HIGHLY PARALLEL PRECONDITIONERS FOR GENERAL SPARSE MATRICES [*]

Youcef Saad [†]

Abstract

The degree of parallelism in the preconditioned Krylov subspace method using standard preconditioners is limited and can lead to poor performance on massively parallel computers. In this paper we examine this problem and consider a number of alternatives based both on multi-coloring ideas and polynomial preconditioning. The emphasis is on methods that deal specifically with general unstructured sparse matrices such as those arising from finite element methods on unstructured grids. It is argued that multi-coloring can be combined with multiple-step relaxation preconditioners to achieve a good level of parallelism while keeping the rates of convergence to good levels. We also exploit the idea of multi-coloring and independent set orderings to introduce a multi-elimination incomplete LU factorization named ILUM, which is related to multifrontal elimination. The main goal of the paper is to discuss some of the prevailing ideas and to compare them on a few test problems.

Keywords Large linear systems; Krylov subspace methods; iterative methods; preconditioned conjugate gradient; Multi-coloring; incomplete LU preconditioning.

[*]This research was supported in part by the Army Research Office under contract DAAL03-89-C-0038 and in part by NIST under grant number 60NANB2D1272 .

[†]University of Minnesota, Computer Science Department, 4-192 EE/CSci Building, 200 Union Street S.E., Minneapolis, MN 55455

1 Introduction

Direct methods for solving linear systems are often preferred to iterative methods in real applications. For two-dimensional problems, direct solvers can be quite effective and, perhaps more importantly, they are very robust. On the other hand, the memory and computational requirements for 3-dimensional problems or 2-dimensional PDE's with many degrees of freedom per grid point, may cause serious challenges to the most efficient direct solvers available today. In addition, iterative methods are currently gaining ground because they are much easier than direct methods to implement efficiently on high-performance computers. Typical sparse direct solvers have taken decades to reach the stage of their current level of efficiency [21]. Given the complexity of these computer codes, it will take a substantial time before we reach a comparable level of efficiency on new architectures. In contrast porting iterative methods on massively parallel computers, will only require to develop a good implementation of the low level primitives, such as matrix-vector multiplications, inner products, and vector combinations, and to select a preconditioner that achieves a good compromise between the degree of parallelism and the convergence rate.

Preconditioners that allow a large degree of parallelism such as, in the simplest case, diagonal scaling, may necessitate a much larger number of iterations to converge when compared with their sequential counterparts, e.g., the standard Incomplete LU factorization (ILU). For example, a rather popular technique is to reorder the equations by coloring the unknowns in such a way that no two unknowns of the same color are related by an equation. In the simplest case of the 5-point matrix arising from the centered difference discretization of the Laplacean in 2 or 3 dimensional spaces, only two colors are needed, commonly referred to as red and black. If the unknowns of the same color are numbered consecutively, then a large degree of parallelism is available in the preconditioning phase. The drawback of this approach is that if an ILU type preconditioned Krylov subspace method is used then the number of iterations may increase substantially, often defeating the benefits gained from the higher degree of parallelism. Although it was observed [12] that there are orderings that have precisely the opposite effect these are not too well understood, and it is unlikely that any specific rules can be derived for the general case where the matrix arises from the discretization of coupled Partial Differential Equations on an unstructured grid as is typically the case in Computational Fluid Dynamics (CFD) for example.

Our experiments show that much can be done in recovering the good convergence properties by increasing the accuracy of the highly parallel preconditioner e.g., by performing several steps in SOR/SSOR or by using higher level of fill-in in the ILU factorization. For example, a k-step SSOR preconditioner applied to the Red-Black ordered matrix can achieve an excellent parallel performance. This leads us to explore the possibilities of exploiting general purpose multi-coloring techniques for general sparse matrices.

We will start by giving an overview of the standard techniques used in implementing preconditioned Krylov subspace methods. Then we will discuss some newer ideas and issues related to multi-coloring.

In order to illustrate the techniques, we found it preferable to present our numerical experiments along the way rather than at the end of the paper. Throughout the paper we will use the following three model problems.

Problem 1. We consider the Partial Differential Equation:

$$-\Delta u + \gamma \left(\frac{\partial e^{xy} u}{\partial x} + \frac{\partial e^{-xy} u}{\partial y} \right) + \alpha u = f \ ,$$

with Dirichlet Boundary Conditions and $\gamma = 10, \alpha = -60$; discretized with centered differences on a $27 \times 27 \times 27$ grid. This leads to a linear system with $N = 25^3 = 15,625$ unknowns.

Problem 2. We consider the same problem as above but with the parameters $\gamma = 60, \alpha = -10$.

Problem 3. We consider a linear system made up from matrix Sherman-3 of the Harwell-Boeing collection. This matrix is of dimension $N = 5,005$ and has $nz = 20,033$ nonzero elements. It arises in a IMPES (IMplicit Pressure, Explicit Saturation) simulation of a black-oil reservoir on a $35 \times 11 \times 13$ grid.

For all the above matrices, we construct the right hand side artificially to be of the form $b = Ae$, where the solution e is the vector whose components are all ones. The initial guess is always a random vector. Although the problems are associated with regular 3-dimensional grids, this was not exploited, i.e., the matrices are considered as general unstructured sparse. These three problems are by no means easy to solve by iterative methods but they are not representative of the most difficult ones that arise in some industrial applications. We also point out that all the methods tested start with the same given data and that no additional information on the problem is exploited by any of the methods. For example, the polynomial preconditioning techniques are not supplied with spectral information or information concerning the location of the eigenvalues. Concerning the stopping criterion, all iterative solution methods are stopped as soon as the residual norm is reduced by a factor of $\epsilon = 10^{-7}$. A double star in the tables means that the method under consideration has failed to converge within the maximum number of iterations allowed. All experiments have been performed on a Cray-2 in single precision (64bits arithmetic).

2 Preconditioned Krylov Subspace Methods

We consider the linear system

$$Ax = b, \tag{1}$$

where A is a large sparse nonsymmetric real matrix of size N. Several conjugate gradient-type iterative techniques based on projection processes on so-called Krylov subspaces have been proposed in recent years to solve such systems. A small list of references in this area is [8, 9, 13, 19, 18, 24, 22, 28, 29, 31, 50, 36, 41, 42]. GMRES is a technique introduced in [42] for solving general large sparse nonsymmetric linear systems of equations by minimizing the 2-norm of the residual vector $b - Ax$ over x in the Krylov subspace

$$K_m = \text{Span}\{r_0, Ar_0, \ldots, A^{m-1}r_0\},$$

where r_0 is the initial residual vector $b - Ax_0$.

The idea of preconditioning is simply to transform the above system, e.g., by multiplying it through with a certain matrix M^{-1}, into one that will be easier to solve by a Krylov subspace method. For example, when the preconditioner M is applied to the right, we will be solving instead of (1), the preconditioned linear system

$$(AM^{-1})(Mx) = b. \tag{2}$$

In some applications it can be important to allow the preconditioner to fluctuate from step to step in the inner Krylov subspace iteration method. For later reference we now describe a variant of the GMRES algorithm based on this approach which was developed in [39]. The last step in the GMRES algorithm for solving (2) forms the solution as a linear combination of the preconditioned vectors $z_i = M^{-1}v_i, i = 1, \ldots, m$, where v_i are the Arnoldi vectors and M is the preconditioning. Because these vectors are all obtained by applying the same preconditioning matrix M^{-1} to the v's, we need not save them. We only need to apply M^{-1} to the linear combination of the $v's$, i.e., to $V_m y_m$. If we allowed the preconditioner to vary at every step, i.e., if z_j is now defined by

$$z_j = M_j^{-1} v_j$$

we may think of saving these vectors to use them in up-dating x_m. This observation yields the following 'flexible' preconditioning GMRES algorithm.

ALGORITHM **2.1 FGMRES: Flexible variant of preconditioned GMRES**

```
1. Start: Choose $x_0$ and a dimension $m$ of the Krylov subspaces. Define an $(m+1) \times m$
   matrix $\bar{H}_m$ and initialize all its entries $h_{i,j}$ to zero.
2. Arnoldi process:
   (a) Compute $r_0 = b - Ax_0$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$.
   (b) For $j = 1, ..., m$ do
       • Compute $z_j := M_j^{-1} v_j$
       • Compute $w := Az_j$
       • For $i = 1, \ldots, j$, do   $\begin{cases} h_{i,j} := (w, v_i) \\ w := w - h_{i,j} v_i \end{cases}$
       • Compute $h_{j+1,j} = \|w\|_2$ and $v_{j+1} = w/h_{j+1,j}$.
   (c) Define $Z_m := [z_1, ...., z_m]$.
3. Form the approximate solution: Compute $x_m = x_0 + Z_m y_m$ where
   $y_m = \mathrm{argmin}_y \|\beta e_1 - \bar{H}_m y\|_2$ and $e_1 = [1, 0, \ldots, 0]^T$.
4. Restart: If satisfied stop, else set $x_0 \leftarrow x_m$ and goto 2.
```

For additional details, see reference [39].

2.1 Standard preconditioners

Apart from the trivial preconditioners such as Jacobi preconditioning (i.e., diagonal scaling), the most popular sequential preconditioners for general sparse linear systems are variants of the following three approaches.

1. Incomplete LU factorization without fill-in (ILU(0)).
2. Increased fill-in Incomplete LU (ILU) factorizations. Two distinct approaches have been used in the past. The first, level-of-fill approach, is based on the pattern only [30, 51]. The second is based on numerical values and threshold strategies [40, 57].
3. Relaxation type preconditioners (SOR, SSOR).

We now briefly discuss these three approaches in turn.

ILU(0). The Incomplete LU factorization without fill-in [30] is one of the simplest and most popular techniques in sequential machines. The principle is to find a pair of matrices L and U, where L is unit lower triangular and U upper triangular such that $L+U$ has the same pattern as A and $(LU)_{ij} = a_{ij}$ for each pair (i,j) that belong to the nonzero structure of A, i.e., for every i,j so that $a_{i,j} \neq 0$. In fact such a factorization is not uniquely defined. The usual way of defining it properly is to give a constructive definition. For example: *ILU(0) is obtained by performing the standard Gaussian Elimination and replacing by zero any fill-in element during the process.* Clearly, the zeros that are introduced need not be stored.

ILUT(p,τ). This incomplete factorization technique is based on a two-parameter strategy for dropping elements [40]. The algorithm is based on the usual (i,k,j) (row-oriented) version of Gaussian elimination, i.,e., the i rows of L and U are determined simultaneously at step i. At each step the algorithm will perform the linear combination of the current version of the row (called *row* in the algorithm) with the jth row of U if *row*(k) is nonzero. During this combination small elements, namely those smaller that the tolerance times the norm of the original norm are dropped. ILUT(p,τ) is simply a sparse implementation of the following algorithm.

Algorithm **2.2 ILUT(p,τ)**

```
do i=2, n
   row(1:n) = a(i,1:n)
   nrm := || row(1:n) ||
   do  for (k=1,i-1 and where row(k) is nonzero)
       row(k) := row(k) / a(k,k)
       if (row(k) .lt. tau * nrm) then row(k) := 0
        if (|row(k)| * || u(k,k+1:n)|| .gt.  tau*nrm) then
           row(k+1:n)=row(k+1:n)-row(k)*u(k,k+1:n)
        endif
        do j=k+1,n
           if (abs(row(j)) .lt. tau*nrm) row(j) := 0
        enddo
   enddo
   l(i,1:i-1) = { p largest elements in row(1:i-1) }
   u(i,i:n) = diagonal element + {p largest elements in row(i+1:n)}
enddo
```

The scalar τ can be viewed as a parameter enabling a filtering mechanism to reduce the overall execution time and the experiments do confirm that it is a very important factor in reducing computational costs. Note that a slight variant of the above algorithm is to allow p elements on the i-th row of L *in addition* to the number of nonzero elements originally in the i-th row of L. Similarly for U.

SOR/ SSOR (k). A class of preconditioners that will be emphasized in this paper is one based on relaxation techniques. The best known of these is the SSOR preconditioning matrix defined by

$$M_{SSOR}(A) = (D - \omega E)D^{-1}(D - \omega F)$$

in which $-E$ is the strict lower part of A, $-F$ its strict upper part, and D its diagonal.

In the context of preconditioning, it is common to simply take $\omega = 1$ as the gains from selecting an optimal ω are typically small. Note also that an optimal ω for the SOR iteration is unlikely to be the optimal ω for the combination SOR/GMRES. However, it is clear that one can use different values of ω at each step of FGMRES and this can open up the possibility of using heuristics to determine the best ω dynamically, by simply monitoring convergence.

More generally, for any iteration of the form

$$u_{k+1} = M^{-1}(Nu_k + b) \ , \quad A = M - N \tag{3}$$

we can define a preconditioning matrix associated with applying $k + 1$ steps of this iterative process by noticing that

$$u_{k+1} = u_0 + \left[\sum_{j=0}^{k}(M^{-1}N)^j\right] M^{-1}[b - Au_0] \tag{4}$$

which means that $k+1$ steps of (3) amount to approximating the exact solution $x = u_0 + A^{-1}[b - Au_0]$ by (4). Therefore the corresponding preconditioning matrix is

$$M_k = \left[\sum_{j=0}^{k}(M^{-1}N)^j\right] M^{-1} \tag{5}$$

We also note that M_k is nothing but the truncated Neuman series applied to

$$[M(I - M^{-1}N)]^{-1} = (I - M^{-1}N)]^{-1}M^{-1}$$

The preconditioning matrix (5) will be referred to as the k-step preconditioner associated with the splitting $A = M - N$. Adams [4, 1] studied such preconditioners in the context of the conjugate gradient method and showed in particular that for the SSOR splitting, the preconditioning matrix is positive definite under certain conditions. Clearly, the same techniques can be applied to nonsymmetric matrices as well, although the theory is not as well understood. An important observation here is that in the non-Hermitian case, there is little attraction in using SSOR as opposed to the simpler SOR. In fact we found that for our model problems SOR(k) was generally superior to SSOR(k). We point out that the preconditioned matrix which is given by

$$M_kA = \left[\sum_{j=0}^{k}(M^{-1}N)^j\right] M^{-1}(M - N) = I - (M^{-1}N)^{k+1} \tag{6}$$

will typically have eigenvalues with much better separation for larger values of k than for smaller k, leading to fewer outer iterations to converge. However, the cost of each inner iteration is higher.

2.2 Standard parallel implementations

Looking at a typical conjugate gradient type algorithm we observe that the main operations are the following.

1. Setting up the preconditioner;
2. Matrix vector multiplications;
3. BLAS1 type operations: vector updates and dot products;
4. Preconditioning operations.

In the above list potential difficulties may arise in setting-up the preconditioner (1) and in the solution of linear systems with M, i.e., operation (4). The rest causes no major difficulties. Thus, matrix-by-vector product operations are relatively easy to implement efficiently on most computers. In the simplest case where the matrix is regularly structured, i.e., when it consists of a few diagonals this operation can be performed by multiplying diagonals with the vector [27]. The matrix can be stored in a rectangular array together with the offsets of these diagonals from the main diagonal. A number of generalizations of this formats for general sparse matrices have been proposed, the first of which is the ELLPACK-ITPACK format [32, 56]. Assuming that the maximum number of nonzero elements per row $jmax$ is small we can store the entries of the matrix in a real array $C(1:n, 1:jmax)$, the i-th row of which contains the nonzero elements of the i-th row of A. We also need an integer array $JC(1:n, 1:jmax)$ to store the column numbers of each entry of C. A more general scheme is the so-called Jagged-diagonal storage format. We start by reordering the rows of the matrix in decreasing number of nonzero elements. Then, a new data structure is built by constructing what we call "jagged diagonals". The leftmost element from each row are stored along with their column indices. This is followed by the second jagged diagonal consisting of the second nonzero element in each row, and so on. The number of these jagged diagonals is equal to the number of nonzero elements of the first row, i.e., to the largest number of nonzero elements per row. These two popular schemes lead to reasonably good performance for matrix-vector products on vector machines.

An operation that may be more troublesome in preconditioned Krylov subspace methods, is the preconditioning operations. At each step, we must solve a system of the form

$$Mz = y.$$

For SSOR and for ILU type preconditioners, M is the product of a sparse unit lower triangular matrix L and a sparse upper triangular matrix U. Let us assume that the linear system (1) arises from a centered finite difference discretization of an elliptic partial differential equation on a rectangle using an $n \times m$ mesh. This yields a linear system of size $N = nm$. For ILU(0) and SSOR the corresponding incomplete factors L and U have the same structure as the lower part and the upper part of the matrix A respectively. If the unknowns are labeled using the natural ordering, then the 'stencil' represented in Figure 2.1 establishes the data dependency between the unknowns in the lower triangular system solution when considered from the point of view of a grid of unknowns. Specifically, in order to compute the unknown in position (i, j) we only need to know the two unknowns in positions $(i-1, j)$ and $(i, j-1)$. As a result we can start computing x_{11} which does not depend on any other variable, and then use this to get $x_{1,2}$ and

$x_{2,1}$ simultaneously. Then these two values will in turn enable us to obtain $x_{3,1}, x_{2,2}$ and $x_{1,3}$ simultaneously and so on. The computation can be performed in wavefronts.

An important observation to make here is that the maximum degree of parallelism reached, or vector length in the case of vector processing, is the minimum of n_x, n_y for 2-D problems. For 3-D problems the parallelism is of the order of the maximum size of the sets of domain points $x_{i,j,k}$, where $i + j + k = lev$, a constant level lev. See [47] for details on vector implementations. However, as can be easily seen there is little parallelism or vectorization at the beginning and at the end of the sweep. Initially the degree of parallelism is very small, starting at one then increasing to its maximum to decrease back to 1 again at the end of the sweep. These first and last few steps may take a heavy toll on achievable speed-ups on massively parallel computers.

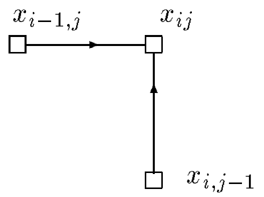

Figure 2.1 Stencil of the lower triangular matrix.

The simple scheme described above can be generalized to irregular grids. The technique, referred to as *level scheduling* is described for example in [6, 5, 43, 52] but we omit the details.

p	τ	ILUT time	GMRES time	tot. time	Iter.
Problem 1, Natural ordering – Unoptimized Version					
ILU(0)	NA	0.340E+00	0.125E+02	0.129E+02	81
1	0.1E-02	0.239E+01	0.160E+02	0.183E+02	90
3	0.1E-02	0.330E+01	0.106E+02	0.139E+02	48
5	0.1E-03	0.812E+01	0.787E+01	0.160E+02	30
10	0.1E-03	0.122E+02	0.933E+01	0.215E+02	25
Problem 1, Natural ordering – Optimized Version					
0	0.1E-03	0.532E+01	0.558E+01	0.109E+02	80
1	0.1E-02	0.241E+01	0.627E+01	0.868E+01	90
3	0.1E-02	0.332E+01	0.369E+01	0.701E+01	48
5	0.1E-03	0.808E+01	0.255E+01	0.106E+02	30
10	0.1E-03	0.121E+02	0.270E+01	0.148E+02	25

Table 2.1 Performance Comparison of GMRES(10)-ILUT(p,τ), for problem 1, without and with the use of level-scheduling.

We now report some numerical results for our three test problems using the ILUT preconditioner with different values of the parameters τ and fill-in p. For Problem 1 a comparison is given with the standard sequential technique in Table 2.1. The first line shows the performs with the standard ILU(0). We note that ILU(0) is slightly different from ILUT$(0, 0.0)$ which attempts to keep the largest elements during the incomplete factorization. However, as is shown in the optimized part of the table, the number of iterations is almost identical with that achieved with ILUT$(0, 0.0001)$. The iteration part of an optimized ILU(0)-GMRES(10) should therefore take about the same time as that with ILUT(0,0.0001) - GMRES(10) shown on the second part of the table. We also point out that the meaning of p is slightly different for that in [40]. We allow p fill-ins *in addition* to the number of nonzero elements in the lower part of A. Similarly with the upper part. In all the tests below the dimension of the Krylov subspace in GMRES is taken to be equal to 10. As is to be expected the number of iterations required for convergence decreases as the accuracy of the preconditioner increases. However, the cost of setting up the ILUT factorization increases rapidly as does the memory cost. In the 'optimized' part of the tables, level scheduling is used to optimize the forward and backward solves. In addition, the matrix vector multiplications that arise in the algorithm as well as in the level-scheduled forward and backward solutions are optimized by using what is referred to as the "jagged diagonal" format [38, 6]. The numbers in the 'memory' columns reported for Problems 2 and 3, represent the memory locations needed to store the ILUT factorization only. No attempt has been made to optimize the preprocessing phase and as a result the ILUT computation is likely to run close to scalar speed. Some optimization can be achieved but the returns are likely to be limited, at least with this approach. It is interesting to note that even if we disregard the ILUT time, the speed-up achieved here is modest. A factor of at most three is gained in the iteration part of the solution.

Problem 2, Natural ordering – Optimized Version						
p	τ	ILUT time	GMRES time	tot. time	Mem.	Iter.
1	0.100E-02	0.254E+01	0.950E+00	0.349E+01	151016	13
3	0.100E-02	0.302E+01	0.903E+00	0.392E+01	207418	11
5	0.100E-02	0.337E+01	0.862E+00	0.423E+01	241435	10
7	0.100E-02	0.364E+01	0.909E+00	0.455E+01	264305	10
5	0.100E-03	0.117E+01	0.123E+01	0.240E+01	51465	43
10	0.100E-03	0.169E+01	0.761E+00	0.245E+01	75822	20
15	0.100E-03	0.212E+01	0.935E+00	0.305E+01	96745	19

Table 2.2 Performance of Comparison of GMRES(10)-ILUT(p,τ), for problem 2, using various values of p and τ, with level-scheduling.

Problem 3 – Optimized Version of ILUT						
p	τ	ILUT time	GMRES time	tot. time	Mem.	Iter.
1	0.100E-02	0.531E+00	0.350E+01	0.403E+01	30060	158
2	0.100E-02	0.578E+00	0.320E+01	0.378E+01	35217	139
3	0.100E-02	0.630E+00	0.255E+01	0.318E+01	40174	105
4	0.100E-02	0.679E+00	0.235E+01	0.303E+01	44864	92
5	0.100E-02	0.723E+00	0.184E+01	0.256E+01	49240	68
7	0.100E-02	0.768E+00	0.205E+01	0.282E+01	56965	69
10	0.100E-02	0.819E+00	0.204E+01	0.286E+01	63324	61
5	0.100E-03	0.117E+01	0.123E+01	0.240E+01	51465	43
10	0.100E-03	0.169E+01	0.761E+00	0.245E+01	75822	20
15	0.100E-03	0.212E+01	0.935E+00	0.305E+01	96745	19

Table 2.3 Performance of Comparison of GMRES(10)-ILUT(p,τ), for problem 3, using various values of p and τ, with level-scheduling.

3 Polynomial preconditioning

When vector computers first became available, polynomial preconditioners were among the first alternatives proposed to the standard ILU preconditioning techniques [7, 25, 35, 26, 44, 53]. These methods consist of choosing a polynomial s and replacing the original linear system by

$$s(A)Ax = s(A)b \quad \text{or} \quad A(s(A)y) = b, \quad x = s(A)y \ , \tag{7}$$

which is then solved by a conjugate gradient type technique. The motivation here is that the only operations that are needed with the matrix are matrix by vector products.

The determination of an optimum polynomial s is guided by a requirement to have the preconditioned matrix $As(A)$ as close as possible to the identity matrix in some sense. This is achieved by exploiting the fact that we may adaptively construct a complex domain that roughly represents the spectrum of the matrix A. If we assume that the spectrum $\sigma(A)$ is contained in some continuous set E that encloses it, then a good choice for the polynomial s is obtained by solving the problem,

$$\begin{aligned}&\text{Find } s \in \Pi_k \text{ which minimizes:}\\ &\max_{\lambda\in E} |1 - \lambda s(\lambda)|.\end{aligned} \tag{8}$$

Several possible choices for E have been used in the literature. The simplest idea is to use an ellipse E [28, 29] that encloses an approximate convex hull of the spectrum. Then the shifted and scaled Chebyshev polynomials are nearly optimal, and even optimal in some special instances [17]. A second alternative is to use a polygon H that contains $\sigma(A)$ [46, 36]. The motivation here is that polygons may better represent the shape of an arbitrary spectrum. The polynomial is not explicitly known but it may be computed iteratively by a Remez-type algorithm.

An alternative considered by Johnson et al. in [25] for the Hermitian case and generalized to non-Hermitian matrices in [36] is to use a least squares polynomial instead of the infinity norm polynomial. We would then need to solve

$$\begin{aligned}&\text{Find } s \in \Pi_k \text{ that minimizes:}\\ &\|1 - \lambda s(\lambda)\|_w,\end{aligned} \tag{9}$$

where w is some weight function on the boundary of H and $\|.\|_w$ is the L_2-norm associated with the corresponding L_2 inner product. In [36] we used an L_2-norm associated with Chebyshev weights on the edges of a polygon H containing the spectrum and expressed the best polynomial as a linear combination of Chebyshev polynomials associated with the ellipse of smallest area containing H. If the contour of H consists of μ edges each with center c_i and half-length d_i, then the weight on each edge is defined by

$$w_i(\lambda) = \frac{2}{\pi}\, |d_i - (\lambda - c_i)^2|^{-1/2}. \tag{10}$$

With these weights, or any other Jacobi weights on the edges, there is a finite procedure to compute the best polynomial *that does not require numerical integration*; for details see [36]. An advantage of the least squares approach over the Chebyshev approach is its better robustness properties. In addition, there are common simple regions in the complex plane which ellipse cannot represent well. Thus, it is also more general. Its disadvantages are that it involves more arithmetic and that the maximum degree that can be used is limited, mostly due to stability constraints.

In the symmetric positive definite case, where H is reduced to an interval, it is not too difficult to obtain an approximate interval H containing the spectrum by exploiting the relationship between the Lanczos algorithm and the conjugate gradient method; see e.g., [11, 22]. In the non-Hermitian case a number of techniques can be used similarly. We next describe an implementation based on a combination with GMRES [42]. Eigenvalue estimates can be computed from the Hessenberg matrices generated from GMRES. Thus, the idea is to use a certain number, say m_1, of steps of GMRES to get eigenvalue estimates and to improve the current solution. The eigenvalue estimates are used to improve the current polygon H and to compute the next least squares polynomial. A number of steps (say m_2) of GMRES are then performed for the preconditioned system $s(A)Ax = s(A)b$. This procedure is outlined below.

ALGORITHM **3.1 Polynomial Preconditioned GMRES**

1. **(Re-) Start:** *Compute current residual vector* $r := b - Ax$.

2. **Adaptive GMRES run:** *Run* m_1 *steps of GMRES for solving* $Ad = r$. *Update* x *by* $x := x + d$. *Get eigenvalue estimates from the eigenvalues of the Hessenberg matrix.*

3. **Compute new polynomial**. *Refine* H *by using new eigenvalue estimates. Get new best polynomial* s_k.

4. **Polynomial Iteration:**

    - *Compute the current residual vector* $r = b - Ax$.
    - *Run* m_2 *steps of GMRES applied to* $As_k(A)q = r$. *Update* x *by* $x := x + s_k(A)q$.
    - *Test for convergence. If solution converged then stop else goto 1.*

Note that several enhancements to the above procedure can be made. First, the degree of the polynomial need not be fixed. We can start with a small degree polynomial and increase the degree as the convex hull becomes more and more accurate. In addition, we can combine polynomial preconditioning with some other preconditioning technique, such as one of the ones to be described later which offer a large degree of parallelism. 224z In the current version of our experimental Polynomial preconditioned GMRES (PGMR) code, we have not implemented the variable degree and use only diagonal preconditioning on the matrix A. In addition, our code sets $m_2 = m_1$.

Since the matrix-by-vector operation is the only sparse operation in PGMR, it is important to optimize it in order to achieve good overall performance. We used the jagged diagonal storage scheme [38, 37] for this purpose.

In the tables, m is the dimension of the Krylov subspaces, i.e., $m = m_1 = m_2$. The variable *deg* refers to the degree k of the polynomial used in the above algorithm, while *its* is the number of outer iterations, and *matvecs* is the total number of matrix by vector products needed to achieve convergence. One observation we can make here is that the times obtained for different values of the parameters m and *deg* show substantial variations. It is our experience that this behavior is quite common with polynomial preconditionings. Indeed, a poor estimation of the convex hull, may lead the iteration to proceed for many steps before it can correct H and reach a good convergence rate. On the other hand if several systems must be solved with the same matrix, then the information about the convex hull can be exploited for subsequent systems. Observe also that the best times are comparable to those obtained with the optimized ILUT preconditioners. We note that for problem 3, the performance was rather poor for small values of m and we omit the corresponding results. In general the larger m the better the approximation of the convex hull H obtained. However, the solution of the larger eigenvalue problems start to contribute significantly to the total times for larger m's.

Polynomial preconditioning – Problem 1				
m	deg.	total CPU	matvecs	its
10	5	0.227E+01	323	6
10	10	0.760E+01	1190	10
10	15	0.547E+01	870	5
15	5	0.141E+01	183	2
15	8	0.104E+01	147	2
15	10	0.112E+01	164	1
15	20	0.133E+01	214	1
20	5	0.148E+01	178	2
20	10	0.118E+01	169	1
20	15	0.146E+01	214	1
20	20	0.157E+01	239	1

Table 3.1 Performance of GMRES(m) with least-squares polynomial preconditioning for Problem 1.

Polynomial preconditioning – Problem 2				
m	deg.	total CPU	matvec's	its
10	5	0.914E+00	132	3
10	10	0.109E+01	168	2
10	15	0.152E+01	243	2
15	5	0.763E+00	102	2
15	10	0.839E+00	124	1
15	15	0.877E+00	134	1
20	5	0.848E+00	104	1
20	10	0.860E+00	119	1
20	15	0.976E+00	139	1

Table 3.2 Performance of GMRES(m) with least-squares polynomial preconditioning for Problem 2.

Polynomial preconditioning – Problem 3				
m	deg.	total CPU	matvec's	its
15	20	0.230E+01	1590	5
20	5	0.181E+01	883	8
20	10	0.223E+01	1324	6
20	15	0.178E+01	1126	4
20	20	0.175E+01	1157	3

Table 3.3 Performance of GMRES(m) with least-squares polynomial preconditioning for Problem 3.

4 The Red-Black ordering

A commonly used approach for solving large sparse linear systems on parallel computers is to resort to the red-black ordering which is the simplest form of the general multi-color orderings to be described later. In its simplest form the general problem that is addressed by multi-coloring techniques is to color the nodes of a graph so that neighboring vertices have different colors. In the case of a simple 2-dimensional finite difference grid (5-point operator) only two colors (referred to as Red and Black) are needed. This 2-color (red-black) ordering is illustrated in Figure 4.1 for a 4×3 grid.

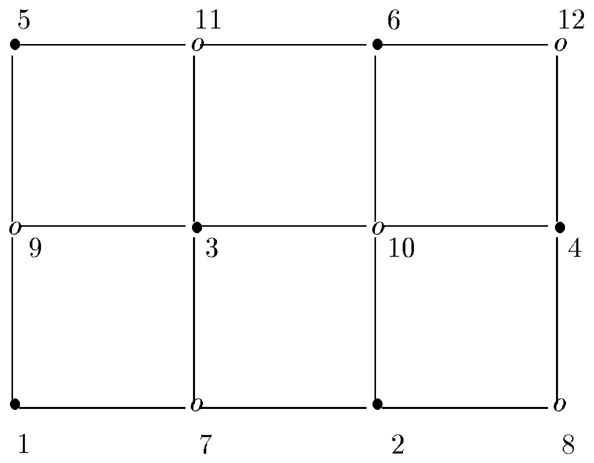

Figure 4.1 Red black labeling of a 4×3 grid. $\bullet$ = black node, o = red node.

If we reorder the unknowns in such a way as to list the red unknowns first together and then the black ones, we will obtain a system of the form

$$\begin{pmatrix} D_1 & E \\ F & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \qquad (11)$$

where D_1 and D_2 are diagonal matrices. Matrices that can be permuted into the above form are said to have property A [55]. Once the system has been reduced to the convenient form (11) a number of different techniques can be used for solving such systems. We now examine some of these in turn and discuss their advantages and disadvantages.

4.1 ILU preconditionings on the reduced system

Probably the simplest method that can be used for solving (11) is the standard ILU(0) on this block system. The degree of parallelism here is of order N. A drawback is that often the number of iterations is higher than with the natural ordering, but the approach may still be competitive for *easy* problems.

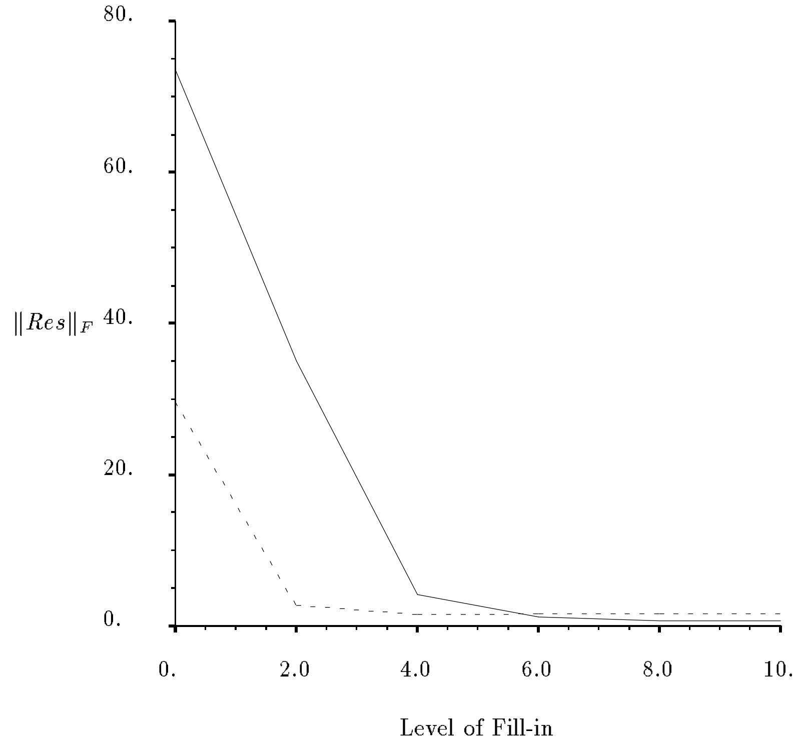

Figure 4.2 Frobenius norm of error matrix for the ILUT preconditioner as the accuracy increases. Dashed line: natural ordering, Solid line: Red-Black ordering.

Based on this, one can raise the interesting question as to whether or not the number of iterations can be reduced back to a competitive level by using a more accurate ILU factorization on the red-black system, e.g., ILUT [40]. Some recent experiments reveal that the answer is yes. In fact the situation is often reversed in that for the same level of fill-in p, the red-black ordering will outperform the natural ordering preconditioner for p large enough, in terms of number of iterations. A look at the plots of the Frobenius norms for the residual matrices $A - LU$, for a test matrix similar to that of model problems 1 and 2, reveals that indeed the incomplete factorization may become more accurate for the red-black matrices, as p increases, see Figure 4.2. The dashed line shows the Frobenius norms of the errors $\|A - LU\|_F$ for the natural ordering and the solid line shows the same norms for the Red-Black ordering using the same fill-in level. Notice that when the level of fill-in reaches 6, the quality of the RB preconditioner as measured by the F-norm of the error becomes better than that of the natural ordering using the same level of fill p. This reverses the situations obtained for smaller values of the fill-in level p.

Unfortunately, one notable difficulty with this approach, is that fill-ins will be introduced in the (2,2) blocks in L and U which loose their diagonal structure. As a result the degree of parallelism is severely reduced.

Table 4.1 shows the performance of GMRES(10)-ILUT(p,τ), for various values of p and τ for solving Problem 1 after the red-black ordering is applied. Similarly to Table 2.1, level scheduling is used to optimize the forward and backward solves and all the matrix vector multiplications are performed using the "jagged diagonal" format [38, 6]. Again, the preprocessing needed to compute the incomplete factorization itself is not optimized.

p	τ	ILUT time	GMRES time	tot. time	Iter.
Red-Black ordering – Unoptimized Version					
1	0.1E-02	0.182E+01	0.258E+02	0.276E+02	**
3	0.1E-02	0.248E+01	0.248E+02	0.273E+02	136
5	0.1E-03	0.414E+01	0.184E+02	0.225E+02	90
10	0.1E-03	0.704E+01	0.724E+01	0.143E+02	28
Red-Black ordering – Optimized Version					
1	0.1E-02	0.186E+01	0.108E+02	0.127E+02	**
3	0.1E-02	0.254E+01	0.970E+01	0.122E+02	136
5	0.1E-03	0.419E+01	0.683E+01	0.110E+02	90
10	0.1E-03	0.712E+01	0.255E+01	0.967E+01	28

Table 4.1 Performance comparison of GMRES(10)-ILUT(p,τ), for various values of p and τ, without and with the use of level-scheduling. Red-Black ordering.

Note that ILU(0) and MILU(0) were also tried in this example but they failed to converge as did ILUT(1,0.001) (The maximum number of steps allowed in GMRES is 160 in this test). In general, the times achieved are higher than those with the natural ordering for the low accuracy preconditioners (p=1 and p=3). However the situation improves drastically for the more accurate versions of ILUT and the conclusion on the comparison with the natural ordering is reversed for $p = 10$, i.e., the red-black ordering now outperforms the natural ordering. We have made the same observation in many other examples. An additional interesting feature here is that the ILUT time is also reduced with the red-black ordering.

4.2 Multiple step SOR/ SSOR preconditioners

As is suggested above for ILUT, high accuracy preconditioners on the Red-Black matrix can reduce the number of iterations required for convergence. However the fill-ins introduced in the "black" part of the system with such high level ILUs ruin the degree of parallelism achieved from the red-black ordering. A remedy against this shortcoming is to resort to similar *high-level SOR or SSOR preconditioners* instead of ILUT. This is a significant advantage of relaxation type preconditioners. We can define high level SOR/SSOR preconditioning by performing several SOR or SSOR steps in each preconditioning operation, instead of just one as is traditionally done. This can be combined with the flexible version of GMRES described earlier if needed.

In Table 4.1 we show the performance of a GMRES(m)-SOR(k) strategy based on this approach for Problem 1. The m parameter takes 2 different values $m = 10$ and $m = 20$. The k parameter, the number of steps in each SOR preconditioning takes several different values. Note that for $k = 1$ and $m = 10$ convergence is not achieved. An important observation we can make in this experiment and many other similar ones is that the optimal k for a fixed m is often achieved for values much larger than one.

m	k	Tot time	Iter. time	Iter.
10	1	0.377E+01	0.357E+01	**
10	5	0.240E+01	0.220E+01	60
10	10	0.205E+01	0.185E+01	30
10	15	0.229E+01	0.208E+01	21
10	20	0.225E+01	0.205E+01	18
10	30	0.185E+01	0.165E+01	10
10	35	0.206E+01	0.186E+01	10
20	1	0.246E+01	0.226E+01	139
20	5	0.154E+01	0.134E+01	36
20	10	0.124E+01	0.104E+01	18
20	15	0.146E+01	0.127E+01	15
20	20	0.164E+01	0.145E+01	13

Table 4.2 Performance of Red-Black SOR(k)–GMRES(m), multiple-step Gauss-Seidel Preconditioned GMRES for Problem 1, for different values of m and k.

We note that with $m = 20$ and $k = 10$, this approach is about 2.5 times faster that the best possible performance that we could obtain from the standard ILU preconditioners. If we compared the total times then the new approach would be even more attractive. Another attractive feature is that the preprocessing – coloring and forming the reduced system and coloring again – is highly parallel and quite easy to optimize whereas high level-of-fill ILU-based preconditioner offer a limited degree of parallelism.

m	k	Tot time	Iter.time	Iter.
Problem 2				
10	1	0.157E+01	0.137E+01	100
10	5	0.691E+00	0.492E+00	13
10	10	0.657E+00	0.459E+00	7
10	15	0.720E+00	0.520E+00	5
20	5	0.703E+00	0.505E+00	12
20	10	0.671E+00	0.471E+00	7
20	15	0.723E+00	0.522E+00	5
20	20	0.866E+00	0.664E+00	4
Problem 3				
20	5	0.316E+01	0.311E+01	261
20	10	0.494E+01	0.480E+01	238
20	15	0.579E+01	0.565E+01	184

Table 4.3 SOR(k)–GMRES(m), multiple-step Gauss-Seidel Preconditioned GMRES, for different values of m and k for problems 2 and 3.

Figure 4.3 shows the iteration times and total times obtained for Problem 2 as k varies from 1 to 30. Here the dimension m of the Krylov subspace is fixed to 10. Note that the preprocessing time which is roughly constant and equal to 0.2 second becomes nonnegligible for this problem because the iteration times are smaller than for Problem 1. Again we should emphasize that the preprocessing is a highly parallel process but it has not been not optimized in any way for these experiments. We show in Table 4.2 the results of a few runs for Problem 1 and, similarly, in Table 4.3 some results for Problems 2 and 3. In all the tables the total times include the times for permutations and conversion to the Ellpack-Itpack format required for optimizing the Red-black SOR iterations.

A second possibility not explored in this paper is to use block versions of the SOR iteration such as a line SOR relaxation.

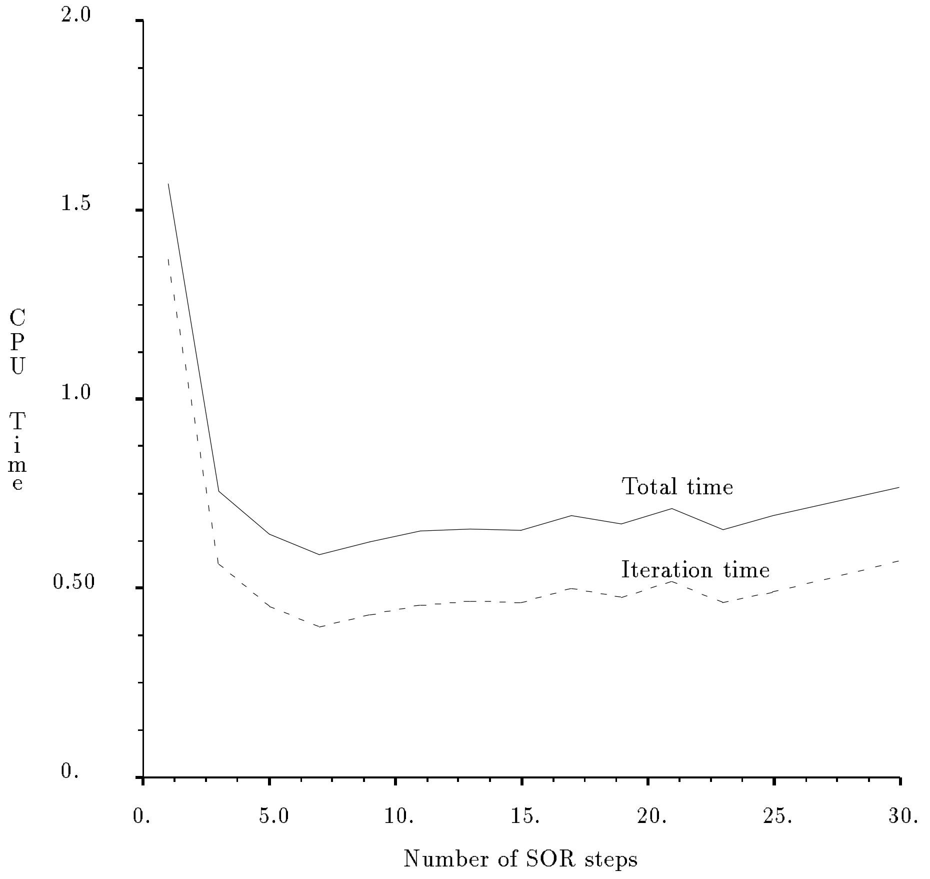

Figure 4.3 Iteration and total times for SOR(k)-GMRES(10) as k varies.

4.3 Reduced system approach

A second method that has been used in conjunction with the red-black ordering is to eliminate the red unknowns by forming the reduced system which involves only the black unknowns, namely,

$$(D_2 - FD_1^{-1}E)x_2 = b_2 - FD_1^{-1}b_1.$$

This linear system is again sparse and has about half as many unknowns. It has been observed that for 'easy problems' this reduced system can often be efficiently solved with only diagonal preconditioning. The preprocessing to compute the reduced system is highly parallel and inexpensive. In addition the reduced system is usually well-conditioned and has some interesting properties when the original system is highly nonsymmetric [15]. We should point out that it is not necessary to form the reduced system. This latter strategy is more often employed when D_1 is not diagonal, such in Domain Decomposition methods, but it can also have some uses in other situations. For example, applying the matrix to a given vector x can be achieved using nearest-neighbor communication in the original finite difference grid and can be performed efficiently without forming the Schur complement matrix. In addition this can save storage, which may be more critical in some applications.

We experimented with problems 1, 2, and 3, using this approach. A two-coloring of the matrix is done and then GMRES(m) is used with only diagonal preconditioning to solve the reduced system. The jagged diagonal data structure is used to get a good performance for the matrix-vector products which are the only sparse matrix operations required. The stopping criterion we used is to require the same condition on the residual norm for this reduced system as for the original problem. It is clear that the test on the reduced system does not guarantee that the corresponding solution to the original system will satisfy the same condition. Table 4.4 shows the times obtained with $m = 10$ and $m = 20$ for both problems. The iteration time is the time needed for GMRES to converge for the reduced system. The rest of the time includes the time for coloring, reducing the system, converting into jagged diagonal data structure, and back-substituting to obtain the 'red' unknowns. We observe that for Problem 3, the method essentially failed to converge and similarly for Problem 1, convergence is achieved for larger values of m only. In general when convergence is achieved the CPU times are much higher than with other techniques seen earlier. As a general rule, it seems that an approach that is based on a simple diagonal preconditioned GMRES iteration applied to the reduced system is not reliable.

m	Total time	GMRES time	GMRES steps
		Problem 1	
10	0.101E+02	0.837E+01	**
20	0.607E+01	0.436E+01	134
		Problem 2	
10	0.442E+01	0.268E+01	80
20	0.366E+01	0.194E+01	59
		Problem 3	
10	0.217E+01	0.191E+01	**
20	0.220E+01	0.195E+01	**

Table 4.4 Performance comparison of a reduced system approach using diagonal preconditioning GMRES(m) solver.

5 Multi-coloring

Generalizations of the red-black ordering have often been used in the context of PDE's as a means of introducing parallelism. The goal of multi-coloring is to color the vertices of an arbitrary adjacency graph in such a way that two adjacent nodes do not have a common color. The optimality issue, getting large color sets or small numbers of colors, is secondary. In fact in the context of iterative methods keeping preprocessing costs to a minimum is essential and in this framework, a sub-optimal multi-coloring is generally sufficient. The general technique of multi-coloring has been used in particular for understanding the theory of relaxation techniques [55, 48] as well as for deriving efficient alternative formulations of some relaxation algorithms [48, 20]. More recently, it was emerged as a useful tool for introducing parallelism in iterative methods, see for example [4, 3, 34, 14, 33]. Multi-Coloring is also commonly employed in a slightly different form – coloring elements (or edges) as opposed to nodes – in finite elements (or finite volume) techniques [10, 49]. Multi-coloring is especially useful in element-by-element techniques when forming the residual, i.e., in multiplying an unassembled matrix by a vector. The contributions of the elements of the same color can all be evaluated and applied simultaneously to the resulting vector [23, 16, 45]. We start by describing a simple technique for multi-coloring a graph. Then we will consider the same questions we addressed for the red-black ordering regarding ways of solving multi-colored systems.

5.1 A greedy multi-coloring algorithm

A basic and quite simple algorithm for obtaining a multi-coloring of an arbitrary graph, relying on a greedy approach can be sketched as follows.

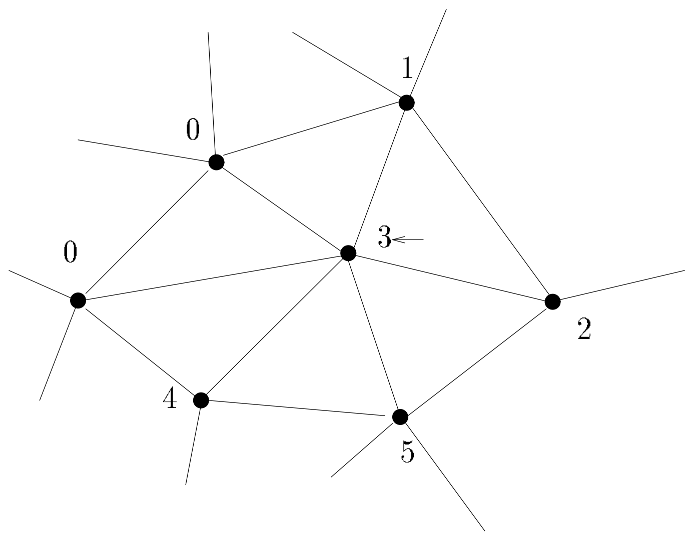

Figure 5.1 Illustration of the greedy approach for multi-coloring. The node being assigned a color is indicated by an arrow. It will be assigned color number 3, the smallest of the allowable color numbers, which are the positive integers other than 1, 2, 4, 5.

1. Initially assign color number zero (uncolored) to every node.
2. Choose an order in which to traverse the nodes.
3. Scan all nodes in the chosen order and at every node i do

$$\text{Color}(i) = \min\{k > 0 \mid k \neq \text{Color}(j), \forall\, j \in \text{Adj}(i))\},$$

As before, Adj(i) represents the set of nodes that are adjacent to node i. The color assigned to node i in step 3 is simply the smallest allowable color number which can be assigned to node i, where *allowable* means different from the colors of the neighbors and positive.

A few simple facts concerning this greedy algorithm, are the following.

- The initial order of traversal may be important in reducing the number of colors.

- If a graph is two-colorable, the algorithm will find the common 2-color (Red-Black) ordering for *Breadth-First* traversals or *Depth-First* traversals.
- The number of colors needed does not exceed the maximum degree of each node $+1$.

The parallelization of the algorithm is important and can be achieved in one of two ways. First, we can use the same algorithm and resort to some form of locking mechanism. The node being colored scans all neighbors in turn, and locks them (possibly after they are unlocked by some other node). Then it reads their colors and assigns a color number to its node and finally unlocks its neighbors. A second strategy, is to use a divide-and-conquer, or 'domain-decomposition' approach: recursively color the subdomains – then the nodes in the interfaces.

The idea of the greedy multi-coloring algorithm is known in Finite Element techniques (to color elements), see e.g., Berger-Brouays-Syre (1982), and Benantar and Flaherty. Wu [54] presents the greedy algorithm for multi-coloring vertices and uses it for SOR type iterations. Finally, the effect of multi-coloring has been extensively studied by Adams [4, 2] and Poole and Ortega [34].

5.2 Multiple step SOR / SSOR preconditioners

Just as for the red black ordering, we can use ILU0 or SOR, SSOR preconditioning on the reordered system. The parallelism of SOR/ SSOR is still of order N in general, but again one can anticipate a loss in the efficiency as the number of iterations may increase.

One important point of detail here concerns the implementation of the multicolor SOR technique. It is clear that similarly to the Red-Black ordering a Gauss-Seidel sweep will essentially require p matrix-vector products, where p is the number of colors. Specifically, let us assume that the matrix is stored in the well-known ELLPACK-ITPACK format and that the block structure of the permuted matrix is defined by a pointer array *iptr*. The index *iptr*(j) is the number of the first row in the j-th block. Thus, the pair $A(n1:n2,*), JA(n1:n2,*)$ represents the sparse matrix consisting of the rows $n1$ to $n2$ in the Ellpack format. The main diagonal of A is assumed to be stored separately in inverted form in a one-dimensional array *diag*. One single step of the multicolor SOR iteration will then take the following form.

Algorithm **5.1 Multi-color SOR sweep in the ELLPACK format**

```
0. Comment:   does one sweep of Gauss-Seidel for solving A y = rhs.
1.  do kol = 1, nkol                        | nkol = number of colors
2.     n1 = iptr(kol)                         | beginning of block # kol
3.     n2 = iptr(kol+1)-1                     | end of block # kol
4.     y(n1:n2) = rhs(n1:n2)
5.     do j=1,ndiag
6.        do i = n1, n2
7.            y(i) = y(i) - a(i,j)*y(ja(i,j))
8.        enddo
9.     enddo
6.     y(n1:n2) = diag(n1:n2) * y(n1:n2)    | D**(-1) * result
7.  enddo
```

In the above code, the outer loop, i.e., the loop in line 1, is sequential. The loop starting in line 6, is vectorizable / parallelizable. Clearly more parallelism can still be extracted in the combination of the loops in lines 5 and 6. For example, on the CM-2 using the CM FORTRAN, Thinking Machines' implementation of FORTRAN-90, these two loops can be rewritten in a one-line instruction as

```
y(n1:n2)=y(n1:n2)-sum(a(n1:n2,1:ndiag)*spread(y(ja(n1:n2),1,n2-n1+1),2)
```

In the above instruction the product of arrays is a component-wise product. The *spread* basically reproduces the vector $n2 - n1 + 1$ times in the first dimension whereas the *sum* 'reduces along the second dimension', i.e., it will sum all the components of the resulting array along the second dimension.

Going back to our three model problems, we note that we have already discussed the reduced system approach, in section 4.3. We have shown in Table 4.2 some timing results for the case where the reduced systems are solved with a simple-minded diagonal preconditioned GMRES iteration. We now show in Table 5.1 the analogue of Table 4.2 for the case where the reduced system is solved using a multicolored SOR(k) preconditioning. Thus, the reduced system obtained from the first coloring is multicolored again and reordered. In addition, the final reordered matrix obtained is processed to be stored in the Ellpack format in order to achieve a good performance in the SOR iteration as is described in Algorithm 5.1. Note that the preprocessing is higher than with the previous approach which required no reduction but only a coloring and a permutation. In the present case, we need to color, permute, reduce, color, and permute again. In addition, at the very last step, we also convert the permuted matrix into the Ellpack-Itpack format. These permutation and coloring operations have not been optimized and as they stand are highly sequential operations.

m	k	Total time	Iter. time	Iter.
10	1	0.306E+01	0.119E+01	69
10	5	0.291E+01	0.102E+01	20
10	10	0.284E+01	0.102E+01	11
10	15	0.308E+01	0.125E+01	9
10	20	0.321E+01	0.136E+01	8
10	25	0.334E+01	0.156E+01	7
20	1	0.270E+01	0.804E+00	41
20	3	0.257E+01	0.705E+00	20
20	5	0.268E+01	0.792E+00	15
20	10	0.288E+01	0.101E+01	11
20	15	0.308E+01	0.122E+01	9
20	20	0.338E+01	0.143E+01	8
20	25	0.355E+01	0.165E+01	7

Table 5.1 Performance of multicolor SOR(k)–GMRES(m) for Problem 1, using multiple-step Gauss-Seidel preconditioning for the reduced system – one level of reduction.

Similar results regarding Problem 2 and Problem 3, are given in Table 5.2 and Table 5.3 respectively. A general observation here is that the iteration times (GMRES times) are much lower than those seen in previous experiments.

m	k	Total time	Iter. time	Iter.
10	1	0.220E+01	0.352E+00	20
10	2	0.204E+01	0.254E+00	10
10	3	0.209E+01	0.230E+00	7
10	4	0.211E+01	0.273E+00	6
10	5	0.211E+01	0.257E+00	5
10	6	0.213E+01	0.254E+00	4
10	8	0.203E+01	0.254E+00	3
10	10	0.213E+01	0.310E+00	3
10	15	0.220E+01	0.339E+00	2

Table 5.2 Problem 2. One level of reduction, GMRES(m)-SOR(k) used to solve the reduced system.

m	k	Total time	Iter. time	Iter.
10	1	0.143E+01	0.114E+01	**
10	5	0.209E+01	0.180E+01	139
10	10	0.261E+01	0.232E+01	99
10	15	0.289E+01	0.260E+01	76
20	1	0.141E+01	0.112E+01	241
20	5	0.124E+01	0.954E+00	76
20	10	0.157E+01	0.128E+01	58
20	15	0.176E+01	0.148E+01	46

Table 5.3 Problem 3. One level of reduction, GMRES(m)-SOR(k) used to solve the reduced system.

5.3 Multi-level Reduced Systems

When we form the reduced system associated with the elimination of the unknowns of the first color we obtain a system that is again sparse. An idea that comes immediately to mind is to color the resulting system again and form the reduced system associated with the elimination of the first color. We will call this second reduced system the second-level reduced system. The process can be continued recursively a few times. However, as the level of the reduction increases, the reduced systems become denser. Nevertheless this approach can be successful if a small number of reductions are used and if the last reduced system is solved efficiently, e.g., by a multi-color SOR preconditioned technique. In addition, a variant of it will lead to a general purpose ILU preconditioner that can be viewed as a parallel version of ILUT which is the subject of a separate section.

We will now illustrate the multi-level reduced system technique in detail. Figure 5.2 shows the stages required in a 2-level reduction. On top from left to right we show the original matrix, and the matrix obtained after the first coloring and permutation. In this case the original matrix is a 100×100 matrix obtained from a 10×10 centered difference discretization of the Laplacean. So the multi-coloring algorithm yielded the usual Red-Black coloring in this case. In the middle we show on the left the reduced matrix obtained by eliminating the first half of the unknowns of the previous matrix. Then we multicolor the reduced matrix to obtain the permuted matrix on the right. We need 4 colors in this case. Finally at the bottom, we show on the left the matrix obtained by eliminating the unknowns associated with the first of the 4 colors in the previous matrix. The resulting matrix is multi-colored again to obtain the final permuted matrix shown on the right. The number of colors needed for this last matrix is 10. There are two reductions and three colorings in this example. The last coloring is only useful in case a multicolor SOR or SSOR preconditioning scheme will be applied to solve the last reduced system, or if the process of reduction were to be continued for another level. The sizes of the two reduced matrices are 50 and 35 respectively.

In the following description the right hand side is conceptually the last column of the $n \times (n+1)$ matrix A. If we call A_j the matrix obtained at the j- step of the reduction, $j = 1, \ldots, nlev$ then we first multi-color it, and then we permute the matrix according to the multicolor ordering to get a matrix in the form

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & X_j \end{pmatrix} \tag{12}$$

where D_j is a diagonal matrix. It is clear that the numbering of the colors is arbitrary. We can select the color yielding the largest color set to be the first color. Finally, we reduce the system by eliminating the unknowns of the first color, to get the next reduced matrix via the formula,

$$A_{j+1} = X_j - E_j D_j^{-1} F_j \ . \tag{13}$$

The transformations used in the elimination process, i.e., the E_j's, need not be saved but the upper F_j matrices must be since they will be used to back-solve for the remaining equations. Regarding the matrix reordering, we can either permute the matrix or simply keep a permutation array.

Algorithm **5.2 Multilevel Reduced System (MRS) Solution Algorithm**

```
1. Let $A_0 = A$;
2. Do for $j = 0, 1, \ldots, nlev - 1$
     Find a multi-coloring of the graph of $A_j$
     Call color 1, the color with the largest number of elements.
     Permute the matrix according to the multicolor ordering into the form (12).
     Reduce the system by eliminating color number 1, i.e., get the system (13).
   enddo
3. Solve the last reduced system with, e.g., GMRES with multistep-multicolor
     SOR preconditioning.
4. Do for $j = nlev - 1, ..., \ldots, 0$
     Backsolve for $A_j$ from the known solution for $A_{j+1}$.
   enddo
```

Figure 5.2 The MRS approach for a 5-point matrix of size 100×100

One of the advantages of this approach comes from the fact that the last matrix for which we must solve a linear system is usually much smaller than the original matrix. This, compounded with the fact that it also usually takes fewer steps for the higher level iterations to converge, makes the scheme quite attractive. The drawbacks are its complexity and the fact that the reduced matrices will eventually become quite dense. Beyond a few (say 2 or 3) levels, further reductions are no longer cost-effective. This motivates the search for an incomplete factorization technique which will basically consist of dropping small elements just as is done for the standard Gaussian Elimination. Before this, we would like to show a number of experiments to compare these techniques using our three model problems.

Some of the results obtained are summarized in Table 5.4. The numbers in the column labeled 'memory' show the memory requirement the 'factorization' alone. Comparing with the results seen in Section 4.3 which correspond to a level of reduction equal to $nlev = 1$, we can observe that in general the times are higher. In addition the memory requirement may become prohibitive for larger problems. This strongly suggests using dropping as a means of reducing the memory cost, as is done in the standard ILU techniques.

$nlev$	m	k	tot CPU	its CPU	memory	itstot
Problem 1						
2	10	1	0.853E+01	0.464E+01	399129	72
2	10	5	0.750E+01	0.366E+01	399129	19
2	10	10	0.805E+01	0.426E+01	399129	11
Problem 2						
2	10	1	0.486E+01	0.105E+01	399129	17
2	10	5	0.477E+01	0.895E+00	399129	4
2	10	10	0.485E+01	0.970E+00	399129	2
Problem 3						
2	10	1	0.648E+01	0.315E+01	66107	250
2	10	5	0.890E+01	0.563E+01	66107	132
2	10	10	0.109E+02	0.758E+01	66107	95

Table 5.4 Two levels of reduction, GMRES(m)-SOR(k) used to solve the reduced system.

5.4 ILUM: a Multi-elimination ILU factorization

As was suggested above, the successive reduced systems become more and more expensive to form and store as the number of levels increases. An alternative to cut memory and preprocessing costs is to use a simple dropping strategy as we form the reduced systems. For example, we can drop any fill-in element introduced, including pivot elements, whenever its size is less than a tolerance times the 2-norm of the original row.

Thus, a modified version of Algorithm 5.2 can be used as a preconditioner, i.e., it will provide an approximate solution $M^{-1}v$, for any given v. The modification will consist of the following changes. First, in the preprocessing phase, dropping is used when constructing the reduced system (13). This means that (13) is replaced by

$$A_{j+1} = X_j - E_j D_j^{-1} F_j + R_j \tag{14}$$

in which R_j is the matrix of the elements that have been dropped. We note that in (14) and throughout this section it is no longer assumed that the right-hand-side is the last column of the matrix.

Second, the linear systems to be solved in the last reduced system, i.e., in step 3 of Algorithm 5.2, need no longer be solved accurately. In fact we can solve it according to the level of tolerance that we have allowed in the preprocessing phase, although we do not know how to actually choose an optimal residual norm stopping criterion. Observe that since we would like to solve the linear system inaccurately, it is imperative to use the flexible variant of GMRES described in Section 2. Alternatively, we could simply use a fixed number of multi-color SOR iterations. The implementation of the ILUM preconditioner (ILU with Multi-elimination) corresponding to this strategy is rather complicated and involves several parameters. The detailed description of the algorithm as well as some analysis and additional experiments will be the object of a forthcoming paper.

However, we would like to summarize the description of the the two phases of the algorithm, namely the preprocessing phase and the forward-backward solution phase. Essentially, the preprocessing phase consists of steps 1, and 2 of Algorithm 5.2 except that the operations related to the right-hand-side are ignored and, instead, the matrices E_j obtained in (12) after the coloring and permutation are now also saved since they will be needed for each forward sweep in the preconditioning operations. Thus we have a succession of block-LU factorizations of the form,

$$P_j A_j P_j^T = \begin{pmatrix} D_j & F_j \\ E_j & X_j \end{pmatrix} = \begin{pmatrix} I & O \\ E_j D_j^{-1} & I \end{pmatrix} \times \begin{pmatrix} D_j & F_j \\ O & A_{j+1} \end{pmatrix} \tag{15}$$

with A_{j+1} defined by (14). We denote by x_j the right hand side associated with the matrix A_j, and recursively, x_{j+1} the transformed right hand side associated with the matrix A_{j+1} and call E'_j the matrix $E'_j = F_j D_j^{-1}$. We can either permute the vector at each level as is suggested by the above factorizations, or we can multiply all permutations together during the factorization phase so that we will only have to apply the permutation at the beginning and the reverse permutation at the end of the backward/forward solution process. Then the forward and backward and solutions consist of an algorithm with the following structure.

Algorithm **5.3** *LU solve for ILUM preconditioner*

```
1. Apply permutation to right-hand-side $x$.
2. Forward solution. For $j = 0, 1, \ldots, nlev - 1$ do: $x_{j+1} := x_{j+1} - E'_j x_j$
3. Solve last reduced system with a relative tolerance $\epsilon$ to get $x_{nlev}$.
4. Backward solution. For $j = nlev - 1, \ldots, 0$ do: $x_j := D_j^{-1}(x_j - F_j x_{j+1})$.
5. Permute the solution vector back to the original ordering.
```

Parameters			Performance				
nlev	τ-ILUM	tol prec	tot CPU	its CPU	memory	its-out	its-tot
1	0.0001	0.100	4.224	2.335	107891	9	70
2	0.0001	0.100	9.141	5.563	142591	9	69
3	0.0001	0.100	12.939	7.349	166454	9	78
1	0.0001	0.010	3.395	1.502	107891	4	56
2	0.0001	0.010	7.216	3.704	142591	4	52
3	0.0001	0.010	11.061	5.409	166454	5	65
1	0.0010	0.010	4.193	2.223	107891	4	85
2	0.0010	0.010	10.419	6.812	142512	6	116
3	0.0010	0.010	11.826	6.854	158788	7	115
1	0.0010	0.100	4.179	2.323	107891	9	70
2	0.0010	0.100	8.127	4.690	142512	10	63
3	0.0010	0.100	9.609	4.678	158788	10	64
1	0.0100	0.010	4.390	2.487	107891	5	103
2	0.0100	0.010	7.841	4.600	131543	20	109
3	0.0100	0.010	12.009	7.769	149984	36	143

Table 5.5 Performance of GMRES(10)-ILUM preconditioning for Problem 1

We now report a few experiments with a preliminary implementation of ILUM. In all the experiments the innermost reduced system is solved by SOR(1)-GMRES(20) and the outer accelerator is GMRES(10). As usual *nlev* is the number of levels in the reduction. The parameter τ-ILUM shown in the second column of the tables is the dropping tolerance τ used in the preconditioning. The parameter $tol - prec$ in the third column is the tolerance used in the GMRES(20)-SOR(1) iteration of the innermost reduced linear system to solve. Again, the numbers in the column labeled 'memory' show the memory requirement the ILUM 'factorization' alone. The numbers $its - out$ refer to the number of outer iterations required, i.e., the number of ILUM preconditioned GMRES iterations to converge. The numbers $its - tot$ refer to the total number of inner iterations required, i.e., the overall total number of matrix vector products in the calls to GMRES(20)-SOR(1) needed to solve all the reduced systems occurring throughout the run to solve the linear system.

Parameters			Performance				
nlev	τ-ILUM	tol prec	tot CPU	its CPU	memory	its-out	its-tot
1	0.0001	0.010	3.127	1.292	107745	5	41
2	0.0001	0.010	5.617	2.262	141167	4	30
3	0.0001	0.010	8.374	3.337	158741	4	30
1	0.0010	0.001	2.777	0.916	106592	3	33
2	0.0010	0.001	5.526	2.307	136035	4	40
3	0.0010	0.001	7.510	2.750	152345	4	37
1	0.0010	0.010	3.065	1.117	106592	4	33
2	0.0010	0.010	5.168	1.896	136035	4	29
3	0.0010	0.010	7.503	2.633	152345	5	33
1	0.0010	0.100	3.861	1.938	106592	10	35
2	0.0010	0.100	6.713	3.321	136035	10	35
3	0.0010	0.100	8.799	3.886	152345	11	32
1	0.0100	0.100	4.165	2.411	97870	12	55
2	0.0100	0.100	5.395	2.538	114047	12	37
3	0.0100	0.100	6.959	2.916	127401	13	40

Table 5.6 Performance of GMRES(10)-ILUM preconditioning for Problem 2.

Parameters			Performance				
nlev	τ-ILUM	tol prec	tot CPU	its CPU	memory	its-out	its-tot
1	0.0001	0.100	1.471	1.197	19438	6	232
2	0.0001	0.100	3.250	2.794	25595	8	246
3	0.0001	0.100	2.962	2.300	30076	8	175
4	0.0001	0.100	3.231	2.350	34479	9	185
5	0.0001	0.100	3.456	2.392	38760	10	179
10	0.0001	0.100	3.482	1.412	53650	10	95
15	0.0001	0.100	4.238	1.287	64381	11	67
20	0.0001	0.100	4.738	1.037	72382	10	40
25	0.0001	0.100	5.242	0.995	78454	10	37
1	0.0010	0.100	2.868	2.594	19432	16	473
2	0.0010	0.100	3.171	2.737	25180	25	260
3	0.0010	0.100	2.993	2.395	29562	32	199
4	0.0010	0.100	4.186	3.463	32341	57	205
5	0.0010	0.100	3.930	3.063	35900	48	174

Table 5.7 Performance of GMRES(10)-ILUM preconditioning for Problem 3.

Method	Total time	Iteration time
Problem 1		
Non-optimized ILU(0)-GMRES(10)	12.5	12.9
ILUT-GMRES	10.6	2.55
Poly-GMRES	1.04	1.04
Red-Black SOR(k)-GMRES(m)	1.24	1.04
MRS – SOR(k)-GMRES(m)	2.57	0.71
ILUM – GMRES(m)	3.39	1.50
Problem 3		
Non-optimized ILU(0)-GMRES(10)	5.79	7.13
ILUT-GMRES	2.40	1.23
Poly-GMRES	1.81	1.81
Red-Black SOR(k)-GMRES(m)	3.16	3.11
MRS / SOR(k)-GMRES(m)	1.24	0.954
ILUM – GMRES(m)	5.24	0.995

Table 5.8 Best performances obtained for each of the preconditioners tested for Problems 1 and 3. The ranking is based on the iteration time only, i.e., it ignores all preprocessing costs.

6 Summary of results and conclusion

Table 5.8 is a summary of the best performances obtained in previous tables with each of the preconditioners tested for Problems 1 and 3. It is only indicative of achievable performance with each of the preconditioners and should not in any way be taken as an absolute measure of comparison since the tables are not exhaustive. The reason why we did not include the preprocessing times in the comparisons is that they have not been optimized as yet. When preprocessing should be accounted for then polynomial preconditioning is certainly quite attractive except for the fact that performance can be variable. Here are a few observations we can make regarding these tests.

- Despite a rather preliminary implementation, ILUM is very competitive for Problem 3, the hardest to solve among the three problems. Using ILUM with 25 levels of reduction gives close to the best time achieved for this problem.
- For problems 1 and 2, the one-level reduced system (no dropping) and multi-color SOR performed best. In addition the preprocessing cost for the MRS approach with one level of reduction is modest.
- The polynomial preconditioning approach is attractive because of the negligible preprocessing cost involved. In fact the only additional cost incurred lies in the computation of the least squares polynomial.

- For the multi-level reduced system approach without dropping, it seems that the one level reduction performs usually better than with levels higher than one. It is also usually better than the zero-level (i.e. multicolor SOR preconditioned GMRES).
- One difficulty with the better performing preconditioners, is that they tend to involve more parameters. This seems to be one of the unavoidable difficulties with parallel algorithms, simply because they are often more complex than the sequential ones.
- In general the preprocessing is highly parallel in all the algorithm seen, except for the standard ILUT, so the costs reported for these times can be dramatically reduced in a parallel environment.

We would like to conclude by observing that in order to develop efficient massively parallel preconditioners it seems important to mix strategies that offer a high degree of parallelism, e.g., multi-color orderings, with strategies that increase the accuracy of the preconditioners, e.g., k-step SOR. The higher accuracy tends to compensate for the poorer convergence rates that might otherwise result from the reordering.

Our next step is to implement these ideas on a massively parallel computer such as the Thinking Machine CM-5. We expect that, generally speaking, similar conclusions will hold true. However, one potential difficulty may be the implementation of the preprocessing phase which is rather complicated for the better performing preconditioners. This may make simpler techniques such as polynomial preconditionings or the SOR(k) – SSOR(k) preconditionings combined with just one level of reduction look quite attractive overall.

Acknowledgement. The author would like to acknowledge the support of the Minnesota Supercomputer Institute which provided the computer facilities and an excellent research environement to conduct this research.

References

[1] L. Adams. M-step preconditioned conjugate gradient methods. *SIAM J. Sci. Statist. Comp*, 6:452–463, 1985.

[2] L. Adams and H. Jordan. Is SOR color-blind? *SIAM J. Sci. Statist. Comp*, 6:490–506, 1985.

[3] L. Adams and J. Ortega. A multi-color SOR Method for Parallel Computers. In *Proceedings 1982 Int. Conf. Par. Proc.*, pages 53–56, 1982.

[4] L. M. Adams. *Iterative algorithms for large sparse linear systems on parallel computers.* PhD thesis, Applied Mathematics, University of Virginia, Charlottsville, VA, 22904, 1982. Also NASA Contractor Report 166027.

[5] E. C. Anderson. Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations. Technical Report 805, CSRD, University of Illinois, Urbana, IL, 1988. MS Thesis.

[6] E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *International Journal of High Speed Computing*, 1:73–96, 1989.

[7] S. F. Ashby. *Polynomial Preconditioning for Conjugate Gradient Methods.* PhD thesis, Computer Science Dept. , University of Illinois, Urbana, IL, 1987. Available as Technical Report 1355.

[8] S. F. Ashby, T. A. Manteuffel, and P. E. Saylor. Adaptive polynomial preconditioning for Hermitian indefinite linear systems. *BIT*, 29:583–609, 1989.

[9] O. Axelsson. A generalized conjugate gradient, least squares method. *Num. Math.*, 51:209–227, 1987.

[10] M. Benantar and J. E. Flaherty. A Six color procedure for the parallel solution of Elliptic systems using the finite Quadtree structure. In J. Dongarra, P. Messina, D. C. Sorenson, and R. G. Voigt, editors, *Proceedings of the fourth SIAM conference on parallel processing for scientific computing*, pages 230–236, 1990.

[11] P. Concus, G. H. Golub, and D. P. O'Leary. A generalized conjugate gradient method for the numerical solution of elliptic partial differential equations. In James R. Bunch and Donald J. Rose, editors, *Sparse Matrix Computations*, pages 309–332, New York, 1976. Academic Press.

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

[13] H. C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations.* PhD thesis, Yale University, Computer Science Dept., New Haven, CT., 1982.

[14] H. C. Elman and E. Agron. Ordering techniques for the precondiotioning conjugate gradient method on parallel computers. Technical Report UMIACS-TR-88-53, UMIACS, University of Maryland, College Park, MD, 1988.

[15] H. C. Elman and G. H. Golub. Iterative methods for cyclically reduced non-self-adjoint linear systems. Technical Report CS-TR-2145, Dept. of Computer Science, University of Maryland, College Park, MD, 1988.

[16] R. M. Ferencz. *Element-by-element preconditioning techniques for large scale vectorized finite element analysis in nonlinear solid and structural mechanics.* PhD thesis, Applied Mathematics, Stanford, CA, 1989.

[17] B. Fischer and R. Freund. Chebyshev polynomials are not always optimal. *J. Approximation Theory*, 65:–, 1991.

[18] R. Freund, M. H. Gutknecht, and N. M. Nachtigal. An implementation of the Look-Ahead Lanczos algorithm for non-Hermitian matrices, Part I. Technical Report 90-11, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1990.

[19] R. Freund and N. M. Nachtigal. An implementation of the look-ahead lanczos algorithm for non-Hermitian matrices, Part II. Technical Report 90-11, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1990.

[20] R. S. Varga G. H. Golub. Chebyshev semi iterative methods successive overrelaxation iterative methods and second order Richardson iterative methods. *Numer. Math.*, 3:147–168, 1961.

[21] Alan George and Joseph W-H Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31:1–19, 1989.

[22] A. L. Hageman and D. M. Young. *Applied Iterative Methods*. Academic Press, New York, 1981.

[23] T. J. R. Hughes, R. M. Ferencz, and J. O. Hallquist. Large-scale vectorized implicit calculations in solid mechanics on a cray x-mp/48 utilizing ebe preconditioning conjugate gradients. *Computer Methods in Applied Mechanics and Engineering*, 61:215–248, 1987.

[24] K. C. Jea and D. M. Young. Generalized conjugate gradient acceleration of nonsymmetrizable iterative methods. *Linear Algebra Appl.*, 34:159–194, 1980.

[25] O. G. Johnson, C. A. Micchelli, and G. Paul. Polynomial preconditionings for conjugate gradient calculations. *SIAM J. Numer. Anal.*, 20:362–376, 1983.

[26] T. L. Jordan. Conjugate gradient preconditioners for vector and parallel processors. In G. N. Birkhoff and A. Schoenstadt, editors, *Elliptic problem solvers II, Proceedings of the elliptic problem solvers conference, Monterey CA. , Jan 10-12 1983*, pages 127–139. Academic Press, 1983.

[27] T. I. Karush, N. K. Madsen, and G. H. Rodrigue. Matrix multiplication by diagonals on vector/parallel processors. Technical Report UCUD, Lawrence Livermore National Lab., Livermore, CA, 1975.

[28] T. A. Manteuffel. The Tchebychev iteration for nonsymmetric linear systems. *Numer. Math.*, 28:307–327, 1977.

[29] T. A. Manteuffel. Adaptive procedure for estimation of parameter for the nonsymmetric Tchebychev iteration. *Numer. Math.*, 28:187–208, 1978.

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

[31] T. C. Oppe, W. Joubert, and D. R. Kincaid. Nspcg user's guide. a package for solving large linear systems by various iterative methods. Technical report, The University of Texas at Austin, 1988.

[32] T. C. Oppe and D. R. Kincaid. The performance of ITPACK on vector computers for solving large sparse linear systems arising in sample oil reservoir simulation problems. *Communications in applied numerical methods*, 2:1–7, 1986.

[33] J. M. Ortega. *Introduction to Parallel and Vector Solution of Linear Systems*. Plenum Press, New York, 1988.

[34] E. L Poole and J. M. Ortega. Mullticolor ICCG methods for vector computers. *SIAM J. Numer. Anal.*, 24:1394–1418, 1987.

[35] Y. Saad. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Stat. Sci. Comput.*, 6:865–881, 1985.

[36] Y. Saad. Least squares polynomials in the complex plane and their use for solving sparse nonsymmetric linear systems. *SIAM J. Num. Anal.*, 24:155–169, 1987.

[37] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Scient. Stat. Comput.*, 10:1200–1232, 1989.

[38] Y. Saad. SPARSKIT: A basic tool kit for sparse matrix computations. Technical Report 90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.

[39] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. Technical Report 91-279, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, Minnesota, 1991.

[40] Y. Saad. ILUT: a dual threshold incomplete ILU factorization. Technical report 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992.

[41] Y. Saad and M. H. Schultz. Conjugate gradient-like algorithms for solving nonsymmetric linear systems. *Mathematics of Computation*, 44(170):417–424, 1985.

[42] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7:856–869, 1986.

[43] J. H. Saltz. Automated problem scheduling and reduction of synchronization delay effects. Technical Report 87-22, ICASE, Hampton, VA, 1987.

[44] M. K. Seager. Parallelizing conjugate gradient for the CRAY X-MP. Technical report, Lawrence Livermore National Lab, Livermore, CA, 1984.

[45] F. Shakib. *Finite element analysis of the compressible Euler and Navier Stokes Equations.* PhD thesis, Aeronautics Dept., Stanford, CA, 1989.

[46] D. C. Smolarski and P. E. Saylor. An optimum iterative method for solving any linear system with a square matrix. *BIT*, 28:163–178, 1988.

[47] H. A. van der Vorst. High performance preconditioning. *SIAM j. Scient. Stat. Comput.*, 10:1174–1185, 1989.

[48] R. S. Varga. *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1962.

[49] V. Venkatakrishnan, H. D. Simon, and T. J. Barth. A MIMD Implementation of a Parallel Euler Solver for Unstructured Grids. Technical Report RNR-91-024, NASA Ames research center, Moffett Field, CA, 1991.

[50] P. K. W. Vinsome. Orthomin, an iterative method for solving sparse sets of simultaneous linear equations. In *Proceedings of the Fourth Symposium on Resevoir Simulation*, pages 149–159. Society of Petroleum Engineers of AIME, 1976.

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

[52] O. Wing and J. W. Huang. A computation model of parallel solution of linear equations. *IEEE Transactions on Computers*, C-29:632–638, 1980.

[53] Y. S. Wong. Solving large elliptic difference equations on CYBER 205. *Parallel Comput.*, 6:195–207, 1988.

[54] C. H. Wu. A multicolour SOR method for the finite-element method. *J. of Comput. and App. Math.*, 30:283–294, 1990.

[55] D. M. Young. *Iterative solution of large linear systems.* Academic Press, New-York, 1971.

[56] D. M. Young, T. C. Oppe, D. R. Kincaid, and L. J. Hayes. On the use of vector computers for solving large sparse linear systems. Technical Report CNA-199, Center for Numerical Analysis, University of Texas at Austin, Austin, Texas, 1985.

[57] Z. Zlatev. Use of iterative refinement in the solution of sparse linear systems. *SIAM J. Numer. Anal.*, 19:381–399, 1982.