Enhanced Parallel Multicolor Preconditioning Techniques for Linear Systems*

Yousef Saad† Maria Sosonkina‡

Abstract

When solving a linear system in parallel, a large overhead in using an incomplete LU factorization as a preconditioner may annihilate any gains made from the improved convergence. This overhead is due to the inherently sequential nature of such a preconditioning. Multicoloring of the subdomains assigned to processors is a common remedy for increasing the parallelism of a global ordering. However, the achieved degree of parallelism is still limited since different colors must be processed sequentially. Further reductions of the parallel overhead are possible. Here we suggest several strategies to decrease the idle time in the multicolor block Gauss-Seidel preconditioning.

1 Introduction

Parallel preconditioners may be developed in two distinct ways: extracting parallelism from efficient sequential techniques or designing a preconditioner from the start specifically for parallel platforms. Preconditioners developed from the first approach yield the same good convergence properties as those of a sequential method but often have a low degree of parallelism, leading to inefficient parallel implementations. In contrast, preconditioners developed from the second approach enjoy a higher degree of parallelism, but may have inferior convergence properties. This approach is based on the ideas from Domain Decomposition methods and considers distributed sparse linear systems as distributed objects. For discussions of this viewpoint, see [19, 7, 16] and references therein. A substantial progress has been made in extracting parallelism from inherently sequential preconditioning techniques using various multicoloring strategies (see, e.g., [8] and [10]). Multicoloring of subdomains assigned to processors is a way to maximize the parallelism of a global ordering. This paper focuses on reducing the parallel overhead incurred by the sequential execution remaining in the multicolor processing order of subdomains. First, we describe a strategy of representing *several* colors in *each* processor resulting in two-level multicolor preconditioner. Second, block iterative methods are considered. Using these methods also reduces the overhead of multicoloring when different colors are assinged to the local vectors in a block-vector. Thus for both strategies, the degree of parallelism for the multicolor preconditioning reaches that of the preconditioning based on the Domain Decomposition ideas. We show how these two strategies can be applied to the block Gauss-Seidel preconditioning.

*Work supported by NSF under grant CCR-9618827, and in part by the Minnesota Supercomputer Institute.

†Department of Computer Science and Engineering, University of Minnesota, 200 Union Street S.E., Minneapolis, MN 55455, saad@cs.umn.edu.

‡Department of Computer Science, University of Minnesota at Duluth, 320 Heller Hall, 10 University Drive, Duluth, MN 55812, masha@d.umn.edu.

The paper is organized as follows. Section 2 gives general definitions for solving a distributed linear system as a distributed object and describes the standard multicolor block SOR preconditioner In Section 3, a way to represent several colors in each processor is considered and the resulting Gauss-Seidel preconditioning is constructed. A combination of block iterative techniques and multicolor Gauss-Seidel is shown in Section 4. Section 5 contains numerical experiments followed by the conclusions section.

2 Standard multicolor block SOR preconditioner for distributed linear systems

Consider a linear system $Ax = b$, where A is a large sparse nonsymmetric real matrix of size n. To solve such a system on a distributed memory computer a graph partitioner is first invoked to partition the adjacency graph of A. Based on the resulting partitioning, the data is distributed to processors such that pairs of equations-unknowns are assigned to the same processor. Thus, each processor holds a set of equations (rows of the linear system) and vector components associated with these rows.

Figure 1 shows a "physical domain" viewpoint of a sparse linear system. This representation borrows from the Domain Decomposition literature. Thus term "subdomain" is often used here instead of the more proper term "subgraph". Note that the concepts of a "subdomain" and a "point" are defined algebraically and do not necessarily have direct geometrical representations. Each point (node) belonging to a subdomain is actually a pair representing an equation and an associated unknown. It is common to distinguish between three types of unknowns: (1) Interior unknowns that are coupled only with local equations; (2) Local interface unknowns that are coupled with both non-local (external) and local equations; and (3) External interface unknowns that belong to other subdomains and are coupled with local equations [13, 15, 7, 19, 6, 18, 9].

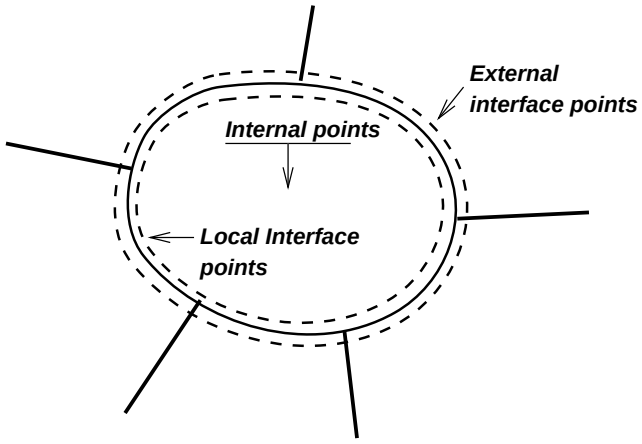

FIG. 1. *A local view of a distributed sparse matrix.*

The matrix assigned to a certain processor i, $(i = 1, \ldots, p)$ is split into two parts: the *local* matrix A_i, which acts on the local variables and an *interface matrix* X_i, which acts on the external variables. Accordingly, the local equations can be written as follows:

$$A_i x_i + X_i y_{i,ext} = b_i. \tag{1}$$

where x_i represents the vector of local unknowns, $y_{i,ext}$ is the vector of the external interface

variables, and b_i is the local part of the right-hand side vector. It is common to reorder the local equations in such a way that the interface points are listed last after the interior points. The local vector of unknowns x_i is split into two parts: the subvector u_i of internal vector components followed by the subvector y_i of local interface vector components. The right-hand side b_i is conformally split into the subvectors f_i and g_i. In accordance with these splittings, the local equations (1) can be written as follows:

$$\begin{pmatrix} B_i & F_i \\ E_i & C_i \end{pmatrix} \begin{pmatrix} u_i \\ y_i \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j \in N_i} E_{ij} y_j \end{pmatrix} = \begin{pmatrix} f_i \\ g_i \end{pmatrix}. \tag{2}$$

Here N_i is the set of indices for subdomains that are neighbors to the subdomain i. The term $E_{ij}y_j$ is a part of the product $X_i y_{i,ext}$ which reflects the contribution to the local equation from the neighboring subdomain j. The result of the multiplication by X_i affects only the local interface unknowns, which is indicated by a zero in the top part of the second term of the left-hand side of (2).

The most important operation when solving distributed sparse linear systems is undoubtedly the preconditioning operation. The simplest Domain Decomposition-based preconditioner is the additive Schwarz procedure, which is a form of the block Jacobi iteration, where the blocks refer to matrices associated with entire subdomains. Another preconditioner, block Gauss-Seidel, may be derived from the multiplicative Schwarz procedure. A block Gauss-Seidel iteration can be carried out as a sequence of eliminations of the local residual components followed by the updates of the local components of the unknown and of the global residual vector. Thus, a global order in which to perform these eliminations as well as some global stopping criterion are required

A global ordering can be based on an arbitrary labeling of the subdomains provided it is consistent, i.e., two neighboring subdomains have a different label. The most common global ordering is multicoloring of the subdomains, which increases parallelism [2, 4, 3, 12, 17]. Thus, if the subdomains are colored and the global ordering of the subdomains defined by the colors, the Gauss-Seidel iteration is performed in each processor as follows:

ALGORITHM 2.1. *Multicolor Block Gauss-Seidel Iteration*

1. *Do* $col = 1, \ldots, numcols$
2. *If* $(col.eq.mycol)$ *Then*
3. *Obtain external data* $y_{i,ext}$
4. *Update local residual* $r_i = (b - Ax)_i = b_i - A_i x_i - X_i y_{i,ext}$
5. *Solve* $A_i \delta_i = r_i$
6. *Update solution* $x_i = x_i + \delta_i$
7. *EndIf*
8. *EndDo.*

Many variations are possible such as overlapping of the domains, inaccurate solves in step 5, inclusion of the relaxation parameter ω.

3 Variants of two-level multicolor block SOR preconditioner

Although the multicoloring of subdomains is a way to extract parallelism from the preconditioning, for large subdomains, however, sequential processing still takes a considerable amount of time in multicolor preconditioners. Thus it is desirable to reduce the size of color subdomains. A general approach is to split each subdomain into several parts: those that can be processed in parallel and those that require a sequential processing order. For

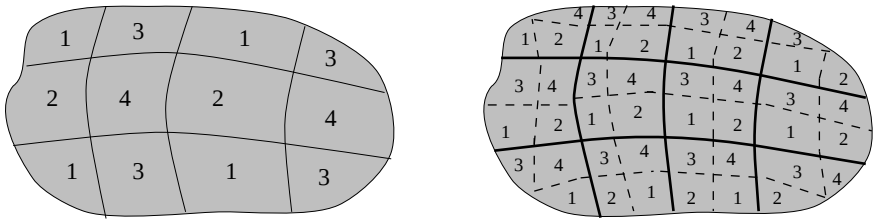

FIG. 2. *Coloring of subdomains (left), 2-level coloring of subdomains (right).*

example, the interior nodes in all subdomains can be processed in parallel followed by a solve with the interface nodes in each subdomain [10], such that the order of the solves is given by multicoloring of subdomains. However, the reduction in the idle time comes at expense of sacrificing the quality of preconditioning (see [10] for more details). By using a level-set expansion technique more advantageous splitting into parts can be achieved in a subdomain. Then all the parts are colored consistently (Figure 2). To minimize the idle time due to ordering, each subdomain should contain the same number of colors. Alternatively, all the colors of subdomains should be represented in each subdomain.

A variation of this approach to splitting is to represent first each part of a subdomain by a node in a graph, define consistent coloring of this graph, and then select color parts by grouping certain nodes for each color. To define a consistent coloring (we call it "interface-coloring"), assume that the local points interfaced to each neighbor are separated into different color parts. Based on this assumption we can create a quotient graph, each node of which represents a set of the local interface points sent to a certain neighbor. Then this graph is colored with any graph coloring technique (see, e.g., [11, 14]). A consistent coloring can now be obtained if the local interface points that are sent to more than one processor (we call such points "multi-interface points") undergo a special treatment. Once the quotient graph is colored, a single separate color is found for *all* the multi-interface points in a subdomain. Note that in the quotient graph, multi-interface points are represented by more than one node. To assign colors to the interior points a level-set technique can be used, such that initially each subdomain has only its local interface points colored.

Load balancing among color parts is extremely important for a subdomain splitting since imbalanced computations could cause a performance degradation, as will be indicated in the numerical experiments. In particular, the interface-color splitting lacks load balancing when the number of multi-interface points is small compared with the average size of a color part. This happens for a rectangular partitioning of regular grids, for example. One way to improve load balancing is to color certain interior points in the color of the multi-interface points. Such points can be taken as "center" points in each subdomain. Let these "center" points be the points of the last level-set of each color part. A local view of the color parts in a subdomain is shown in Figure 3. An additional benefit of this choice is that these interior points together with the multi-interface points may be used as a coarse grid solver for a multilevel preconditioner [1].

A splitting of the subdomain i into color parts defines a symmetric permutation matrix P_i for the local variables, such that $P_i A_i P_i = A_i^c + F_i^c$, where the matrix A_i^c is a block diagonal matrix with the number of blocks equal to the number $numcols$ of colors in the subdomain i. Therefore local equations (1) can be rewritten as

$$A_i^c x_i^c + F_i^c x_i^c + \psi_i^c = b_i^c,$$

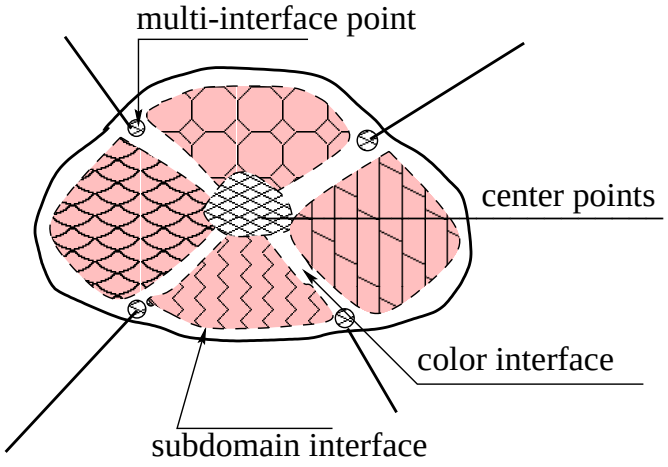

FIG. 3. *Local view of color parts in each subdomain.*

where vectors x_i^c, ψ_i^c, and b_i^c are obtained by applying the permutation P_i to vectors x_i, $(0,\ldots,0,(X_i y_{i,ext})^T)^T$, and b_i, respectively. Let $R_{i_{col}}$ be a restriction operator from the entire subdomain i into a color part col, $(col = 1,\ldots,numcols)$. Then for a permuted vector z_i^c, $R_{i_{col}} z_i^c \equiv z_i^{col}$ belongs to the color part col, and for any matrix M_i^c, $R_{i_{col}} M_i^c R_{i_{col}}^T \equiv M_i^{col}$. An interface-color multicolor SOR algorithm follows.

ALGORITHM 3.1. *Interface-color Block Gauss-Seidel Iteration*

1. *Do* $col = 1,\ldots,numcols$
3. *Obtain external data* $y_{i,ext}$
4. *Update residual* $r_i^{col} = R_{i_{col}} P_i (b - Ax)_i = b_i^{col} - R_{i_{col}} (P_i A_i P_i) x_i^c - \psi_i^{col}$
5. *Solve* $A_i^{col} \delta_i^{col} = r_i^{col}$
6. *Update solution* $x_i^{col} = x_i^{col} + \delta_i^{col}$
7. *EndDo.*

Note that $R_{i_{col}}(P_i A_i P_i) x_i^c = A_i^{col} x_i^{col} + R_{i_{col}} F_i^c x_i^c$ in line 4. Thus, compared with Algorithm 2.1, one extra matrix-vector multiply with a part of the matrix F_i^c has to be performed to compute the residual components involving unknowns of the color part col.

4 Block Krylov accelerators and multicolor block SOR preconditioner

Another way to break the sequential color loop is to use a block version of a Krylov accelerator such that each local vector of the block is associated with a different color. The implementational details will reveal that a flexible version of a block accelerator is needed because at each iteration a different preconditioner is applied to each vector in a block.

In the block Krylov subspace methods the subspace of approximants is defined from a set (block) of initial vectors. For example, given s initial vectors

$$V = [v^1, v^2, v^3, \ldots, v^s] ,$$

the m-th block-Krylov subspace is defined as

$$\text{span}\ \{V, AV, \ldots, A^{m-1}V\} . \tag{3}$$

Thus the dimension of the subspace is $m \times s$, where s is the dimension of the block. A block-Arnoldi generalization of the Arnoldi process can be easily derived, and a block GMRES algorithm can be defined from it (see, e.g., [5] for details). Without describing the algorithm completely, we can show its main matrix operations – the matrix-vector products and the preconditioning application. In a right-preconditioned GMRES algorithm, these two operations take the form:

ALGORITHM 4.1. *Block Arnoldi*

1. *Do* $j = s, \ldots, m$
2. *Solve* $Mw_k = v_{k-s}$ *for* $k = j+1, \ldots, j+s$
3. *EndDo*
4. *Do* $j = s, \ldots, m$
5. $w_k := Aw_{k-s}$ *for* $k = j+1, \ldots, j+s$
6. *EndDo.*

Then the vectors w_k are orthogonalized against all the previous $v_1, \ldots, v_{k-1}$ vectors by a modified Gram-Schmidt algorithm, for example.

How to exploit parallelism in matrix-vector multiplies (lines 4–6) is well known, whereas the preconditioning step (lines 1–3) requires more attention. Note that a single preconditioning matrix M acts on all the columns of the block $V[j+1 : j+s]$. If M is associated with a standard multicolor Gauss-Seidel operation, a straightforward use of a block accelerator is not beneficial. However, it is possible to apply multicolor SOR *differently* to each column of the block: Establish a different processing order of the local subvectors for each vector in the block. In particular, at a color iteration *col*, processor i performs SOR operations (updates and a solve) on the local vector $x_i^j, j \in \{1, \ldots, s\}$. The column j (among $1, \ldots, s$) on which a given processor i will work next is determined as $j := mod(mycol + col, s) + 1$ where *mycol* is the color of processor i. Note that the block size s should be equal to the numbers of colors *numcols*.

ALGORITHM 4.2. *Multicolumn Block Gauss-Seidel Iteration*

1. *Do* $col = 1, \ldots, s$
2. *Select column:* $j := mod(mycol + col, s) + 1$
3. *Obtain external data* $y_{i,ext}^j$
4. *Update local residual*
   $r_i^j = (b - Ax^j)_i = b_i^j - A_i x_i^j - X_i y_{i,ext}^j$
5. *Solve* $A_i \delta_i^j = r_i^j$
6. *Update solution* $x_i^j = x_i^j + \delta_i^j$
7. *EndDo.*

An illustration of the computational order of columns is provided in Figure 4. In the illustration, four subdomains are distributed among processors P1,...,P4 and colored into four colors corresponding to the processor numbers. Thus the size of the block-vector is also 4. The sequence of numbers in each sub-block represents the order of columns handled in a given processor. For example, processor P4 starts with (its part of) the second column followed by the processing of columns 3, 4, and 1, respectively. Because of the different processing order, the preconditioner application changes from column to column of the block-vector and the idea of flexible GMRES [14] can be used to cope with this. We have implemented a flexible-preconditioning capability in the block GMRES algorithm described in [5] following the way in which flexible GMRES is constructed from a standard GMRES implementation.

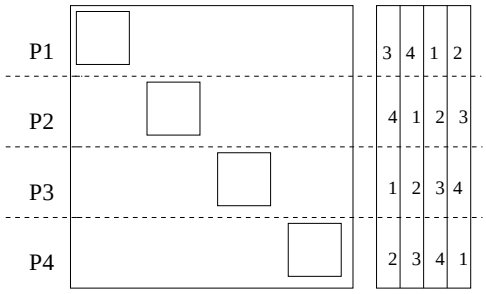

FIG. 4. *Multicolumn SOR iteration.*

5 Numerical experiments

We consider a linear system which arises from a 2-dimensional regular mesh problem. Specifically, consider the elliptic equation:

$$-\Delta u + 100e^{xy}\frac{\partial u}{\partial x} + 100e^{-xy}\frac{\partial u}{\partial y} - 100u = f$$

on the square $(0,1)^2$ with Dirichlet boundary conditions. The shift term $-100u$ makes the problem indefinite. A 9-point centered-differences scheme has been used to obtain the derivatives. When discretized such that there are 360 interior mesh points in each direction, the resulting linear system is of size $n = 360^2 = 129,600$. A flexible variant of restarted GMRES (FGMRES) [14] with a subspace dimension of 10 has been used to solve the problem to reduce the residual norm by 10^6. In Algorithm 4.2 (multicolumn SOR), we have used a flexible variant of block GMRES(10×4), in which 4 is the block size and 10 is the number of restart block-vectors.

The preconditioning phase performs 5 iterations of SOR. In each SOR iteration, a backward-forward substitution with the ILUT factors has been used. For ILUT, the fill-in and tolerance parameters were taken as 15 and 10^{-4}, respectively. Since a 9-point stencil is used, the minimum number of subdomain colors is four and one additional color is assigned to the multi-interface points in interface-color SOR. Figure 5 compares the solution times and number of matrix-vector multiplications for the three variants of the SOR preconditioner. The gains of splitting subdomains into color parts are especially pronounced for the smaller processor numbers when the subdomain size is large. However, if the load is imbalanced in the interface-color SOR preconditioner, e.g., one of the colors is assigned to the multi-interface points only, then the time *per iteration* grows when the number of processors increases (Table 1). In Table 1, the load-imbalanced version of interface-color SOR is labeled "no centers". For multicolumn SOR, the workload is larger than for the other SOR variants (cf. the number of matrix-vector multiplications in Figure 5 – right). However, the performance of multicolumn SOR is superior if the linear systems with multiple right-hand sides are to be solved.

In the following experiments we test scalability of the preconditioners: by increasing the total problem size with the increase in processor numbers. One way to increase the total problem size is to keep the size of a subdomain constant. For example, if we fix a 50×50 mesh size *per processor*, then on 4 processors, the total problem size is $50 \times 50 \times 4 = 10,000$, whereas on 100 processors the total problem size is $50 \times 50 \times 100 = 250,000$. The solution parameters are the same as for the 360×360 mesh problem. Figure 6 shows the solution times and numbers of matrix-vector multiplications for the three multicolor SOR variations.

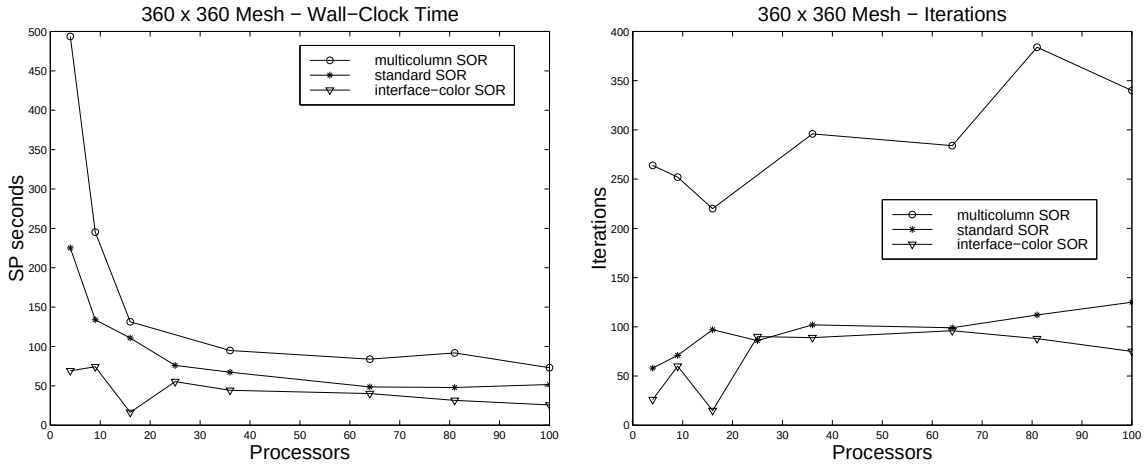

FIG. 5. *Solution times and iterations for solving a* 360×360 *problem with 3 variations of multicolor SOR.*

Keeping subproblem sizes fixed while increasing the number of processors increases the overall size of the problem making it harder to solve and thus increasing the solution time.

TABLE 1
Time to perform an iteration for several variations of SOR.

	Time per iteration					
Processors	4	9	16	36	81	100
Standard	3.88	1.89	1.14	0.66	0.43	0.41
Interface-color	2.66	1.24	1.02	0.50	0.36	0.34
Interface-color (no centers)	3.69	1.49	0.35	0.48	0.53	0.64

Additional tests were performed with the multicolumn and standard SOR preconditioners on the problem $\Delta u = f$ of the same size (50×50 mesh *per processor*) and with the same boundary conditions using the same solution parameters for flexible GMRES(20). Five-point centered differences were used to approximate derivatives. Thus only two colors appear in the domain coloring and the subspace size of block GMRES is 20×2, in which 2 is the block size. Table 2 shows the wall-clock timing results for the preconditioner application, the whole iterative solution (*prec.* and *total sec.*, respectively), and the iteration numbers (*iter.*) on 4, 9, 16, and 25 processors of a CRAY T3E. The preconditioning application and overall solution times for standard SOR are larger than the corresponding times for the multicolumn SOR. The extra storage and computational cost for multicolumn SOR are compensated by the reduced parallel overhead when a block of colors is used in conjunction with a block of vectors and by the enhanced convergence properties of the block Krylov subspace methods.

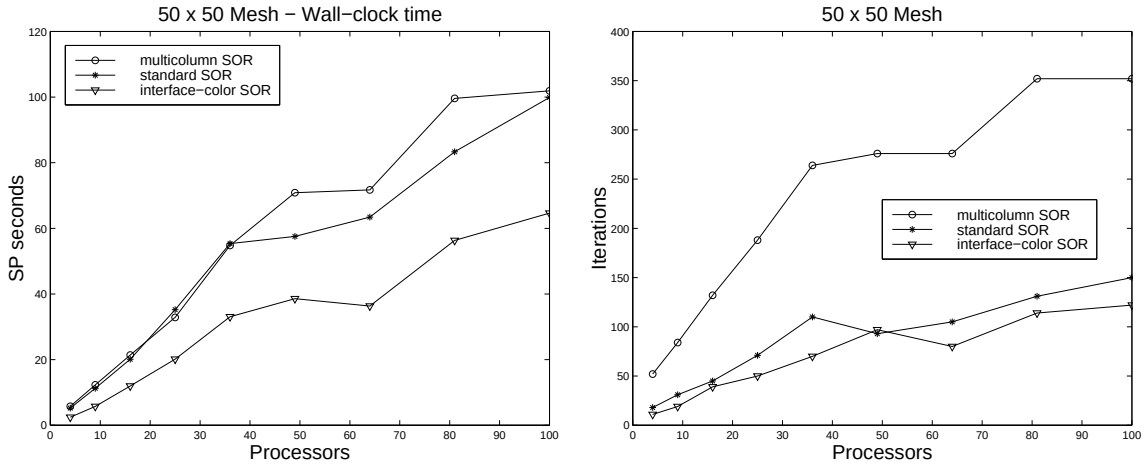

FIG. 6. *Solution times and iterations for solving a problem with* 50×50 *subdomain sizes with 3 variations of multicolor SOR.*

TABLE 2
Multicolumn vs standard SOR for solving a problem with 50×50 *subdomain sizes.*

	Multicolumn			Standard		
Processors	prec.	total sec.	iter.	prec.	total sec.	iter.
4	1.04	1.15	30	1.08	1.26	18
9	1.33	1.50	36	1.53	1.82	25
16	1.80	2.05	50	2.02	2.40	32
25	1.94	2.23	54	3.02	3.58	45

6 Conclusions

This paper addresses the reduction of the parallel overhead incurred in a standard multicolor SOR preconditioner. Two main approaches have been proposed: (1) to represent several colors within each subdomain and (2) to use a block Krylov accelerator such that, in a multicolor SOR iteration, each local column of the block-vector is associated with a different color. Both approaches appear to be beneficial when the local problem size is large, i.e., when the sequential computation within a multicolor SOR iteration takes a considerable amount of time. For the first approach to be effective, a good load balancing should be achieved among the color parts within each subdomain. Having an equal number of colors per subdomain is one way to balance the workload. Adding a few center points to the color part consisting of the points coupled with more than one neighbor subdomains not only improves the load balancing but also has a potential of being used in a coarse grid solver. When multiple linear systems are to be solved, combining a block Krylov accelerator and multicolor SOR is especially advantageous as in approach 2. In addition, enhanced convergence properties of the block methods can decrease an overall solution time despite their amount of work per iteration being larger than that of point methods.

References

[1] X.-C. Cai, *An optimal two-level overlapping domain decomposition method for elliptic problems in two and three dimensions*, SIAM Journal on Scientific and Statistical Computing, 14 (1993), pp. 239–247.

[2] X.-C. Cai, W. D. Gropp, and D. E. Keyes, *A comparison of some domain decomposition and ILU preconditioned iterative methods for nonsymmetric elliptic problems*, J. Numer. Lin. Alg. Appl., (1993). To appear.

[3] X. C. Cai and Y. Saad, *Overlapping domain decomposition algorithms for general sparse matrices*, Numerical Linear Algebra with Applications, 3 (1996), pp. 221–237.

[4] X. C. Cai and O. Widlund, *Multiplicative Schwarz algorithms for some nonsymmetric and indefinite problems*, SIAM Journal on Numerical Analysis, 30 (1993).

[5] A. Chapman and Y. Saad, *Deflated and augmented Krylov subspace techniques*, Numerical Linear Algebra with Applications, 4 (1997), pp. 43–66.

[6] V. Eijkhout and T. Chan, *ParPre a parallel preconditioners package, reference manual for version 2.0.17*, Tech. Rep. CAM Report 97-24, UCLA, 1997.

[7] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro, *Aztec user's guide. version 1.0*, Tech. Rep. SAND95-1559, Sandia National Laboratories, Albuquerque, NM, 1995.

[8] M. T. Jones and P. E. Plassmann, *Parallel iterative solution of sparse linear systems using ordering from graph coloring heuristics*, Parallel Computing, 20 (1994), pp. 753–773.

[9] ——, *BlockSolve95 users manual: Scalable library software for the solution of sparse linear systems*, Tech. Rep. ANL-95/48, Argonne National Lab., Argonne, IL., 1995.

[10] G.-C. Lo and Y. Saad, *Iterative solution of general sparse linear systems on clusters of workstations*, Tech. Rep. UMSI 96/117 & UM-IBM 96/24, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1996.

[11] J. M. Ortega and R. G. Voigt, *Solution of partial differential equations on vector and parallel computers*, SIAM Review, 27 (1985), pp. 149–240.

[12] ——, *Highly parallel preconditioners for general sparse matrices*, in Recent Advances in Iterative Methods, IMA Volumes in Mathematics and Its Applications, G. Golub, M. Luskin, and A. Greenbaum, eds., vol. 60, New York, 1994, Springer Verlag, pp. 165–199.

[13] ——, *Parallel sparse matrix library (P_SPARSLIB): The iterative solvers module*, in Advances in Numerical Methods for Large Sparse Sets of Linear Equations, Number 10, Matrix Analysis and Parallel Computing, PCG 94, Keio University, Yokohama, Japan, 1994, pp. 263–276.

[14] ——, *Iterative Methods for Sparse Linear Systems*, PWS publishing, New York, 1996.

[15] Y. Saad and A. Malevsky, *PSPARSLIB: A portable library of distributed memory sparse iterative solvers*, in Proceedings of Parallel Computing Technologies (PaCT-95), 3-rd international conference, St. Petersburg, Russia, Sept. 1995, V. E. M. et al., ed., 1995.

[16] Y. Saad and M. Sosonkina, *Distributed Schur complement techniques for general sparse linear systems*, Tech. Rep. UMSI 97/159, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 1997. Submitted, Revised.

[17] J. N. Shadid and R. S. Tuminaro, *A comparison of preconditioned nonsymmetric krylov methods on a large-scale mimd machine*, SIAM J. Sci Comput., 15 (1994), pp. 440–449.

[18] B. Smith, P. Bjørstad, and W. Gropp, *Domain decomposition: Parallel multilevel methods for elliptic partial differential equations*, Cambridge University Press, New-York, NY, 1996.

[19] B. Smith, W. D. Gropp, and L. C. McInnes, *PETSc 2.0 user's manual*, Tech. Rep. ANL-95/11, Argonne National Laboratory, Argonne, IL, July 1995.