes remain. Construct the *separator tree* by introducing a node for each edge separator, and connecting each node to its unique parent (if any), and its children (if any). Figure 1 illustrates a simple graph and a separator tree for the graph.

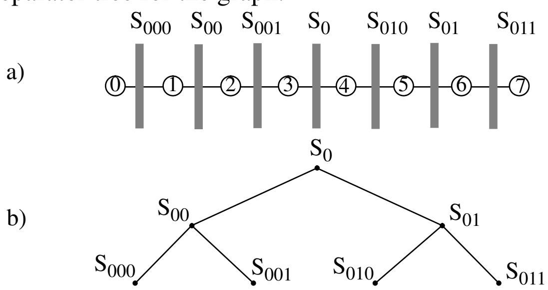

Figure 1: ***A Graph and Separator Tree.***
a) A simple graph with separators shown.
b) Separator tree corresponding to separators in a

Each node S of the separator tree defines a subset R of the nodes in G. Let $w(R)$ denote the total weight of the frontier of R, which is the set of edges in G that connect R and $\overline{R}$. Weight the edge in the separator tree connecting S to its parent by $w(R)$. Connect each leaf node in the separator tree to the singleton nodes of G, weighting the edges by the total weight of the edges incident to the node. Denote the resulting tree by H. H has $\log n$ depth, $2n$-1 nodes, and n leaves. We call H a *support tree* for G. Figure 2 illustrates a support tree for the graph and separator tree of Figure 1.

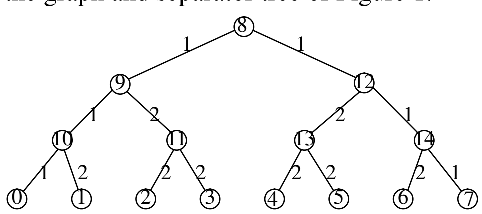

Figure 2: ***Support Tree.*** *This support tree was constructed using the graph and separator tree of Figure* ***2.***

The intuition behind support trees is the idea of maintaining the volume of communication in a graph, while reducing the distance required for the communication. Solving a system of linear equations defined over a graph using an iterative method is like a *mixing* process. A matrix-vector multiply is equivalent to mixing the data at one node with the data from its neighbors. Since a matrix-vector multiply only lets nodes communicate with their immediate neighbors, mixing cannot be complete until information from distant nodes has been obtained. Thus, the convergence is related to the diameter of the graph. For a planar graph with n nodes, the diameter is $O(\sqrt{n})$, while the diameter of a support tree for that graph is only $O(\log n)$. Therefore, mixing (convergence) will occur more rapidly with the support tree. The method of construction of the support tree ensures that the mixing that occurs in the support tree is similar to the mixing that occurs in the original graph.

It should be noted that construction of a support tree can be easily parallelized, since the partitioning processes applied to each subgraph are independent.

Support trees depend upon the separators used to construct them. Even with a given partitioning method, support trees may take different forms. It is clearly possible to partition two or more times at a single level, yielding support trees that are quadtrees, oct-trees, and so on. In practice, we have make the degree of the support tree equal to the dimension of the space in which the graph is embedded. Figure 3 illustrates a quadtree support tree for a 2D regular mesh.

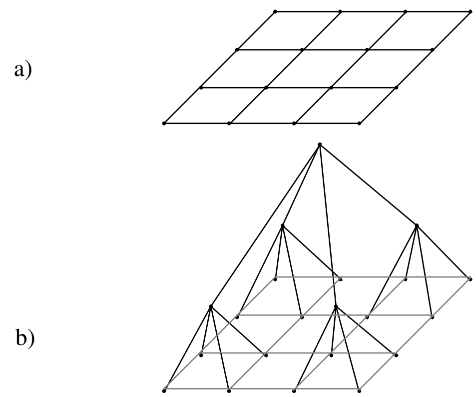

Figure 3: ***2D Regular Mesh and Support Tree.***
a) A 2D regular 4-connected mesh.
b) A quadtree support tree for the mesh in a).

Let H be the support tree for G. Let B be the Laplacian matrix corresponding to H. We would like to use B as a preconditioner for A, but B is of order $2n$-1, and A is of order n. In another paper [6], we describe the theory proving that B can be used as a preconditioner for A. Here, we present an overview of the theory in order to gain some intuition.

Suppose that H is a binary tree with n leaves and n-1 internal nodes. Assume that the leaves are numbered 1 through n. Then B, the matrix corresponding to H, has the form:

$$B = \begin{bmatrix} D & R \\ R^t & S \end{bmatrix} \qquad (1)$$

where D is nxn and diagonal.

Many matrix operations correspond to graph operations. In particular, Gaussian elimination corresponds to a graph operation we call node reduction. A single step of symmetric Gaussian elimination applied to zero out row/column k of a Laplacian M corresponding to a graph G is equivalent to deleting all the edges in G incident to node k and adding edges between all the (former) neighbors of node k. Pivoting in a matrix is equivalent to renumbering the nodes in the graph. These facts yield two particularly useful results:

- Applying Gaussian elimination to a tree from the root down, stopping at the leaves, results in a complete graph on the leaves.
- Gaussian elimination applied to the leaves of a tree produces no fill. Therefore, the Laplacian of a tree can be ordered such that its LU factoriza-

tion has zero fill.

Applying Gaussian elimination to B from equation (1) in the order from root to leaves (i.e., from row $2n$-1 up to row n+1), stopping at the leaves, yields a matrix C of the form:

$$C = \begin{bmatrix} K & 0 \\ 0 & E \end{bmatrix} \tag{2}$$

where E is diagonal of order n-1, and K is dense of order n.

In [6], we show the following:

- K is an effective preconditioner for A;
- If $K \cdot z = r$, then we also have $B \cdot \begin{bmatrix} z \\ w \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$.

That is, K is an effective preconditioner, but we can obtain the same results using a larger, but sparser tree-structured system, by simply discarding the unneeded additional vector elements (w, above). We shall show in the next section that the tree-structured system is both very sparse and computationally efficient, leading to highly parallel code.

3 Implementation of STCG

The key step in any PCG method is solving the preconditioning system $Bz = r$. We solve this system by computing the Cholesky factors of B and solving each of the resultant triangular systems. However, since B represents a tree, the factors are sparse and can be evaluated very efficiently.

Every tree has a zero-fill, or perfect ordering, that yields a Cholesky factorization with no fill. Prior to calling our subroutine that implements STCG, we find a perfect ordering and permute the equations accordingly.

Let $B = C \cdot D \cdot C^t$ be the root-free Cholesky factorization of B. Then C represents a tree with all edges directed from the leaves towards the root, C^t represents the same tree with the edges reversed, and D represents a scaling of the node values. Thus, solving a tree-structured linear system involves propagating information up the tree, scaling the values at all nodes, then propagating information back down the tree.

The structure of a tree permits efficient parallel evaluation. The fact that leaves are not interconnected means that leaves can be evaluated independently. Hence, all the nodes at a single level can be evaluated in parallel, so that a complete binary tree with n leaves requires only $2 \cdot \lceil \log n \rceil$ parallel steps. Prior to calling the STCG subroutine, we determine an optimal order in which to evaluate nodes. We call this ordering *rake-order*, since leaves are "raked" off the tree at each step. Figure 4 illustrates leaf raking.

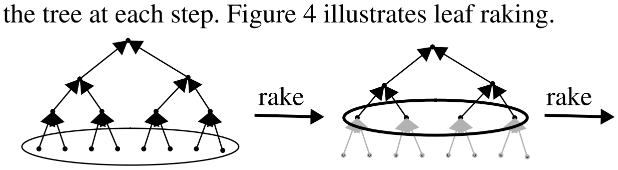

Figure 4: ***Leaf Raking.***

Additional parallelism of STCG is possible due to the tree structure — separate subtrees may be evaluated in parallel on multiple vector processors.

The performance of STCG is dominated by two operations: sparse matrix multiplication and preconditioning. In fact, the raking operation performed at each level is essentially a sparse matrix multiplication. Thus, the bulk of the computation in STCG can be implemented with a single general-purpose sparse matrix multiplication subroutine. On the Cray C-90, we use an algorithm called SEGMV, which accomodates arbitrary row sizes using "segmented scan" operations [2]. Compared to other methods (such as Ellpack/Itpack and Jagged Diagonal), SEGMV performance is comparable for structured matrices, and superior for most irregular matrices. Thus our STCG implementation performs well on both regular and irregular meshes.

Finally, we evaluate the resource requirements of STCG, and compare them with those of DSCG and ICCG. We assume that the entire diagonal is stored for DSCG. For ICCG and STCG, we assume that the preconditioner B has been factored as $B = CDC^t$. Table 1 and Table 2 give the resource requirements for 2D square and 3D cubic meshes, respectively; lower order terms have been ignored. The support trees are quadtrees in 2D, and octrees in 3D.

Table 1: ***Preconditioner Resource Requirements for an nxn Mesh.***

2D (nxn)	DSCG	ICCG	STCG
storage	n^2	$5n^2$	$4n^2$
+	0	$3n^2$	$2n^2$
*	0	$4n^2$	$(8/3)n^2$
/	n^2	n^2	$(4/3)n^2$

Table 2: ***Preconditioner Resource Requirements for an nxnxn Mesh.***

3D(nxnxn)	DSCG	ICCG	STCG
storage	n^3	$7n^3$	$(24/7)n^3$
+	0	$5n^3$	$2n^3$
*	0	$6n^3$	$(16/7)n^3$
/	n^3	n^3	$(8/7)n^3$

Note that DSCG is the cheapest preconditioner to use. In 2D, STCG is slightly better than ICCG, but the difference is larger in 3D. The resource requirements for ICCG increase with increasing graph connectivity, while those for STCG depend only upon the form of the support tree.

4 Empirical Evaluation of STCG

Greenbaum et al.[5] and Heroux et al.[9] both conducted empirical evaluations of preconditioner performance. Both studies found that ICCG significantly improved the convergence rate. However, both studies also found that the advantages of ICCG with respect to total execution time diminish or vanish on vector and parallel machines. In this section, we will confirm the results of their investigations with respect to DSCG and ICCG, but will also demonstrate that STCG is superior to DSCG and ICCG for solving large problems, and is easily and effectively parallelized.

We limit ourselves to only comparing DSCG, ICCG, and STCG. We made no attempt to go beyond the obvious optimizations of ICCG. Numerous other authors have reported on optimizations of ICCG (see, for example [3], [4], [11]). We studied their results and concluded that a general, optimal implementation of ICCG would require twice the time per iteration of DSCG. Accordingly, we report these extrapolated optimal values as ICCG-OPT.

All results were obtained using a single processor on the Cray C-90 at the Pittsburgh Supercomputing Center.

4.1 Two-dimensional regular meshes

In their work, Greenbaum et al. considered the time-independent version of the diffusion equation defined on the unit square with Dirichlet boundary conditions:

$$\nabla \cdot \rho(x, y) \nabla u(x, y) = f(x, y)$$
$$(x, y) \in (0, 1) \times (0, 1)$$
$$u(0, y) = u(1, y) = u(x, 0) = u(x, 1) = 0$$

For our experiments on regular meshes, we used $\rho(x,y) = 1.0$, which reduces the diffusion equation to Poisson's equation. We discretized the equation using the 5-point finite difference operator, and varied the size of the *n*x*n* mesh using *n* ranging from 8 to 128 in powers of 2.

For our initial experiments, we used the same forcing function as Greenbaum et al.:

$$f(x, y) = -2x(1-x) - 2y(1-y)$$

We used as our stopping criterion the condition reported to be superior by Arioli et al. [1]:

$$\omega_2 = \frac{\| b - \mathrm{A} \cdot \hat{x} \|_\infty}{\| \mathrm{A} \|_\infty \cdot \| \hat{x} \|_1 + \| b \|_\infty} \qquad (3)$$

We halted when $\omega_2 \le 1.0 \times 10^{-10}$. The starting vector was the zero vector. The results are reported in Figure 5.

In a second set of experiments, we selected a more difficult forcing function, an impulse function b, with $b_0 = 1.0$, $b_i = 0.0$ for $i \neq 0$. For each mesh, node 0 was at the lower left corner.

Again, we halted when $\omega_2 \le 1.0 \times 10^{-10}$ and used the zero vector as the starting vector. The results for this forcing function are reported in Figure 6.

4.2 Two-dimensional irregular meshes

We also investigated the relative performance of STCG on irregular meshes. We had available to us a nested sequence of meshes developed for the computation of stress on a cracked plate. There are 11 meshes in all, with 10×2^i nodes in each mesh, i = 0,1,2,...,10. Figure 7 illustrates two of the meshes. The crack in the plate runs from the center to the left side, parallel to the x-axis.

The crack data consisted of pattern-only information. We constructed non-singular coefficient matrices by augmenting the Laplacians of the meshes with additional diagonal weights at each corner node.

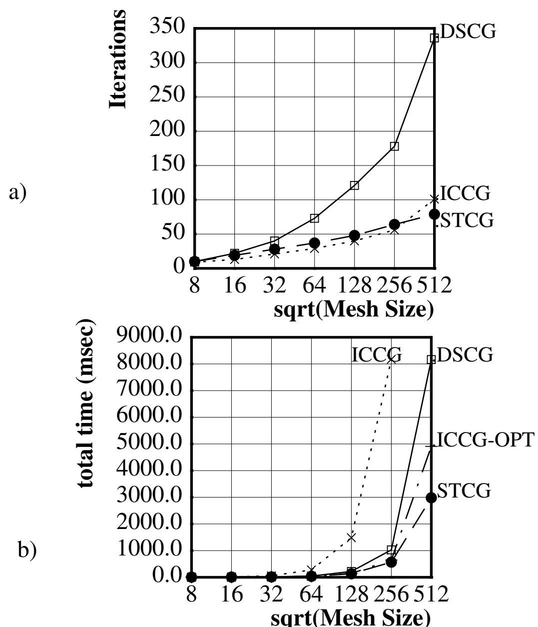

Figure 5: ***Results for 2D Regular Meshes, Smooth Input.***
a) number of iterations for convergence.
b) total time for iterative process on a Cray C-90.

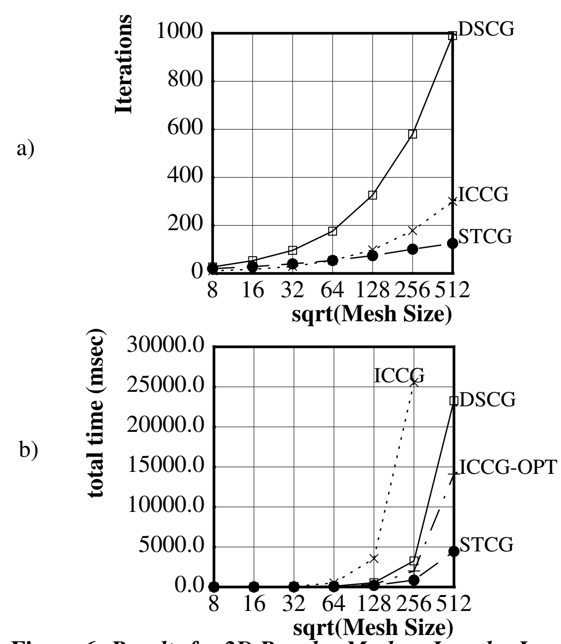

Figure 6: ***Results for 2D Regular Meshes, Impulse Input.***
a) number of iterations for convergence.
b) total time for iterative process on a Cray C-90.

For the forcing function, we used an impulse function with $b_0 = 1.0$, $b_i = 0.0$ for $i \neq 0$; node 0 is at the bottom left corner of each mesh. We halted when $\omega_2 \le 1.0 \times 10^{-10}$. The starting vector was the zero vector. The results are reported below in Figure 8.

5 Summary and Discussion

In this paper, we compared the performance of a new variant of preconditioned conjugate gradient, STCG, against the

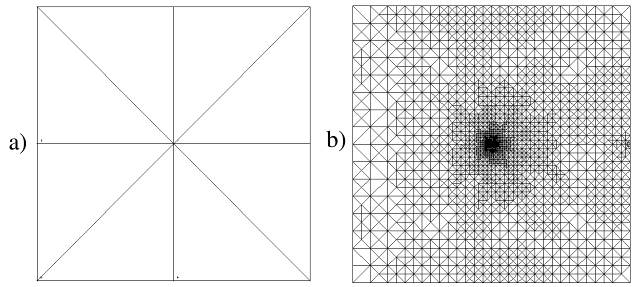

Figure 7: Crack Meshes.
a) crack00 with 10 nodes. b) crack08 with 2560 nodes

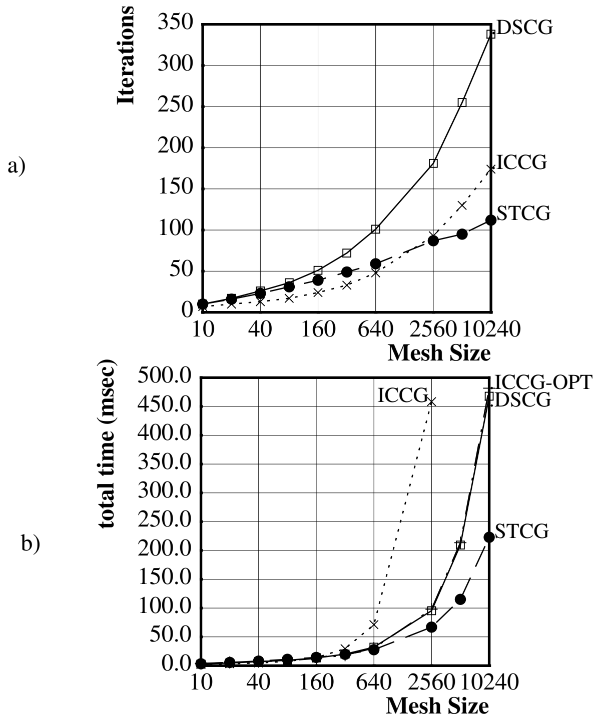

Figure 8: ***Results for 2D Irregular Meshes, Impulse Input.***
a) number of iterations for convergence.
b) total time for iterative process on a Cray C-90.

performance of two well-known variants, DSCG and ICCG. We have demonstrated that on both irregular and regular meshes:

- STCG requires less overall storage and less work per iteration than ICCG.
- in terms of iterations to converge, STCG meets or exceeds the performance of ICCG, which in turn, outperforms DSCG.
- in terms of execution time, STCG far outperforms both ICCG and DSCG on a vector processor.

At present, support tree preconditioners can only be applied to Laplacian matrices. One of our goals is to extend the support tree methodology to larger classes of matrices.

An expanded version of this paper is available as a technical report [7].

6 Acknowledgments

The authors would like to thank Guy Blelloch, Omar Ghattas, and Mike Heroux for many useful conversations. We would also like to thank Omar Ghattas for the crack meshes.

7 References

[1] M. Arioli, I. Duff, D. Ruiz, *Stopping criteria for iterative solvers*. **SIAM J. Matrix Anal. Appl.** 13(1):138-144, 1992.

[2] G. E. Blelloch, M. A. Heroux, and M. Zagha, *Segmented operations for sparse matrix computation on vector multiprocessors.* CMU-CS-93-173, School of Computer Science, Carnegie Mellon University, 1993.

[3] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, **Solving Linear Systems on Vector and Shared Memory Computers.** SIAM, 1991.

[4] I. S. Duff, and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*. **BIT** 29:635-657, 1989.

[5] A. Greenbaum, C. Li, and H. Z. Chao, *Comparison of linear system solvers applied to diffusion-type finite element equations*. **Numer. Math.** 56:529-546, 1989.

[6] K. D. Gremban, and G. L. Miller, *Towards the Application of Graph Theory to Finding Parallel Preconditioners for Sparse Symmetric Linear Systems.* Technical Report, Computer Science Department, Carnegie Mellon University, in preparation.

[7] K. D. Gremban, G. L. Miller, and M. Zagha, *Performance Evaluation of a New Parallel Preconditioner.* CMU-CS-94-205, School of Computer Science, Carnegie Mellon University, 1994.

[8] X. -Z. Guo, *Multilevel Preconditioners: Analysis, performance enhancements, and parallel algorithms.* CS-TR-2903, Department of Mathematics, University of Maryland, 1992.

[9] M. A. Heroux, P. Vu, and C. Yang, *A parallel preconditioned conjugate gradient package for solving sparse linear systems on a Cray Y-MP.* **Appl. Num. Math.** 8:93-115, 1991.

[10] J. A. Meijerink, and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix.* **Math. Comp**. 31:148-162, 1977.

[11] H. A. van der Vorst, *ICCG and related methods for 3D problems on vector computers.* **Comp. Physics Comm.** 53:223-235, 1989.