

# Distributed ILU(0) and SOR Preconditioners for Unstructured Sparse Linear Systems \*

S. Ma and Y. Saad †

February 23, 1998

## Abstract

There are two common ways of implementing ILU or (S)SOR preconditioned Krylov subspace methods on parallel computers. The first resorts to multicoloring of the adjacency graph and often results in a parallelism of order  $N$ , where  $N$  is the dimension of the matrix. This strategy sometimes suffers from the deterioration of the rate of convergence. The second technique exploits the intrinsic parallelism available in the original forward and backward solutions. This approach does not suffer from the deterioration of the convergence rate since the resulting algorithm is mathematically equivalent to the original one. On the other hand, the maximum parallelism is limited by the length of the wavefronts which are often nonuniform. In this paper we consider the implementation of these two approaches on distributed memory computers and compare their performance on the CM-5. We have found that for the problems tested ILU(0) with multicoloring outperforms the other alternatives.

**Key words:** Preconditioning; Distributed ILU(0); point SOR preconditioner; Block SOR; Parallel preconditioners.

**AMS (MOS) Subject Classification:** 65F10.

---

\*Work supported by ARPA under grant number NIST 60NANB2D1272, by NSF under grant number NSF/CCR-9214116, and by AHPARC (University of Minnesota) under Army Research Office grant number DAAL03-89-C-0038.

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

# 1 Introduction

The preconditioning operations used in standard preconditioned Krylov subspace methods are often difficult to implement efficiently on parallel computers. A technique that is often used is to reorder the equations through *multi-coloring*. This typically provides a parallelism of order  $N$ , where  $N$  is the dimension of the coefficient matrix. For example, if the matrix has property A, as is the case for the standard 5-point matrices obtained from centered Finite Difference (FD) discretizations of elliptic Partial Differential Equations (PDE's), we can color the grid-points with two colors such that the unknowns associated with one color only coupled with unknowns of the other color. Using this coloring we can reorder the matrix into the form

$$\begin{pmatrix} D_1 & E \\ F & D_2 \end{pmatrix}$$

in which  $D_1, D_2$  are diagonal matrices. This structure can be exploited in several different ways. For example, all the unknowns associated with one color can be eliminated, leading to a reduced system of size half that of the original system. The resulting reduced system is often well-conditioned and this approach works quite well for symmetric problems. This 'two-coloring' often referred to as a red-black or checkerboard ordering, can be generalized to arbitrary sparse matrices by using heuristic graph-coloring and independent set orderings, see, e.g., [12, 6, 5]. However, a well-known drawback of this approach is that the resulting convergence of the preconditioned Krylov subspace technique, is often poorer than with the original ordering.

An alternative means of parallelizing standard preconditioners, is to use an approach known in the literature as *level-scheduling* or *wavefront* technique. This technique preserves the original ordering but performs the tasks of the forward and backward solution in a different order which can exploit parallelism. Thus, this is a parallel implementation of the original (sequential) algorithm in which parallelism is extracted by examining dependences between the tasks. The disadvantage of this approach, is that the maximum parallelism available is determined by the lengths of the wavefront, which are often short and nonuniform. A clear advantage is that it is mathematically equivalent with its sequential counterpart which is typically well understood.

Thus, among the possible alternatives available for implementing preconditioned conjugate gradient-type methods on parallel computers, we have two broad options. A first option is to use a standard method on the original, non reordered system, but to look for parallelism in the backward and forward sweeps. The intrinsic properties of this approach are well understood but the intrinsic parallelism is moderate. The second approach is to resort to multicoloring techniques which offer substantially more parallelism but which may require more steps to converge. In this paper we will examine these two broad options and compare them on some model problems on a CM-5 parallel computer. The methods which we consider are designed for general sparse irregularly structured matrices. We exploit the data structures and general approach of *distributed sparse matrices* described in [14, 16]. The underlying assumption we make is that we have a linear system which is row-wise distributed among a number of processors and we would like to implement a

distributed preconditioned Krylov subspace method to solve such a system. The methods we consider are akin to domain decomposition techniques but are geared towards general sparse linear systems.

The outline of the paper is as follows. In section 2 we discuss the two main ideas used in the literature to parallelize ILU and SOR or SSOR preconditioners. In Section 3, we describe distributed versions of ILU(0) and SOR preconditioners. Section 4 describes our experiments with a few model problems on the CM-5. Finally, we summarize our findings and conclude in Section 5.

## 2 Multi-coloring and Wavefront reordering

Multi-coloring consists of assigning a color to each node of a mesh such that no two nodes coupled by an equation have the same color. In graph terminology this termed *graph coloring* of the adjacency graph of the sparse matrix representing the sparse linear system. For the 5-point discretization of the Laplacian it is well-known that this can be achieved with two colors. For a square domain, this coloring can be performed by a checkerboard-like manner. Similarly, four colors are needed to color the grid points of the same color of the 9-point Laplacian. Since there are no couplings between nodes of the same color the values at the node associated with one color can be updated simultaneously in a forward or backward sweep associated with an SOR or ILU(0) preconditioning. However, it is often observed that the convergence rate for the reordered systems deteriorates. For the model problem, SSOR and the preconditioned-CG with the Red/Black ordering have a worse convergence rate than with the natural ordering, while SOR has the same convergence rate if the optimal  $\omega$  is used. The following table, see e.g., [4], shows the rates of convergence for the SOR, SSOR, and ILU(0) preconditioned CG methods with natural and red/black ordering for the 5-point Laplacian matrix, when the optimal  $\omega$  is used.

	SOR	SSOR	ILU-CG
Natural Ordering	$O(h)$	$O(h)$	$O(\sqrt{h})$
Red/Black Ordering	$O(h)$	$O(h^2)$	$O(h)$

Table 1: Rate of convergence as a function of the mesh-size  $h$ , when reordering is used

For the nine-point Laplacian with properly selected  $\omega$  the convergence rate of SOR remains the same as with the natural ordering [8]. O’Leary [8] considered several other ordering schemes for the 9-point Laplacian and has shown that the convergence rate of SOR iteration is not worse than that of the natural ordering. From this viewpoint, SOR has a slight advantage over SSOR in the presence of reordering via multi-coloring.

Instead of attempting to increase parallelism in the backward and forward solution sweeps via reordering, level-scheduling (or wavefront) techniques exploit the intrinsic parallelism in these sweeps. Since the matrix is sparse, the sequence of tasks required in a forward sweep can be ‘topologically’ sorted in groups of tasks that can be performed in

parallel, by analyzing the dependencies between the tasks. In the simplest example when the matrix arises from the 5-point Finite Difference discretization of an elliptic PDE, then during the forward sweep the value at a given node can be updated as soon as its south and west neighbors have already been calculated. This defines a ‘schedule’ of nodes to be computed in parallel, which are all the nodes located on the same South to West line.

More generally, it can be seen that the forward solution technique is equivalent to re-ordering the original lower and upper triangular sparse matrix into a block form, in which the diagonal blocks are diagonal matrices. The wavefront technique (or level scheduling) consists of finding this reordering by a pass through the adjacency graph. The new ordering is then explicitly or implicitly exploited. This technique would work equally well for three dimensional problems as well as two dimensional problems. For further details see, for example, [1, 10] and the references therein.

### 3 Distributed ILU(0) and SOR preconditioners

In this section we will describe parallel variants of the Block Successive Over-Relaxation (BSOR) and ILU(0) preconditioners for distributed memory environments. For both ILU and SOR, we can use multicoloring or level-scheduling at the macro-level, to extract parallelism. Here, by macro-level we mean the level of parallelism corresponding to the processors, or blocks, or subdomains in the domain decomposition framework.

#### 3.1 Distributed ILU(0)

In the ILU(0) factorization [7] the LU factors have the same nonzero patterns as the original matrix  $A$ , so that in distributed mode the references of the entries belonging to the external processors in the ILU(0) factorization are identical with those of the matrix-vector product operation with the matrix  $A$ . This is not the case for the more accurate ILU( $p$ ) factorization, with  $p > 0$ . If we attempt to implement a wavefront ILU preconditioner on a distributed memory computer, we face the difficulty that the natural ordering for the original sparse problem may put an unnecessary limit on the amount of parallelism available. We can instead define a two-level ordering. We can first define a ‘global’ ordering which is a wavefront ordering for the subdomains. This is based on the graph which describes the coupling between the subdomains. Two subdomain are coupled if and only if they they contains at least a pair of coupled unknowns one from each subdomain. Within each subdomain we can then define a local ordering.

To describe the possible parallel implementations of our ILU(0) preconditioners it is sufficient to consider a local view of the distributed sparse matrix, as is illustrated in Figure 1 in an example. The problem is partitioned into  $p$  subdomains or subgraphs using some graph partitioning technique. This results in a mapping of the matrix into processors in which we assume that the  $i$ -th equation (row) and the  $i$ -th unknown are mapped to the same processor. We distinguish between *interior* points and *interface* points. The interior points are those nodes that are not coupled with nodes belonging to

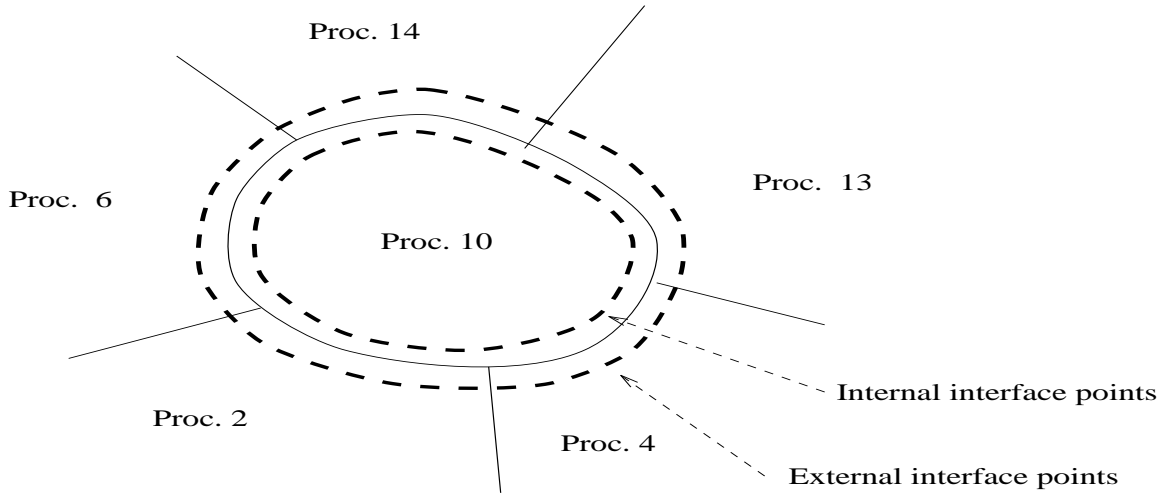


Figure 1: A distributed view of ILU(0)

other processors. Interface nodes are those local nodes that are coupled with at least one node which belongs to another processor.

Thus, processor number 10 in the figure holds a certain number of rows that are local rows. Consider the rows associated with the interior nodes. The unknowns associated with these nodes do not depend on the other variables. As a result the rows associated with these nodes, can be eliminated independently and in parallel in the ILU(0) process. The rows associated with the nodes on the interface of the subdomain will require more attention. Recall that an ILU(0) factorization is entirely determined by the order in which we process the rows. We have already hinted that we can start eliminating the interior nodes. Once this is done we can then eliminate the interface rows *in a certain order*. There are two natural choices. The first would be to impose a global order based on the labels of the processors. Thus in our illustrative Figure we will process the rows belonging to processors 2, 4 and 6 before those in processor 10. The interface rows in Processor 10 must in turn be processed before those of Processors 13 and 14. The local order, i.e., the order in which we process the interface rows in the same processor (e.g. Processor 10) may not as important. This global order based on PE-number, defines a natural priority graph and parallelism can easily be exploited in a data driven implementation.

However, it is somewhat unnatural to base the ordering just on the processor labeling. We observe that we can also define a proper order in which to perform the elimination by *replacing the PE-numbers by any labels provided any two neighboring processors have a different label*. The most natural way of doing this is to perform a multicoloring of the subdomains, and use the colors in exactly the same way we used the labels before to define an order of the tasks. The colors can be defined to be positive integers. We write the algorithms in this general form, i.e., with a label associated with each processor. Thus, the simplest valid labels are the PE numbers, which leads to the PE-label based order.

In the following we will define  $Lab_j$  as the label of Processor number  $j$ .

**ALGORITHM 3.1 Distributed ILU(0) factorization in wavefront order**

1. In each processor  $P_i$ ,  $i = 1, \dots, p$  Do:
2.     Perform the ILU(0) factorization for interior local rows;
3.     Receive the factored rows from the adjacent processors  $j$  with  $Lab_j < Lab_i$ ;
4.     Perform the ILU(0) factorization for the interface rows with pivots received from the external processors in step 3;
5.     Perform the ILU(0) factorization for the boundary nodes, with pivots from the interior rows completed in step 2;
6.     Send the completed interface rows to adjacent processors  $j$  with  $Lab_j > Lab_i$ ;
7. EndDo

Step 2 of the above algorithm can be performed in parallel as it does not depend on data from other processors.

Once this distributed ILU(0) factorization has been completed, the preconditioned Krylov subspace algorithm will require a forward and backward sweep at each step. The distributed forward/backward solution based on this factorization can be implemented as follows.

**ALGORITHM 3.2 Distributed forward and backward sweep**

1. In each processor  $P_i$ ,  $i = 1, \dots, p$  Do:
2.     Forward solve:
3.     Perform the forward solve for the interior nodes.
4.     Receive the updated values from the adjacent processors,  $j$  with  $Lab_j < Lab_i$ ;
5.     Perform the forward solve for the interface nodes;
6.     Send the updated values of boundary nodes to the adjacent processors,  $j$ , with  $Lab_j > Lab_i$ ;
7.     Backward solve:
8.     Receive the updated values from the adjacent processors,  $j$  with  $Lab_j > Lab_i$ ;
9.     Perform the backward solve for the boundary nodes;
10.     Send the updated values of boundary nodes to the adjacent processors,  $j$  with  $Lab_j < Lab_i$ ;
11.     Perform the backward solve for the interior nodes.
12. EndDo

As in the ILU(0) factorization the interior nodes do not depend on the nodes from the external processors, and can be computed in parallel in steps 2 and 11. In the forward solve, the solution of the interior nodes is followed by an exchange of data and the solution on the interface. The backward solve works in reverse in that the boundary nodes are first computed, then they are sent to adjacent processors, and finally, interior nodes are updated.

## 3.2 Block SOR and SSOR preconditioning

The same ideas as those described for ILU(0) can be exploited for SOR and SSOR. Thus, for SOR, we only have to label the nodes using multicoloring or the processor numbers as labels. Then, the distributed forward and backward sweeps for the SSOR preconditioner would be identical with that of Algorithm 3.2. The main difference between the ILU(0) and SSOR preconditioner is that the latter does not require preprocessing of the original matrix to obtain the factorization, since this is expressed in terms of the lower and upper triangular parts of  $A$ . The SOR preconditioning operation requires only a forward sweep as opposed to a forward and backward sweeps.

## 3.3 Implementation

In this section we will give some additional details on the implementation of the domain-parallel forms of the Block Successive Over-Relaxation (BSOR) and ILU(0) preconditioners described in the previous sections. We can conceptually reorder the unknowns by labeling the block of unknowns associated with processor 1, followed by the unknowns associated with processor 2, etc.. The terms blocks and processors will often be used interchangeably in what follows.

In our implementation, we use the general data structure for distributed matrix computations developed in [16]. This data-structure handles the general situation where  $A$  is irregularly structured. A key feature of the data structure is the separation of the boundary points from the interior points for efficient interprocessor communication. This is illustrated in Figure 2. In our data structure, the interface nodes are always listed last after the interior nodes. We will call this local reordering of the data the *local* ordering.

The first step that the parallel distributed iterative solver executes is to set-up all this boundary information. In particular, each processor needs to know (1) the processors with which it must communicate, (2) the list of interface points and (3) a break-up of this list into pieces of data that must be sent and received to/from the “neighboring processors”. The complete description of the data structure associated with this boundary information is as follows,

- `map` – the mapping from a node with global ordering to the processor to which it belongs.
- `nloc` – the total number of nodes in a processor.
- `nbnd` – the start of the interface nodes.
- `nproc` – the number of neighboring processors.
- `proc` – list of the `nproc` neighboring processors.
- `ix` – the list of all nodes whose values need to be exchanged with neighboring processors.

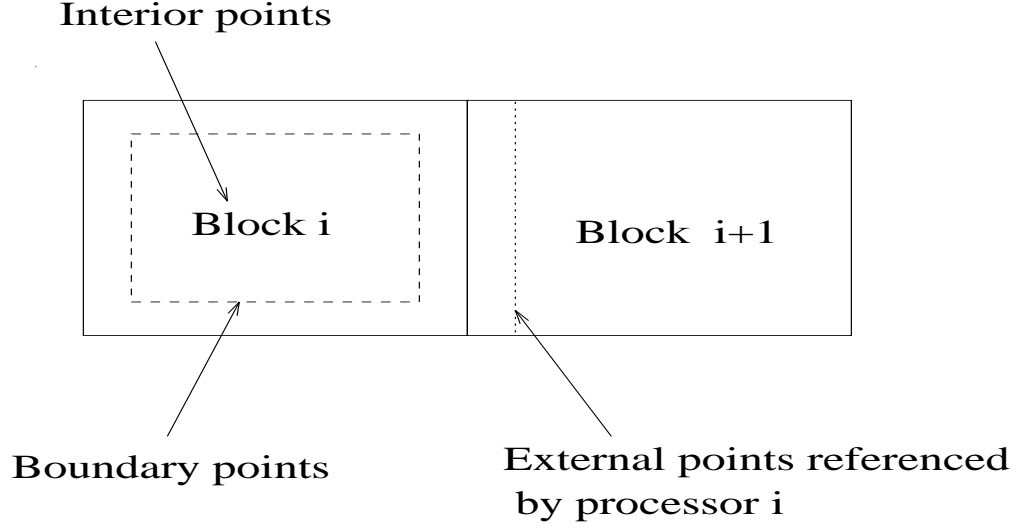


Figure 2: Interior points, boundary points and external points

- `ipr` – the pointer to the beginning of the list in array `ix` of each of `nproc` neighboring processors.

The number of boundary nodes will be  $Nbdy = nloc - nbnd + 1$ . To store the local matrix we separate it into two parts, a part which acts on the local nodes and a second which acts on the remote nodes. Currently all matrices are stored in the general row-sparse oriented format (CSR format, see [11]). The local matrix is stored using the local ordering. The first  $nloc$  rows of the  $(aloc, jaloc, ialoc)$  data structure for the CSR format reference the interior nodes only. The last  $nbdy$  rows reference interface nodes located in the neighboring processors. This allows, for example, the parallel execution of the first part in the matrix-vector product without any communication. The second part which involves interface points can be done after some exchange of information between the processors.

## 4 Numerical Experiments

We implemented the schemes described in the previous sections and compared them on a Connection Machine 5. We first give a very brief overview of the architecture of the CM-5.

## 4.1 The CM-5

The CM-5 is an MIMD massively parallel computer in which the PN's (processing nodes) are interconnected in a tree-like structure. The processing nodes are located in the leaves of the tree. The parent nodes themselves are not PNs but Data Router (DR)s. To avoid problems with low bandwidth, there is an increase of the bandwidth as we go up each level of the tree. This feature overcomes the usual communication bottleneck that often characterizes tree structures. In theory, the DN provides enough bandwidth for every Network Interface (NI) to sustain data transfer rates of 20Mbytes per second to any other NI within its group of four; 10 Mbytes to any other NI with its group of 16; and 5 Mbytes per second to any other in the system. Thus, in theory, the best to worst performance ratio is a factor of at most 4. In addition, communication of small messages which is common in sparse matrix computations, is dominated by the message startup, which tends to be quite high on many distributed memory message passing computers.

Comparing the communication speed of 5-20 Mbytes/sec with that of about five Mflops/sec of each SPARC microprocessor chip, it is apparent that communication can easily overshadow the performance of the arithmetic of a typical sparse matrix algorithm. For additional details on the architecture of the CM-5 see [2]. For details on the performance of the CM-5 for sparse matrix computations, see [9].

The numerical experiments described in this section have been performed in SPMD (Single Program Multiple Data) mode, using CMMD, the CM-5 message-passing library.

## 4.2 Test problems

We take as model problems linear systems that arise from the discretizations of elliptic partial differential equation on a square domain. Note that *the regular structure of these model problems is not exploited*, i.e., the data structures and the algorithms used are those that would be used for irregularly structured sparse matrices. In these model problems, the square is divided into  $p$  rectangle-shaped blocks, where  $p$  is the number of the available processors. The domain to processor assignment is shown in Figure 3.

Our first test problem (Problem 1) discretizes the Partial Differential Equation,

$$\begin{aligned} -(bu_x)_x - (cu_y)_y + (du)_x + du_x + (eu)_y + eu_y + fu &= g \\ \Omega &= (0, 1) \times (0, 1) \\ u &= 0 \text{ on } \partial\Omega \end{aligned} \tag{1}$$

where

$$\begin{aligned} b &= \exp(-xy), \quad c = \exp(xy), \quad d = \beta(x+y), \\ e &= \gamma(x+y), \quad f = \frac{1}{(1+xy)}, \end{aligned}$$

and  $g$  is chosen so that the exact solution is  $u = x \exp(xy) \sin(\pi x) \sin(\pi y)$ . The problem is discretized using 5-point centered finite differences. This class of problems was described by Elman [3]. Recall that the number of colors required for this problem is two.

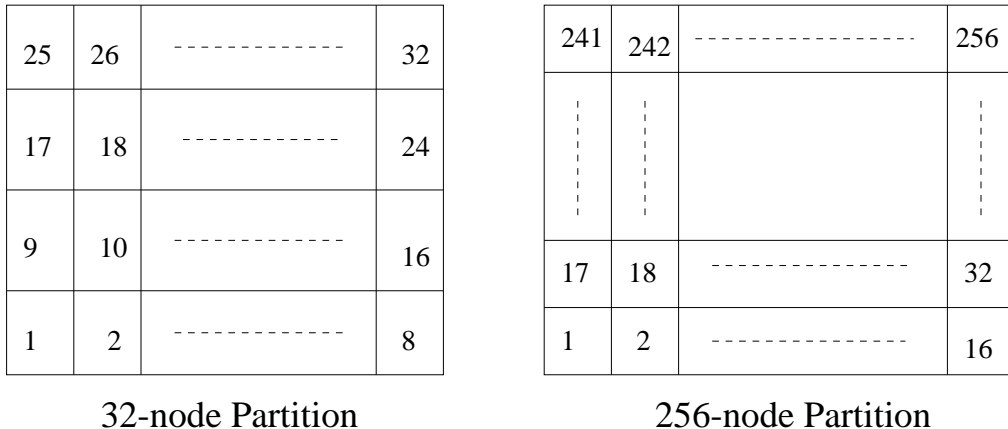


Figure 3: Mapping between a square domain and processors

Our second test problem (Problem 2) results from the discretization of the convection-diffusion equation

$$-\epsilon \Delta u + \cos \alpha u_x + \sin \alpha u_y = f, \quad (2)$$

$$\Omega = (0, 1) \times (0, 1)$$

$$u = 0 \text{ on } \partial\Omega$$

(3)

For this second problem, we used 9-point discretization of the Laplacian. The number of colors required to multicolor the graph associated with this problem is four.

### 4.3 Results

The matrices were stored in CSR (Compressed Sparse Row) format [11] while the ILU factors were stored in the MSR format. In order to avoid deadlocks the adjacency processor list was sorted in ascending order. The parameter  $\omega$  was set to 1.0 for the BSOR and point-SSOR methods. FGMRES, the flexible variant of GMRES [15, 18] was used as the outer accelerator. With this option, the preconditioner is allowed to vary at each GMRES step, as is the case when the preconditioner is defined through an iterative scheme. We set  $\gamma = 50$ ,  $\beta = 1$ ,  $\epsilon = 0.1$ ,  $\alpha = 15$ , so that the resulting matrices for problem 1 and 2 are nonsymmetric. BSOR( $l$ ) stands for the Block SOR algorithm described in the previous section, iterated  $l$  times for higher accuracy [17]. The vector units were not used for our experiments. For the BSOR method the diagonal blocks were inverted iteratively by GMRES preconditioned with  $ILUT(4)$ , preconditioning [13]. In these inner solves the solution is iterated until the initial residual is reduced by a factor of  $\epsilon = 10^{-3}$ . To obtain the multi-color ordering, we used a simple greedy heuristic described in the literature, see for example, [17].

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	8.9/22	4.9/38	4.4/44	6.9/60
MC-BSOR(2)	12.2/15	4.5/20	4.6/27	8.7/43
SSOR-Wavefront	6.1/145	5.3/227	6.2/155	8.4/150
DISTILU(0)-Wavefront	4.1/120	3.5/105	4.7/119	6.2/122
DISTILU(0)-Multicol	3.1/119	2.3/102	3.2/122	5.3/138

Table 2: Problem 1 with FDM,  $\gamma = 50, \beta = 1$  with FGMRES(10),  $N=128 \times 128$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	63.0/28	25.3/50	16.9/65	13.7/80
MC-BSOR(2)	80.8/19	33.3/35	18.4/39	16.4/58
SSOR-Wavefront	36.6/284	21.9/290	19.5/298	23.3/294
DISTILU(0)-Wavefront	25.6/221	14.7/243	13.4/258	18.0/257
DISTILU(0)-Multicol	21.1/221	9.0/245	8.7/255	11.6/261

Table 3: Problem 1 with FDM,  $\gamma = 50, \beta = 1$  with FGMRES(10),  $N=256 \times 256$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

The CPU time and the number of iterations are shown in the tables 2 - 7. Comparing the number of iterations for the wavefront technique and for those in the multi-coloring order, we notice little difference. This is in contrast with the situation for the symmetric case, e.g., in the IC(0) preconditioned CG. Hence, for the problems tested the multi-coloring ordering does seem to outperform the wavefront ordering as is seen in the tables. In all cases ILU(0) with the multi-coloring order gives the best performances.

Tables 8 - 9 show the megaflop rates and the ratio of the time spent in communication over the total CPU time. We can see that this ratio tends to increase with the number of PNs. The percentages are very high, except for the 32 node partition with ILU(0) preconditioning. Since this does not include the communication times spent in the provided library routines such as the dot-product, the actual percentages could be even higher. The maximum speed reached was 213 Megaflop on a 512 node partition. The largest size problem which we could test was for a  $512 \times 512$  mesh. We believe the Megaflop rates might improve with larger problem sizes.

Our multicoloring algorithm needs two colors for the problem 1 and four colors for the problem 2. As we compare the Megaflop rates between problem 1 and 2, we see little difference in ILU(0) but there is a slight decrease for MC-BSOR. This slowdown is expected since higher number of colors will lead to more CPU idle time. In ILU(0) this is less pronounced due to the fact that the interior nodes are basically computed in parallel, as we noted earlier.

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	549/36	106/63	50.2/77	38.7/104
MC-BSOR(2)	750/25	147/44	72.1/57	54.3/77
SSOR-Wavefront	305/538	102/543	80.1/550	77.8/552
DISTILU(0)-Wavefront	240/459	75.6/469	60.0/477	56.0/484
DISTILU(0)-Multicol	221/465	54.1/466	40.5/481	34.6/486

Table 4: Problem 1 with FDM,  $\gamma = 50, \beta = 1$  with FGMRES(10),  $N=512 \times 512$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	32.0/33	16.6/57	14.7/69	21.2/93
MC-BSOR(2)	50.0/26	16.9/29	17.8/46	21.7/52
SSOR-Wavefront	18.6/284	16.4/300	20.3/300	27.6/328
DISTILU(0)-Wavefront	7.6/146	7.1/154	9.1/166	13.3/176
DISTILU(0)-Multicol	5.2/145	4.5/160	6.3/173	9.6/174

Table 5: Problem 2 with 9-point Laplacian for the diffusion term,  $\epsilon = 0.1, \alpha = 15$  with FGMRES(10),  $N=128 \times 128$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	344.6/66	98.2/93	69.8/117	51.6/158
MC-BSOR(2)	304.1/29	113.2/55	74.2/65	52.4/87
SSOR-Wavefront	170.9/928	78.9/661	74.3/639	85.8/618
DISTILU(0)-Wavefront	34.0/233	37.2/414	39.3/440	55.5/502
DISTILU(0)-Multicol	30.3/260	20.6/406	21.0/433	33.4/513

Table 6: Problem 2 with 9-point Laplacian for the diffusion term,  $\epsilon = 0.1, \alpha = 15$  with FGMRES(10),  $N=256 \times 256$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	cpu/iter			
MC-BSOR(1)	2173/80	444/131	249/160	176/217
MC-BSOR(2)	2635/49	492/72	272/89	184/118
SSOR-Wavefront	592/818	681/2350	628/2550	471/2000
DISTILU(0)-Wavefront	406/628	135/563	238/1234	216/1184
DISTILU(0)-Multicol	330/567	132/906	137/1279	92.8/971

Table 7: Problem 2 with 9-point Laplacian for the diffusion term,  $\epsilon = 0.1, \alpha = 15$  with FGMRES(10),  $N=512 \times 512$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR and SSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	Flops/Communication			
MC-BSOR(1)	21/80%	71/82%	102/87%	86/94%
DISTILU(0)-Wavefront	28/29%	47/65%	51/75%	45/80%
DISTILU(0)-Multicol	34/15%	74/47%	78/63%	79/68%

Table 8: Problem 1 with FDM,  $\gamma = 50, \beta = 1$  with FGMRES(10),  $N=256 \times 256$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	Flops/Communication			
MC-BSOR(1)	13/80%	72/80%	141/82%	213/86%
DISTILU(0)-Wavefront	25/11%	82/76%	106/61%	114/74%
DISTILU(0)-Multicol	28/46%	144/54%	157/44%	186/58%

Table 9: Problem 1 with FDM,  $\gamma = 50, \beta = 1$  with FGMRES(10),  $N=512 \times 512$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	Flops/Communication			
MC-BSOR(1)	12/93%	41.8/94%	63/95%	56/98%
DISTILU(0)-Wavefront	26/35%	43/67%	41/77%	39/81%
DISTILU(0)-Multicol	33/20%	73/47%	74/60%	67/68%

Table 10: Problem 2 with 9-point Laplacian,  $\epsilon = 0.1, \alpha = 15$  with FGMRES(10),  $N=256 \times 256$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
	Flops/Communication			
MC-BSOR(1)	8.2/94%	42/94%	80/95%	123/96%
DISTILU(0)-Wavefront	27/15%	72/51%	91/67%	95/77%
DISTILU(0)-Multicol	30/8%	119/25%	162/43%	182/57%

Table 11: Problem 2 with 9-point Laplacian,  $\epsilon = 0.1, \alpha = 15$  with FGMRES(10),  $N=512 \times 512$ , with uniform-sized subdomains,  $\omega=1.0$  for BSOR

	nprocs = 32	nprocs = 128	nprocs = 256	nprocs = 512
N	Flops/Communication			
256x256	42/6.4%	160/27%	254/40%	353/50
512x512	45/6.5	167/12%	339/20%	638/28

Table 12: Flops for Matrix-vector Product for 5-point Matrix, with uniform-sized subdomains

## 5 Summary and conclusion

We have shown how the general ideas of multicoloring and level-scheduling can be exploited for implementing a general sparse iterative linear system solver on a distributed memory computer. The underlying data structure is that of a distributed sparse matrix, in which rows and corresponding unknowns are assigned to the same processor. This is a very general paradigm and can accommodate both regularly structured and irregularly structured sparse matrices. A coarse-level multi-color or wavefront reordering of the nearest-neighbor graph yields a parallelism at the domain or block level. This can then be exploited in a message passing environment.

For general nonsymmetric matrices, the multicolor ILU(0) seems to outperform the other preconditioners tested in this paper. We observed that communication overhead can be very high. This resulted in low overall performance for all the methods we have tried. Our experiments also show that the situation worsens as the number of processors increases. One of the reasons for this may be that the sequence of the data exchanges (swaps) in the matrix-vector products generate too many busy-waits since a processor cannot proceed to the exchange with the next adjacent processor before the receive from the current processor in the loop is completed. It would be best if all exchanges with the adjacent processors were performed in parallel by one single communication command. An efficient communication primitive of this type is sorely needed for distributed sparse matrix computations and for all domain decomposition type algorithms.

## References

- [1] E. C. Anderson and Y. Saad. Solving sparse triangular systems on parallel computers. *International Journal of High Speed Computing*, 1:73–96, 1989.
- [2] The connection machine cm-5 technical summary. Technical report, Thinking Machine Corporation, Cambridge, Massachusetts, 1992.
- [3] H. C. Elman. *Iterative Methods for Large Sparse Nonsymmetric Systems of Linear Equations*. PhD thesis, Yale University, Computer Science Dept., New Haven, CT., 1982.
- [4] C. C. Jay Kuo and T. F. Chan. Two-color Fourier analysis of iterative algorithms for elliptic problems via red/black ordering. *SIAM J. Scient. Stat. Comput.*, 11:767–793, 1990.
- [5] M. T. Jones and P. E. Plassmann. Parallel iterative solution of sparse linear systems using ordering from graph coloring heuristics. Technical Report MCS-P198-1290, Argonne National Lab., Argonne, IL, 1990.
- [6] R. Leuze. Independent set orderings for parallel matrix factorizations by Gaussian elimination. *Parallel Computing*, 10:177–191, 1989.

- [7] 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.
- [8] D. O’Leary. Ordering schemes for parallel processing of certain mesh problems. *SIAM J. Sci. Statist. Comput.*, 5:620–632, 1984.
- [9] S. Petiton, Y. Saad, K. Wu, and W. Ferng. Basic sparse matrix computations on the cm-5. *Internat. J. of Modern Physics*, 4:65–83, 1993.
- [10] Y. Saad. Krylov subspace methods on supercomputers. *SIAM J. Scient. Stat. Comput.*, 10:1200–1232, 1989.
- [11] 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.
- [12] Y. Saad. ILUM: a parallel multi-elimination ILU preconditioner for general sparse matrices. Technical Report 92-241, University of Minnesota, Army High Performance Computing Research Center, Minneapolis, Minnesota, 1992. submitted, under revision.
- [13] Y. Saad. ILUT: a dual threshold incomplete ILU factorization. Technical Report 92-38, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, 1992. to appear, SISSC.
- [14] Y. Saad. Krylov subspace methods in distributed computing environments. Technical Report 92-126, Army High Performance Computing Research Center, Minneapolis, MN, 1992.
- [15] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Stat. Comput.*, 14:461–469, 1993.
- [16] Y. Saad. Data structures and algorithms for domain decomposition and distributed sparse matrix computations. Technical Report 94—, Army High Performance Computing Research Center, Minneapolis, MN, 1994.
- [17] Y. Saad. Highly parallel preconditioners for general sparse matrices. In G. Golub, M. Luskin, and A. Greenbaum, editors, *Recent Advances in Iterative Methods, IMA volumes in Mathematics and its Applications*, volume 60, pages 165–199. Springer Verlag, 1994.
- [18] 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.