SIAM J. SCI. COMPUT.
Vol. 20, No. 6, pp. 2261–2281

NEW PARALLEL SOR METHOD BY DOMAIN PARTITIONING*

DEXUAN XIE† AND LOYCE ADAMS‡

Abstract. In this paper we propose and analyze a new parallel SOR method, the PSOR method, formulated by using domain partitioning and interprocessor data communication techniques. We prove that the PSOR method has the same asymptotic rate of convergence as the Red/Black (R/B) SOR method for the five-point stencil on both strip and block partitions, and as the four-color (R/B/G/O) SOR method for the nine-point stencil on strip partitions. We also demonstrate the parallel performance of the PSOR method on four different MIMD multiprocessors (a KSR1, an Intel Delta, a Paragon, and an IBM SP2). Finally, we compare the parallel performance of PSOR, R/B SOR, and R/B/G/O SOR. Numerical results on the Paragon indicate that PSOR is more efficient than R/B SOR and R/B/G/O SOR in both computation and interprocessor data communication.

Key words. parallel computing, SOR, multicolor SOR, JSOR, PSOR, convergence analysis, nonmigratory permutation

AMS subject classifications. Primary, 65Y05; Secondary, 65F10

PII. S1064827597303370

1. Introduction.

The successive overrelaxation (SOR) iterative method is an important solver for a class of large linear systems [36, 10, 11, 22]. It is also a robust smoother as well as an efficient solver of the coarsest grid equations in the multigrid method [12, 26, 29, 34, 35]; however, the SOR method is essentially sequential in its original form. With the increasing use of parallel computers, several parallel versions of the SOR method have been proposed and studied.

Defined by using the multicolor ordering technique, the multicolor SOR method is a widely used parallel version of SOR and has been studied by many authors (such as Adams and Ortega [1], Adams and Jordan [2], and Adams, LeVeque, and Young [3]). Since the multicolor SOR method is parallel only within the same color, the two-color SOR method (usually referred to as the red/black SOR method) is preferred; but for some complicated problems more than two colors are required to define a multicolor ordering. Block, Frommer, and Mayer [6] proposed a general block multicolor SOR method. Harrar [13] and Melhem and Ramarao [16] studied how to quickly verify and generate a multicoloring ordering according to the given structure of a matrix or a grid. Moreover, Yavneh [35] showed that the red/black SOR method is a more efficient smoother than the sequential SOR method.

With multisplitting [14], a generalization of regular splittings introduced by Varga [28], White [30, 31, 32] proposed and analyzed the multisplitting SOR method, showing that it can be convergent for M-matrices and can be implemented in parallel on multiprocessor computers. That paper [18], together with a later paper [8], showed

*Received by the editors June 13, 1997; accepted for publication (in revised form) May 29, 1998; published electronically July 22, 1999.

http://www.siam.org/journals/sisc/20-6/30337.html

†Courant Institute of Mathematical Sciences, New York University and Howard Hughes Medical Institute, 251 Mercer Street, New York, NY 10012 (dexuan@cims.nyu.edu). The work of this author was supported in part by the National Science Foundation through award DMS-9105437 and ASC-9318159.

‡Department of Applied Mathematics, University of Washington, Seattle, WA 98195 (adams@amath.washington.edu). The work of this author was supported by NSF grant DMS-96226645 and DOE grant DE-FG03-96ER25292.

that the asymptotic rate of convergence of multisplittings is typically less satisfactory than that of standard serial stationary iterations.

Some other techniques such as the pipelining of computation and communication and an optimal schedule of a feasible number of processors are also studied and applied to define parallel versions of SOR for banded or dense matrix problems [7, 17, 19, 25]. These techniques can isolate the parts of SOR that can be implemented in parallel without changing the sequential SOR method. Two parallel SOR methods for particular parallel computers can also found in [9, 23].

On today's MIMD machines [5, 24], the time required to update one value at a grid point is very small compared to the time required to communicate it between processors. To reduce the amount of interprocessor data communication, domain decomposition techniques are widely used in the implementation of a parallel algorithm on a MIMD computer. Obviously, it is attractive to define a parallel SOR method based on domain decomposition because the resulting method can be efficiently implemented on MIMD machines and can easily deal with boundary value problems arising in science and engineering that are posed on complicated regions.

A simple parallel SOR method by domain decomposition can be defined as follows [34]: Consider a parallel implementation of the Jacobi method (a completely parallel algorithm) on p processors based on p subgrids. Here each subgrid is mapped to one processor, and contains sufficient grid points to avoid interprocessor data communication overhead. Clearly, iterates defined on each subgrid are allowed to be calculated sequentially because they are in one processor. Hence, to improve the convergence rate of the Jacobi method, Jacobi iterates defined on each subgrid can be replaced by the corresponding sequential SOR iterates, resulting in a simple parallel SOR method by domain decomposition. This is called the JSOR method in [34] because it merges Jacobi with SOR. The JSOR analysis in [34] shows that JSOR has a faster convergence rate than Jacobi but a much slower convergence rate than the sequential SOR method.

In this paper, with a novel use of interprocessor data communication techniques, we modify JSOR into a new parallel version of SOR by domain partitioning and refer to it as the PSOR method. We then show that PSOR is just the SOR method applied to a reordered linear system; hence, the SOR theory can be applied to the analysis of PSOR. In particular, we prove that the PSOR method has the same asymptotic rate of convergence as the red/black SOR (R/B SOR) method for the five-point stencil on strip and block partitions and as the R/B/G/O SOR method [3] for nine-point stencil strip partitions. We then demonstrate the parallel performance of the PSOR method on four different message-passing multiprocessors (a KSR1, the Intel Delta, a Paragon, and an IBM SP2) for solving a Poisson model problem. Numerical results show that PSOR is an efficient parallel version of SOR by domain decomposition.

Since the multicolor SOR method is a widely used parallel version of SOR, we compare the parallel performance of PSOR versus R/B SOR for solving the five-point stencil of the Poisson model problem and the R/B/G/O SOR method for solving the nine-point stencil on the Paragon. Numerical results point to the effectiveness of PSOR in both computation and interprocessor data communication. Since the multicolor SOR method is also usually implemented on parallel computers based on a domain partition, each m-color SOR iteration needs to communicate the "boundary values" between processors m times (one color each time). For strip partitions, for example, the first and last row of each strip will contain each of the m colors, so about $2m$ messages will need to be sent. In contrast, PSOR defined on a strip partition needs to communicate only twice during each iteration to neighboring processors

(once to the south processor and once to the north processor). Hence, PSOR takes less interprocessor data communication time than the multicolor SOR method.

Being defined on a domain partitioning, PSOR can be more easily applied to solving complicated problems (such as irregular geometries, high orders of discretization, and local grid refinement) than the multicolor SOR method. For such complicated scientific problems, there are graph heuristic algorithms for generating a global multicolor ordering, but determining a proper decoupling of each partition perimeter as required by PSOR may be an easier task. We also note that the preconditioner defined by the symmetric SOR (SSOR) method [12, 36] with red/black and multicolor orderings may seriously degrade the rate of convergence of the conjugate gradient method compared to the natural ordering [20]. Since PSOR can keep the natural ordering within each strip partition subgrid, we can expect a parallel SSOR method, defined by using PSOR, to be an efficient parallel preconditioner for the conjugate gradient method. In a subsequent work, we plan to compare a PSOR–SSOR preconditioner to a SSOR preconditioner that uses the natural rowwise ordering.

The remainder of this paper is organized as follows. In section 2, we present an analysis of the PSOR method for solving the five-point stencil approximation of Poisson's equation. In section 3, we present a general description of PSOR. In section 4, we prove that the ordering for PSOR is a *consistent ordering* whenever the global ordering for a domain-partitioned SOR is consistently ordered. We note that this is the case for the five-point stencil on both strip and block partitions. From this result, we can conclude that PSOR has the same asymptotic rate of convergence as the natural rowwise SOR method. We also prove that PSOR has the same asymptotic convergence rate as rowwise SOR and R/B/G/O SOR methods for the nine-point stencil and strip partitions. In section 5, we demonstrate the parallel performance of the PSOR method using either a strip or a block partition on the four distinct multiprocessor computers. We also compare the PSOR method versus the R/B SOR method for the five-point stencil and the R/B/G/O SOR method for the nine-point stencil. Finally, some conclusions are given in section 6.

2. Analysis of PSOR for a model problem. We consider the five-point approximation to Poisson's equation on a unit square with zero boundary data

$$4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{ij} \quad \text{in } \Omega_h \tag{2.1}$$

and $u_{ij} = 0$ on $\partial\Omega_h$. Here, grid size $h = 1/(m+1)$ for some positive integer m, f_{ij} is the value of function f at mesh point (ih, jh), u_{ij} denotes the approximation of $u(ih, jh)$, and Ω_h and $\partial\Omega_h$ are the sets of the interior mesh points and boundary mesh points, respectively.

Under some ordering of unknowns, (2.1) can be written in a matrix form $Au = f$ with A being an $m^2 \times m^2$ matrix, and both u and f being column vectors of order m^2 with components u_{ij} and $h^2 f_{ij}$ for $i, j = 1, 2, \ldots, m$, respectively. Obviously, there are many ways to order the unknowns, but the natural rowwise ordering as shown in Figure 2.1 and the red/black ordering as shown in Figure 2.2 (here mesh points (ih, jh) with $i + j$ being even and odd are called red and black points, respectively) are two widely used orderings in practice.

The SOR method using the natural rowwise ordering generates a sequence of iterates from a given initial guess $u_{ij}^{(0)}$ and a real number $\omega \in (0, 2)$ by the form

$$u_{ij}^{(k+1)} = (1-\omega)u_{ij}^{(k)} + \frac{\omega}{4}(h^2 f_{ij} + u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}), \tag{2.2}$$

which is completely sequential.

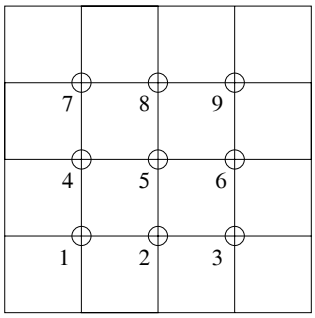

FIG. 2.1. *Natural rowwise ordering.*

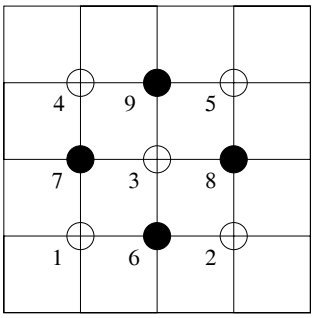

FIG. 2.2. *Red/black ordering.*

The R/B SOR method defines iterates $u_{ij}^{(k+1)}$ first on red points by

$$u_{ij}^{(k+1)} = (1-\omega)u_{ij}^{(k)} + \frac{\omega}{4}(h^2 f_{ij} + u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}) \tag{2.3}$$

and then on black points by

$$u_{ij}^{(k+1)} = (1-\omega)u_{ij}^{(k)} + \frac{\omega}{4}(h^2 f_{ij} + u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k+1)} + u_{i,j+1}^{(k+1)}). \tag{2.4}$$

Clearly, R/B SOR can be implemented entirely in parallel on the same colors.

For the model problem, it has been shown that the natural rowwise ordering and the red/black ordering are "consistent orderings"; hence, the SOR method using the rowwise ordering and the R/B SOR method have the same convergence rate [36].

JSOR is a parallel version of SOR by domain decomposition, which has been analyzed in [34]. For simplicity, we suppose that the grid mesh Ω_h is partitioned into p strips $\Omega_{h,\nu}$ for $\nu = 1, 2, \ldots, p$. Each strip contains at least two grid lines. We denote by $\Omega_{h,\nu}^1$ the first grid line of strip $\Omega_{h,\nu}$, and $\Omega_{h,\nu}^2 = \Omega_{h,\nu} - \Omega_{h,\nu}^1$. Then the JSOR iterate $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^1$ is defined by

$$u_{ij}^{(k+1)} = (1-\omega)u_{ij}^{(k)} + \frac{\omega}{4}(h^2 f_{ij} + u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)}), \tag{2.5}$$

while the JSOR iterate $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^2$ is the same as (2.2).

Clearly, the JSOR scheme can be implemented in parallel on p processors by mapping $\Omega_{h,\nu}$ into processor ν for $\nu = 1, 2, \ldots, p$. Between JSOR iterations, we need to communicate the updated values of $u_{ij}^{(k+1)}$ on the first and last grid lines between processors in two steps:

Step 1. Send $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^1$ from processor ν to processor $\nu-1$ for $\nu = 2, 3, \ldots, p$.

Step 2. Send $u_{ij}^{(k+1)}$ on the last grid line of $\Omega_{h,\nu}$ from processor ν to processor $\nu+1$ for $\nu = 1, 2, \ldots, p-1$.

Since each JSOR iteration needs to do interprocessor data communication only twice (once in Step 1 and once in Step 2), the same as the Jacobi method implemented in parallel on the strip partition, JSOR can be efficiently implemented on parallel machines. While it is faster than the Jacobi method, the convergence rate of JSOR slows down almost linearly with respect to the number p of strips [34]. Due to this, it is not an efficient parallel solver for linear systems but is still a robust smoother for parallel multigrid methods [33, 34].

The PSOR method is a new type of parallel SOR that is generated from JSOR by a novel use of interprocessor data communication techniques. In PSOR, Step 1 of JSOR is carried out as soon as the computation of (2.5) is done, so that the updates $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^1$ for $\nu = 2, 3, \ldots, p$ can be employed by the updates $u_{ij}^{(k+1)}$ on the last grid line of $\Omega_{h,\nu}$ such as are defined by

$$u_{ij}^{(k+1)} = \frac{\omega}{4}(h^2 f_{ij} + u_{i-1,j}^{(k+1)} + u_{i,j-1}^{(k+1)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k+1)}) + (1-\omega)u_{ij}^{(k)}. \tag{2.6}$$

For clarity, we write a pseudocode of PSOR on p processors for five-point strip partitions below.

PSOR Algorithm (Five-point Strips). *For $\nu = 1, 2, \ldots, p$ in parallel*

(a) *compute $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^1$ by using* (2.5);

(b) *do Step* 1 *of the interprocessor data communication;*

(c) *compute $u_{ij}^{(k+1)}$ on $\Omega_{h,\nu}^2$ by using* (2.2) *and on the last grid line by using* (2.6);

(d) *do Step* 2 *of the interprocessor data communication.*

Clearly, if we switch the positions of (b) and (c), (2.6) reverts to equation (2.2), so that PSOR goes back to JSOR. This indicates that the computation of JSOR and that of PSOR use the same local ordering and that their implementations on a parallel computer differ only in the arrangement of interprocessor data communication. Hence, PSOR can be implemented as efficiently as JSOR on parallel computers.

Remarkably, PSOR can have a much faster convergence rate than JSOR. For the model problem, we can use the SOR theory in [36] to demonstrate that PSOR can have the same asymptotic rate of convergence as the SOR iteration (2.2). We postpone to the next two sections the discussion and analysis of PSOR on more general problems.

In fact, PSOR with a strip partition is equivalent to SOR using a new ordering as shown in Figure 2.3. Let $A = (a_{ij})$ be the matrix associated with the new ordering. Here $a_{ij} \neq 0$ if and only if mesh node i is adjacent to node j. For the ordering as shown in Figure 2.3, we have a disjoint partition of the index set $W = \{1, 2, \ldots, 36\}$: $W = \cup_{i=1}^{7} S_i$, where

$$S_1 = \{1, 7, 13\}, \quad S_2 = \{2, 8, 14, 19, 25, 31\}, \quad S_3 = \{3, 9, 15, 20, 26, 32\},$$
$$S_4 = \{4, 10, 16, 21, 27, 33\}, \quad S_5 = \{5, 11, 17, 22, 28, 34\},$$
$$S_6 = \{6, 12, 18, 23, 29, 35\}, \quad \text{and} \quad S_7 = \{24, 30, 36\}.$$

Obviously, if $a_{i,j} \neq 0$ and $i \in S_k$, then $j \in S_{k+1}$ if $j > i$ and $j \in S_{k-1}$ if $j < i$. This shows that these sets satisfy the definition of a consistently ordered matrix [36]; thus A is consistently ordered. Therefore, from the SOR theory in [36] it follows that the PSOR method has the same asymptotic rate of convergence as the SOR iteration (2.2) and hence the R/B SOR iteration (2.3) and (2.4).

Similarly, we can define the PSOR method on a block partition as shown in Figure 2.5. In order to communicate the data between processors efficiently, we propose a particular local ordering of the mesh points at each block $\Omega_{h,\nu}$ as shown in Figure 2.6, which is not the local rowwise ordering but is still consistently ordered. In fact, for the ordering shown in Figure 2.6, we construct the following subsets:

$$S_1 = \{1\}, \quad S_2 = \{2, 5\}, \quad S_3 = \{3, 6, 8\}, \quad S_4 = \{4, 7, 9, 11\},$$
$$S_5 = \{10, 12, 14\}, \quad S_6 = \{13, 15\}, \quad \text{and} \quad S_7 = \{16\},$$

which satisfy the definition of a consistently ordered matrix.

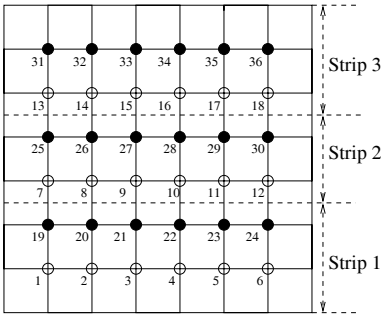

FIG. 2.3. *A global PSOR ordering on a strip partition.*

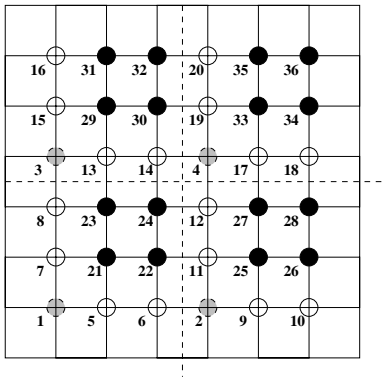

FIG. 2.4. *A global PSOR ordering on a block partition.*

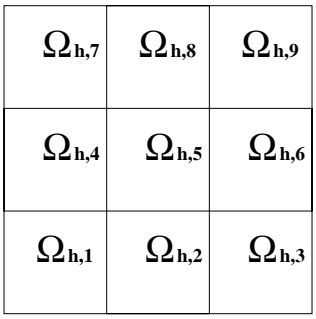

FIG. 2.5. *A block partition of the grid mesh domain* Ω_h.

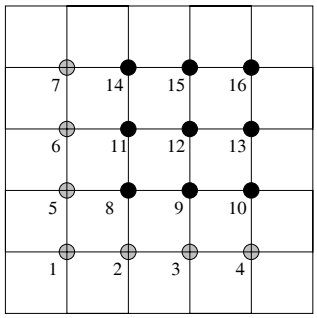

FIG. 2.6. *A local ordering on each block* $\Omega_{h,\mu}$.

PSOR on a block partition is also equivalent to the SOR method with a new global ordering as shown in Figure 2.4 (which is consistently ordered). In fact, for the ordering shown in Figure 2.4, we have the following subsets:

$$S_1 = \{1\}, \quad S_2 = \{2,3,5,7\}, \quad S_3 = \{4,6,8,9,11,13,15,21\},$$
$$S_4 = \{10,12,14,16,17,19,22,23,25,29\}, \quad S_5 = \{18,20,24,26,27,30,31,33\},$$
$$S_6 = \{28,32,34,35\}, \quad \text{and} \quad S_7 = \{36\},$$

which satisfy the definition of a consistently ordered matrix. Therefore, from the SOR theory it follows that PSOR on the block partition has the same convergence rate as the SOR method using the natural rowwise and the R/B orderings.

During the implementation of PSOR we must take care to communicate the values on the perimeter of each partition to neighboring processors before that value is needed in a calculation according to the precedence indicated in the global ordering. This ensures that PSOR is indeed a successive overrelaxation method. For example, in the block partition case from Figure 2.4, we observe that the value of node 4 must be sent west and south to be used in the calculation of nodes 14 and 12, respectively. By replicating the local ordering shown in Figure 2.6 to all processors, we can implement the PSOR method as shown in the global ordering in Figure 2.4 with only five send messages each iteration combined with calculation as follows: (calculate 1),

(node 1—send west), (calculate 2,3,4), (nodes 1,2,3,4—send south), (calculate 5,6,7), (nodes 5,6,7—send west), (calculate 8 to 16), (nodes 4,10,13,16—send east), (nodes 7,14,15,16—send north). This is a savings over the seven messages that would be required during a R/B SOR iteration for the same partition. We note that only seven instead of the normal eight messages are needed since R and B nodes can be sent to the south processor simultaneously and still arrive there in time for the B nodes in that processor to utilize the new R values (assuming at least a 4×4 block of nodes per processor).

3. General form of the PSOR iteration. We consider the solution of the linear system that arises from a finite element or a finite difference discretization of an elliptic boundary value problem. We assume that the mesh domain is decomposed into p partitions and each partition can be divided into t types such that the nodes of a given type are not connected across partitions. This means that the entry a_{ij} of matrix A is zero if nodes i and j are the same type but on different partitions. There is no restriction of connectivity within a given partition (nodes of the same type *can* be connected within the same partition). We further assume that nodes of type i in a partition μ can be connected only to nodes of type greater than i in partitions numbered less than μ and can be connected only to nodes of type less than i in partitions numbered greater than μ. We order first by partitions, then within partitions by node type. The resulting matrix A has the block form

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix}, \tag{3.1}$$

where

$$A_{\mu\nu} = \begin{bmatrix} A_{\mu\nu}^{11} & A_{\mu\nu}^{12} & \cdots & A_{\mu\nu}^{1t} \\ A_{\mu\nu}^{21} & A_{\mu\nu}^{22} & \cdots & A_{\mu\nu}^{2t} \\ \vdots & \vdots & \ddots & \vdots \\ A_{\mu\nu}^{t1} & A_{\mu\nu}^{t2} & \cdots & A_{\mu\nu}^{tt} \end{bmatrix}, \quad \mu, \nu = 1, \ldots, p, \tag{3.2}$$

and $A_{\mu\nu}^{ij}$ represents the connectivity between unknowns (nodes) of type i in partition μ to unknowns of type j in partition ν for $\mu, \nu = 1, 2, \ldots, p$ and $i, j = 1, 2, \ldots, t$.

With our assumptions, the matrices $A_{\mu\nu}$ in (3.2) can be simplified as

$$A_{\mu\nu} = \begin{cases} \begin{bmatrix} 0 & A_{\mu\nu}^{12} & \cdots & A_{\mu\nu}^{1t} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & A_{\mu\nu}^{t-1,t} \\ 0 & 0 & \cdots & 0 \end{bmatrix} & \text{for } \nu < \mu, \\ \begin{bmatrix} 0 & 0 & \cdots & 0 \\ A_{\mu\nu}^{21} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ A_{\mu\nu}^{t1} & \cdots & A_{\mu\nu}^{t,t-1} & 0 \end{bmatrix} & \text{for } \nu > \mu. \end{cases} \tag{3.3}$$

Based on the above partitions, we define block forms of vectors u and f by

$$u = ((\mathcal{U}_1)^T, (\mathcal{U}_2)^T, \ldots, (\mathcal{U}_p)^T)^T \quad \text{and} \quad f = ((\mathcal{F}_1)^T, (\mathcal{F}_2)^T, \ldots, (\mathcal{F}_p)^T)^T, \tag{3.4}$$

where $\mathcal{U}_\mu = ((\mathcal{U}_\mu^1)^T, (\mathcal{U}_\mu^2)^T, \ldots, (\mathcal{U}_\mu^t)^T)^T$, $\mathcal{F}_\mu = ((\mathcal{F}_\mu^1)^T, (\mathcal{F}_\mu^2)^T, \ldots, (\mathcal{F}_\mu^t)^T)^T$, and $\mathcal{U}_\mu^j$ and $\mathcal{F}_\mu^j$ are column vectors for $\mu = 1, 2, \ldots, p$ and $j = 1, 2, \ldots, t$. Here the superscript T denotes a vector transpose.

Using (3.1) and (3.3), we write the linear system $Au = f$ in a block form for the equations of type i in partition μ as

$$\sum_{j=1}^{t} A_{\mu\mu}^{ij}\mathcal{U}_\mu^j + \sum_{\nu=1}^{\mu-1}\sum_{j=i+1}^{t} A_{\mu\nu}^{ij}\mathcal{U}_\nu^j + \sum_{\nu=\mu+1}^{p}\sum_{j=1}^{i-1} A_{\mu\nu}^{ij}\mathcal{U}_\nu^j = \mathcal{F}_\mu^i, \tag{3.5}$$

where $i = 1, 2, \ldots, t$, and $\mu = 1, 2, \ldots, p$.

For five-point and nine-point strip partitions (see below) where nodes in the first row of each partition are type 1 and nodes in successive rows are type 2, equation (3.5) is satisfied (with $t = 2$). In addition, matrix A in (3.1) is the natural rowwise ordered matrix if we maintain rowwise precedence within nodes of the same type in each partition.

This equation is also satisfied by the block partitions of five-point and nine-point stencils, with three and four types ($t = 3$ and 4) of nodes, respectively, as shown below. For these partitions, A in (3.1) is not the natural rowwise ordered matrix.

```
2 2 2 2          2  3  3  3          3  4  4  4
2 2 2 2          2  3  3  3          3  4  4  4
2 2 2 2          2  3  3  3          1  4  4  4
1 1 1 1          1  2  2  2          1  1  2  2
strips        5-point block       9-point block
```

The PSOR Algorithm can be described in general for each iteration as follows.

General PSOR Algorithm. *For $\mu = 1, 2, \ldots, p$ in parallel do*

For $i = 1, 2, \ldots, t$ do

1. *Calculate $\mathcal{U}_\mu^i$ (unknowns of type i in partition μ) by applying one-point–SOR iteration to equation* (3.5) *using all available updated iterates.*
2. *Communicate the appropriate portion of nodes of type $j \le i$ as needed to minimize the number of communication packets and to ensure nodes of type $i+1$ use updated information from connected nodes of type less than or equal to i.*

We note that other node typings with *fewer* node types exist for these stencils, but we don't consider them here unless (3.3) is satisfied. One such partition for the five-point stencil, for example, would replace the type 3 nodes in the five-point block partition above with type 1 nodes. This would violate (3.3) if blocks are ordered left to right, bottom to top.

Let D_μ be the diagonal of $A_{\mu\mu}$ and L_μ and U_μ be the strictly lower and upper triangular matrices, respectively, such that $D_\mu^{-1}A_{\mu\mu} = I - L_\mu - U_\mu$. Then we can see that matrix A can be written as

$$D^{-1}A = I - B - N - C - M, \tag{3.6}$$

with $D = \text{diag}(D_1, D_2, \ldots, D_p)$, $B = \text{diag}(L_1, L_2, \ldots, L_p)$, $C = \text{diag}(U_1, U_2, \ldots, U_p)$,

$$N = -D^{-1} \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ A_{21} & 0 & 0 & \cdots & 0 \\ A_{31} & A_{32} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{p(p-1)} & 0 \end{bmatrix},$$

and

$$M = -D^{-1} \begin{bmatrix} 0 & A_{12} & A_{13} & \cdots & A_{1p} \\ 0 & 0 & A_{23} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & A_{(p-1)p} \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

So a general form of the PSOR iterations can be expressed by

$$u^{(k+1)} = M_{PSOR}(\omega)u^{(k)} + F(\omega) \quad \text{for } k = 0, 1, 2, \ldots, \tag{3.7}$$

where $u^{(0)}$ is an initial guess,

$$M_{PSOR}(\omega) = [I - \omega(B + M)]^{-1}[(1 - \omega)I + \omega(C + N)], \tag{3.8}$$

and $F(\omega) = \omega[I - \omega(B + M)]^{-1}D^{-1}f$. We refer to $M_{PSOR}(\omega)$ as the PSOR iteration matrix.

4. General PSOR analysis. In this section we show that PSOR is the successive overrelaxation method applied to a reordered linear system. So, the general SOR theory can be used to study the convergence of PSOR.

Based on the partitions in PSOR, we define another global ordering: order first by node type, then within node type by partitions. Also, order nodes of the same type within a given partition in the same precedence that was used to define A in (3.1). Let P be a permutation matrix that relates the unknowns ordered by this global ordering to those first ordered by partition as seen in (3.4). Then the linear system $Au = f$ can be reordered to the form $\hat{A}\hat{u} = \hat{f}$ with $\hat{u} = P^T u$, $\hat{f} = P^T f$, and $\hat{A} = P^T AP$.

Using the notation in (3.2), we get

$$\hat{A} = \begin{bmatrix} \hat{A}_{11} & \hat{A}_{12} & \cdots & \hat{A}_{1t} \\ \hat{A}_{21} & \hat{A}_{22} & \cdots & \hat{A}_{2t} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{A}_{t1} & \hat{A}_{t2} & \cdots & \hat{A}_{tt} \end{bmatrix}, \tag{4.1}$$

where $\hat{A}_{ij}$ $(i, j = 1, 2, \ldots, t)$ represents the connections of nodes of type i to those of type j and can be expressed by

$$\hat{A}_{ij} = \begin{cases} \operatorname{diag}(A_{11}^{ii}, A_{22}^{ii}, \ldots, A_{pp}^{ii}) & \text{if } i = j, \\ \begin{bmatrix} A_{11}^{ij} & 0 & \cdots & 0 \\ A_{21}^{ij} & A_{22}^{ij} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ A_{p1}^{ij} & A_{p2}^{ij} & \cdots & A_{pp}^{ij} \end{bmatrix} & \text{if } i < j, \\ \begin{bmatrix} A_{11}^{ij} & A_{12}^{ij} & \cdots & A_{1p}^{ij} \\ 0 & A_{22}^{ij} & \cdots & A_{2p}^{ij} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & A_{pp}^{ij} \end{bmatrix} & \text{if } i > j. \end{cases} \tag{4.2}$$

Now, if we let $\hat{D} = \operatorname{diag}(\hat{A})$ and split $\hat{D}^{-1}\hat{A} = I - \hat{L} - \hat{U}$, where $\hat{L}$ and $\hat{U}$ are lower and upper triangular matrices, respectively, we can see that the SOR method applied to the reordered system $\hat{A}\hat{u} = \hat{f}$ can be written as

$$(I - \omega\hat{L})\hat{u}^{(k+1)} = ((1-\omega)I + \omega\hat{U})\hat{u}^{(k)} + \omega\hat{D}^{-1}\hat{f}, \tag{4.3}$$

and the iteration matrix $\hat{M}_{SOR}$ from (4.3) is given by

$$(I - \omega\hat{L})^{-1}((1-\omega)I + \omega\hat{U}). \tag{4.4}$$

A simple calculation gives that $\hat{D} = P^T DP, \hat{L} = P^T(B+M)P$, and $\hat{U} = P^T(C+N)P$. Hence, the iteration matrices in (3.8) and (4.4) have the same eigenvalues because they are related by

$$\hat{M}_{SOR}(\omega) = P^T M_{PSOR}(\omega) P.$$

We will say two iterative methods are *equivalent* if their iteration matrices have the same eigenvalues. Therefore, PSOR is equivalent to SOR applied to $\hat{A}$.

We would like to show, however, that PSOR is equivalent to SOR applied to A. If both A and $\hat{A}$ were consistently ordered, we would know from SOR theory that the two methods would be equivalent. The following theorem shows this is true if A is consistently ordered.

THEOREM 4.1. *Let A and $\hat{A}$ be defined in* (3.1)–(3.3) *and in* (4.1)–(4.2)*, respectively. If A is consistently ordered, then $\hat{A}$ is consistently ordered.*

Proof. The proof is by construction of the consistently ordered sets for $\hat{A}$. Let P^T map the node numbered i in vector u to the node numbered $\sigma(i)$ in vector $\hat{u}$. Since A is consistently ordered, it follows that each partition $A_{\mu\mu}$ for $\mu = 1, 2, \ldots, p$ is consistently ordered with sets $S_1^{(\mu)}, S_2^{(\mu)}, \ldots, S_{n_\mu}^{(\mu)}$. Since the same ordering precedence is maintained in A and $\hat{A}$ for nodes within a given partition, we can construct n_μ consistently ordered sets for nodes in partition μ using the ordering used in $\hat{A}$ by simply performing the mapping $\sigma(i)$ to each node i in the sets above. This yields the consistently ordered sets $\hat{S}_1^{(\mu)}, \hat{S}_2^{(\mu)}, \ldots, \hat{S}_{n_\mu}^{(\mu)}$.

We begin by merging the n_2 sets $\{\hat{S}_j^{(2)}\}_{j=1}^{n_2}$ from partition 2 into the n_1 sets $\{\hat{S}_j^{(1)}\}_{j=1}^{n_1}$ from partition 1. To do this, we find a set, say $\hat{S}_i^{(2)}$, in partition 2 that has a node with a connection to some set, say $\hat{S}_k^{(1)}$, in partition 1. Since A is consistently ordered, we know that $\hat{S}_k^{(1)}$ is the only set in partition 1 that nodes in set $\hat{S}_i^{(2)}$ in

partition 2 can have connections to. It also follows that if any nodes in set $\hat{S}_{i+1}^{(2)}$ in partition 2 have connections to nodes in partition 1, they must be connected to nodes in set $\hat{S}_{k+1}^{(2)}$; otherwise, matrix A would not have been consistently ordered. Hence, once we find the correct place to merge set $\hat{S}_i^{(2)}$ from partition 2, the placement of the other sets in partition 2 is determined in a sequential fashion. We know the node in set $\hat{S}_k^{(1)}$ in partition 1 must be of higher type than the node it is connected to in set $\hat{S}_i^{(2)}$ of partition 2; hence, we must merge set $\hat{S}_i^{(2)}$ with set $\hat{S}_{(k-1)}^{(1)}$. The newly merged consistently ordered sets are then $\hat{S}_1^{(1)}, \ldots, \hat{S}_{k-i-1}^{(1)}, \hat{S}_1^{(2)} \cup \hat{S}_{k-i}^{(1)}, \ldots, \hat{S}_i^{(2)} \cup \hat{S}_{k-1}^{(1)}, \ldots, \hat{S}_{n_1+i-k+1}^{(2)} \cup \hat{S}_{n_1}^{(1)}, \hat{S}_{n_1+i-k+2}^{(2)}, \ldots, \hat{S}_{n_2}^{(2)}$. We denote these merged sets as $Q_1, Q_2, \ldots, Q_{K_2}$.

Next, we merge the sets from partition 3 into the newly merged sets from the previous step and continue until the sets from partition p are merged into the newly merged sets from the partitions $1, 2, \ldots, p-1$. Suppose after step $j-1$ we have K_{j-1} consistently ordered sets that have resulted from merging the sets of partitions $1, 2, \ldots, j-1$. For simplicity we denote these sets as $Q_1, Q_2, \ldots, Q_{K_{j-1}}$. Then, since A is consistently ordered, each set in partition j can have nodes with connections to nodes in at most one of these Q's. Suppose that $\hat{S}_i^{(j)}$ has nodes with connections to nodes in set Q_k. Then we know that the connection is to a node of higher type in Q_k. Hence, we must include the $\hat{S}_i^{(j)}$ nodes in set Q_{k-1} because higher type connections are ordered later in $\hat{A}$. That is, Q_{k-1} is replaced by $Q_{k-1} \cup \hat{S}_i^{(j)}$. Once a set $\hat{S}_i^{(j)}$ with a connection to one of the Q's has been found, the rest of the sets in partition j are merged in sequence by including $\hat{S}_{i-1}^{(j)}$ in set Q_{k-2}, set $\hat{S}_{i-2}^{(j)}$ in set $Q_{k-3}, \ldots$, set $\hat{S}_1^{(j)}$ in set Q_{k-i}, set $\hat{S}_{i+1}^{(j)}$ in set $Q_k, \ldots$, and set $\hat{S}_{n_{j-1}+i-k+1}^{(j)}$ in set $Q_{K_{j-1}}$. The remaining sets in partition j, $\hat{S}_{n_{j-1}+i-k+2}^{(j)}, \ldots, \hat{S}_{n_j}^{(j)}$ are renamed $Q_{K_{j-1}+1}, \ldots, Q_{K_j}$, respectively. The process continues until sets in all p partitions are merged and renamed as needed. The matrix $\hat{A}$ is therefore consistently ordered with the resulting sets, Q_j for $j = 1, 2, \ldots, K_p$. □

From [36] we know that all consistent orderings of the same set of equations leads to SOR iteration matrices with the same eigenvalues. It is well known that the natural rowwise ordering and the R/B ordering for the five-point stencil are consistent orderings. For the strip partition, the matrix A represents this rowwise ordering. Theorem 4.1 then tells us that $\hat{A}$ is consistently ordered; hence, PSOR and rowwise SOR for the five-point stencil are equivalent.

Matrix A for block partitions can be taken to have the ordering shown in Figure 4.1 and is consistently ordered. This can be shown by constructing the consistently ordered sets or by simply observing that it is a nonmigratory permutation of the rowwise ordering. Hence, we know from the theorem above that $\hat{A}$ is consistently ordered and it follows that PSOR and rowwise SOR have the same asymptotic convergence rate.

In general, by using typings that satisfy (3.3) we do not see how to show the equivalence of PSOR (which is SOR applied to $\hat{A}$) and SOR (applied to A) without requiring A to be consistently ordered. Theorem 4.1 is of no use whenever A is not consistently ordered. However, if we restrict ourselves to strip partitions, we show below how we can make a comparison between PSOR and the natural rowwise SOR for the nine-point stencil.

For the special case of the nine-point stencil and strip partitions with a typing satisfying (4.2), we show below that PSOR is equivalent to the R/B/G/O SOR method, and hence to the natural rowwise SOR method.

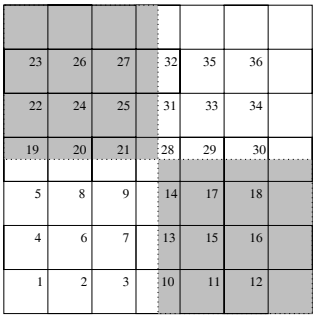

FIG. 4.1. *A global ordering for the five-point stencil leading to a matrix partition $A = (A_{\mu\nu})$ with A and all $A_{\mu\mu}$ being consistently ordered.*

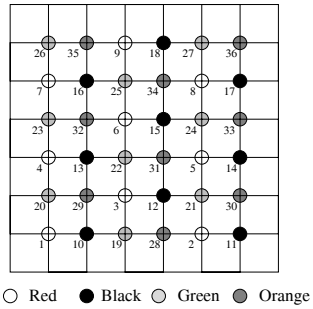

FIG. 4.2. *Four-color ordering for the nine-point stencil.*

We now consider the nine-point approximation to Poisson's equation,

$$\begin{aligned} &20u_{ij} - 4u_{i-1,j} - 4u_{i+1,j} - 4u_{i,j-1} - 4u_{i,j+1} \\ &\quad - u_{i-1,j+1} - u_{i-1,j-1} - u_{i+1,j-1} - u_{i+1,j+1} = h^2 f_{ij}, \text{ in } \Omega_h, \end{aligned} \tag{4.5}$$

with $u_{ij} = 0$ on $\partial\Omega_h$. We show that PSOR using a strip partition for solving (4.5) is equivalent to the R/B/G/O SOR method with the coloring shown in Figure 4.2.

The multicolor SOR method involves four colors for the nine-point stencil. There are three distinct four-color orderings, but we only consider one of them, which is illustrated in Figure 4.2 with ordering R/B/G/O. This multicolor SOR method was shown to be equivalent to the natural rowwise ordering in [2]. In the following theorem we show this ordering is also equivalent to PSOR on strip partitions with the node typing shown in Figure 2.3 which satisfies (4.2).

THEOREM 4.2. *The iteration matrices for the PSOR algorithm using node typing shown in Figure* 2.3 *and the R/B/G/O SOR algorithm illustrated in Figure* 4.2 *have the same eigenvalues for the nine-point stencil with strip partitions.*

Proof. We begin by coloring the nodes in each strip partition by R,B,G, or O so that the entire grid is colored as shown in Figure 4.2. This means the bottom row in each strip will either have the pattern R/B/G/O or G/O/R/B, depending on the assignment of the domain rows to strips as shown below.

```
 .   .   .   .   .   .          .   .   .   .   .   .   .   .
G2  O2  R3  B3  .   .          R3  B3  G3  O3  R4  B4  .   .
R2  B2  G2  O2  .   .          G2  O2  R3  B3  G3  O3  .   .
G1  O1  R2  B2  .   .          R2  B2  G2  O2  R3  B3  .   .
R1  B1  G1  O1  .   .          G1  O1  R2  B2  G2  O2  .   .
Local R/B/G/O Sets                 Local R/B/G/O Sets
```

We also assign each node to a local set, as indicated above by the number following the color. For example, if the bottom row begins with R, the nodes indicated by R1,B1,G1,O1 are in set 1, R2,B2,G2,O2 are in set 2, etc. Likewise, if the bottom row begins with G, the nodes indicated by G1,O1 are in set 1, R2,B2,G2,O2 are in set 2, etc.

Now, recall that the updating of nodes within a given strip with PSOR is bottom to top, left to right. We observe that within a strip this precedence is also maintained

by updating nodes in set 1 before nodes in set 2, etc., and within each set update R first, followed by B, then G, and finally O. Also, notice that for either of the local sets above, nodes in set j are connected only to nodes in sets $j-1$, j, and $j+1$. Furthermore, the nodes in set j are connected to other nodes in set j by a multicolor matrix, M_j, nodes in set j are connected to nodes of a higher color in set $j-1$ by an upper triangular matrix, U_{j-1}, and nodes in set j are connected to nodes of a lower color in set $j+1$ by a lower triangular matrix, L_j. Hence, the precedence maintained by PSOR *within* a strip that has s sets is given by the following *multicolor T matrix*:

$$\begin{bmatrix} M_1 & L_1 & & \\ U_1 & M_2 & \ddots & \\ & \ddots & \ddots & L_{s-1} \\ & & U_{s-1} & M_s \end{bmatrix}. \tag{4.6}$$

We now need to be able to show that the connections *between* strips is such that the precedence maintained by PSOR on the global problem domain is given by a *multicolor T matrix* like the one in (4.6). Then we can use Theorem 1 in Adams, Jordan [2] to conclude that PSOR is equivalent to R/B/G/O SOR. (Corollary 1 in Adams, Jordan [2] has already shown that the natural rowwise SOR is equivalent to R/B/G/O SOR.)

To find this global multicolor T matrix, we simply merge the sets from the local partitions in a way that maintains the precedence of the PSOR method. Assume the problem domain has been partitioned into K strips. Let strip i have n_i local sets, denoted by $s_1^{(i)}$ to $s_{n_i}^{(i)}$. Starting with strip 2, let $s_q^{(1)}$ denote the smallest numbered set in strip 1 that has connections with the first set in strip 2. If the bottom row of strip 2 begins with R, then we can merge the first set of strip 2 with the qth set of strip 1, the second set of strip 2 with the $(q+1)$st set of strip 1, etc. until we end up with the n_2+q-1 global sets $s_1^{(1)}$, $s_2^{(1)}$, $\ldots$, $s_{q-1}^{(1)}$, $s_q^{(1)} \cup s_1^{(2)}$, $s_{q+1}^{(1)} \cup s_2^{(2)}, \ldots, s_{n_1}^{(1)} \cup s_{n_1-q+1}^{(2)}$, $s_{n_1-q+2}^{(2)}, \ldots, s_{n_2}^{(2)}$. Likewise, if the bottom row of strip 2 begins with G, then we merge the first set of strip 2 with the $(q-1)$st set of strip 1, the second set of strip 2 with the qth set of strip 1, etc., until we end up with the n_2+q-2 global sets $s_1^{(1)}$, $s_2^{(1)}, \ldots, s_{q-1}^{(1)} \cup s_1^{(2)}$, $s_q^{(1)} \cup s_2^{(2)}, \ldots, s_{n_1}^{(1)} \cup s_{n_1-q+2}^{(2)}$, $s_{n_1-q+3}^{(2)}, \ldots, s_{n_2}^{(2)}$. This process continues for strips $i, i=3,\ldots,K$ by merging the local sets of strip i with those global sets created so far from strips 1 through $i-1$. The end result is the global multicolor T matrix and the proof is complete. ∎

5. Numerical examples. In this section, we demonstrate the parallel performance of the PSOR method on four parallel MIMD machines: the KSR1 at Houston, the Intel Delta and Paragon at Caltech, and the IBM SP2 at Cornell. Numerical results confirm that the PSOR method on either a strip partition or a block partition for the five-point or nine-point stencil has the same asymptotic convergence rate as the corresponding SOR method. We also compare the parallel performance of PSOR with the R/B and four-color (R/B/G/O) SOR methods on the Paragon and show the superiority of PSOR.

Two PSOR programs were written in Pfortran [4] and MPI [27], respectively. The programs were compiled with optimization level $-O2$ on the KSR1 and the SP2 and $-O4$ on the Intel Delta and the Paragon, respectively. In the Pfortran program, the CPU time was computed by using the *dclock*() system routine on the Intel Delta and the Paragon, the *user_seconds*() system routine on the KSR1, and the *mclock*() on

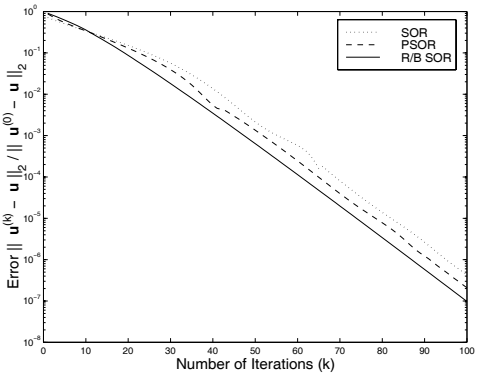

FIG. 5.1. *Convergence comparison of PSOR vs. SOR and R/B SOR for solving the five-point stencil* (2.1) *with* $f(x, y) = 0$ *and* $h = 1/33$.

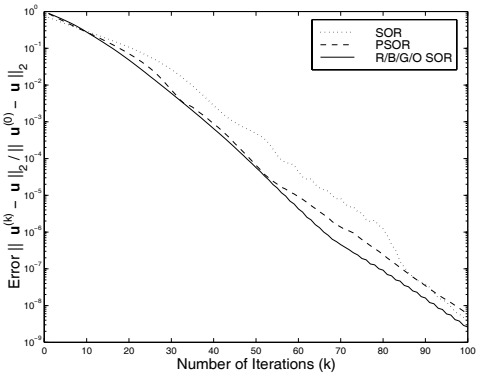

FIG. 5.2. *Convergence comparison of PSOR vs. SOR and R/B SOR for solving the nine-point stencil* (4.5) *with* $f(x, y) = 0$ *and* $h = 1/33$.

the SP2. In the MPI program, MPI function *mpi_wtime*() was used. For simplicity, we fixed the grid size $h = 1/513$, the relaxation parameter $\omega = 1.99$ (which is "optimal" according to our experience), and the initial guess $u^{(0)} = 0$ for all of the numerical experiments in this paper, except those in section 5.1.

In the figures and tables, *Linear Time* stands for the ideal case, which is defined by $T(1)/p$ on p processors, where $T(1)$ is the CPU time on one processor. *Total Time* represents the CPU time spent from the beginning of the iteration until either (5.2) is satisfied or the maximum number of iterations is reached. It does not include the CPU time spent on the calculation of f, the initial guess, and the input/output of data. *Comm. Time* represents the CPU time spent on the interprocessor data communication. *Comp. Time* represents the CPU time spent on the computation of the iteration, including the local L_2-norm of the residual. *Other Time* is *Total Time* − *Comm. Time* − *Comp. Time*, which includes the time spent on the global summations in the computation of (5.2). This also indicates the accuracy of our time measurements.

5.1. Convergence comparison of PSOR with SOR and multicolor SOR. In the previous section, we showed theoretically that PSOR has the same asymptotic rate of convergence as the corresponding sequential SOR method applied to matrix A. In this subsection, we present numerical examples to verify this conclusion.

Figure 5.1 compares the relative errors of the PSOR iterates with those of SOR and R/B SOR for solving the five-point stencil (2.1) with $f(x, y) = 0$ and $h = 1/33$, while Figure 5.2 gives the comparison of PSOR, SOR and R/B/G/O SOR for the nine-point stencil (4.5) with $f(x, y) = 0$ and $h = 1/33$. Here, we set the initial $u^{(0)} = 1$, the optimal relaxation parameter $\omega_{opt} = 2/(1 + \sin \pi h) \approx 1.8262$, and the number of iterations as 100 for all SOR, PSOR, R/B SOR, and R/B/G/O SOR methods. Since the exact solution $u = 0$ and $u^{(0)} = 1$, we computed the relative error by using the following formula:

$$\|u^{(k)} - u\|_2 / \|u^{(0)} - u\|_2 = \frac{(h-1)}{h} \|u^{(k)}\|_2,$$

where $\|\cdot\|_2$ is the L_2 norm.

The PSOR method was based on a strip partition (with 16 strips, each of them containing only two grid lines), and implemented on 16 processors of the Intel Paragon.

TABLE 5.1
PSOR on strips versus blocks for the model (2.1) *with* $h = 1/513$ *on Paragon.*

Processor	Total time		Comm. time		Err ($\times 10^{-5}$)		Resid ($\times 10^{-6}$)	
	Strips	Blocks	Strips	Blocks	Strips	Blocks	Strips	Blocks
1	247.29		0		7.37		2.27	
4	63.40	62.78	1.48	2.21	7.18	7.21	0.96	1.29
16	17.17	17.87	1.61	2.52	6.55	6.98	1.31	0.85
64	6.18	6.52	2.22	2.55	5.93	6.56	0.84	1.17
256	5.44	6.65	4.49	5.70	6.67	5.81	0.78	2.70

SOR, R/B SOR, and R/B/G/O SOR methods were implemented on one processor of the Paragon. These two figures demonstrate the convergence histories of PSOR, SOR, R/B SOR, and R/B/G/O SOR. Since the curves of the relative errors of PSOR are very close to that of SOR, R/B SOR (for the five-point), and R/B/G/O SOR (for the nine-point), they numerically indicate that PSOR has the same asymptotic rate of convergence as SOR and R/B SOR as well as R/B/G/O SOR.

For the model five-point problem with $f = 0$ and $h = 1/33$ we found the values of the reduction factor, $(\|u^{(100)} - u\|_2/\|u^{(0)} - u\|_2)^{1/100}$, after 100 iterations to be 0.8339, 0.8279, and 0.8217 for SOR, PSOR, and R/B SOR, respectively. These values are close to the known spectral radius $\omega_{opt} - 1 = 0.8262$. Likewise, the reduction factors for SOR, PSOR, and R/B/G/O SOR for the nine-point stencil (4.5) with $f = 0$ and $h = 1/33$ were computed to be 0.7957 (for SOR), 0.7990 (for PSOR), and 0.7925 (for R/B/G/O SOR). Hence, SOR, PSOR, and R/B/G/O SOR have the same asymptotical convergence rate in practice.

5.2. PSOR on strips vs. blocks. We considered the five-point stencil (2.1) with $f(x, y) = 2\pi^2 \sin \pi x \sin \pi y$. We fixed the number of PSOR iterations as 1000. In the strip partition case, the grid mesh was partitioned into p strips with equal sizes on p processors. In the block case, the grid mesh was divided into 2×2, 4×4, 8×8 and 16×16 blocks with equal sizes when PSOR was implemented on 4, 16, 64, and 256 processors, respectively.

Table 5.1 compares the performance of the PSOR method on strips versus blocks on the Paragon. Here the relative residual *Resid* and the relative error *Err* are defined by

$$Resid = \|f - Au^{(1000)}\|_2/\|f\|_2 \quad \text{and} \quad Err = \|u^{(1000)} - u\|_2/\|u\|_2,$$

where $u^{(1000)}$ is the 1000th PSOR iterate and u is the exact solution on the grid mesh. Note that PSOR on one processor is just the SOR using the natural rowwise ordering. Table 5.1 verifies that after 1000 iterations both methods have essentially the same residual. The table also shows that PSOR has a better performance on the strip partition than on the block partition because the strip partition has a lower communication overhead. Hence, we only consider the case of strip partitions in the remainder of the paper.

5.3. Performance of PSOR on four parallel machines. To demonstrate the performance of PSOR on different parallel machines, we considered the model problem (2.1) with $f(x, y) = 1$. We used the same Pfortran code on the KSR1 and the Intel Delta, and the same MPI code on the Paragon and the SP2. Noting that PSOR was implemented in the form

$$u^{(k)} = u^{(k-1)} + \omega(f - D[(I - C - N)u^{(k-1)} - (B + M)u^{(k)}]), \tag{5.1}$$

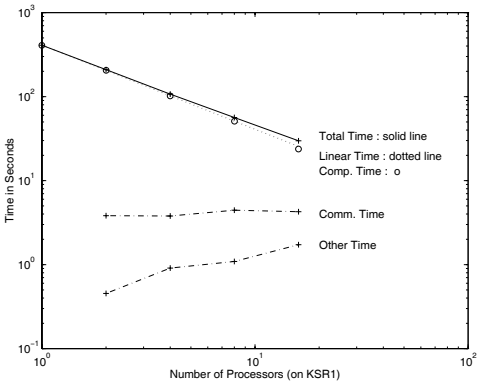

FIG. 5.3. *PSOR on a KSR*1 *with floating point performance* 6.58 *Mflops on one processor.*

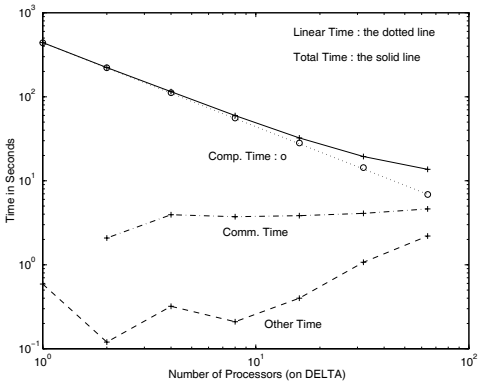

FIG. 5.4. *PSOR on the Intel Delta with floating point performance* 6.09 *Mflops on one processor.*

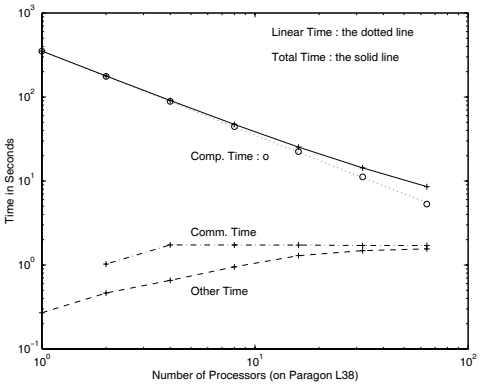

FIG. 5.5. *PSOR on an Intel Paragon with floating point performance* 7.678 *Mflops on one processor.*

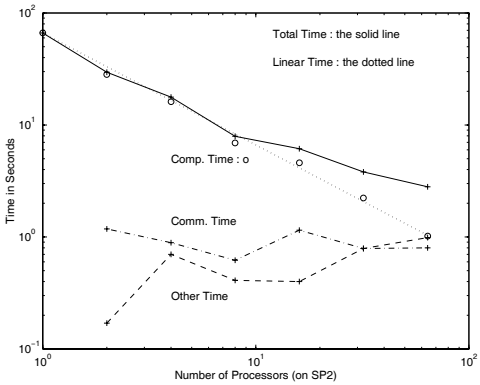

FIG. 5.6. *PSOR on an IBM SP*2 *with floating point performance* 40.63 *Mflops on one processor.*

we used the following stopping convergence criterion in these tests:

$$\|f - D[(I - C - N)u^{(k-1)} - (B + M)u^{(k)}]\|_2 \leq 10^{-5}. \tag{5.2}$$

Since the term $f - D[(I - C - N)u^{(k-1)} - (B + M)u^{(k)}]$ has been calculated in (5.1), criterion (5.2) can save CPU time in checking the termination of PSOR iterations.

Figures 5.3 to 5.6 display the parallel performance of PSOR as a function of the number of processors on the KSR1, the Intel Delta, the Paragon, and the SP2, respectively. The total numbers of PSOR iterations determined by the convergence criterion (5.2) on 1, 2, 4, 8, 16, 32, and 64 processors are 1027, 1026, 1025, 1023, 1018, 1008, and 942, respectively. From the figures we see that *Total Time* is very close to *Linear Time*, *Comp. Time* is almost the same as *Linear Time*, and both *Comm. Time* and *Other Time* are very small. These results demonstrate that the PSOR method is an efficient parallel version of the SOR method using the optimal relaxation parameter on these distinct MIMD multiprocessor machines.

Figure 5.7 compares the performance of PSOR on the KSR1 versus the Intel Delta. The KSR1 has a unique ring architecture that supports virtual shared memory. It

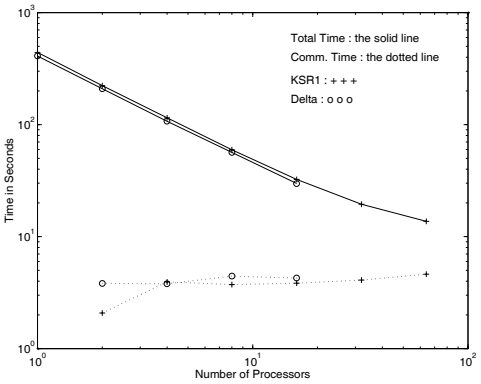

FIG. 5.7. *Parallel performance of PSOR on the KSR*1 *vs. the Intel Delta.*

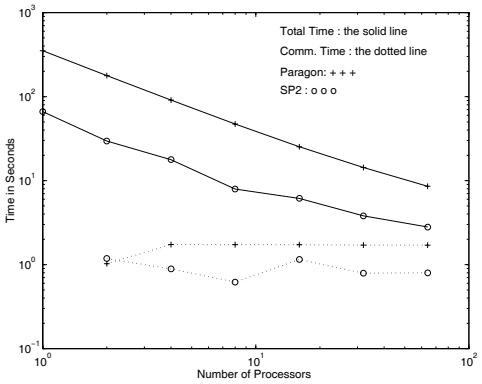

FIG. 5.8. *Parallel performance of PSOR on the Paragon vs. the SP*2.

uses RISC processors with 32 Mbytes of cache memory each. The Intel Delta is a system with processors connected by a two-dimensional mesh. It used the Intel i860 processor with 16 Mbytes memory each. From the figure we see that PSOR has largely the same performance on these two different machine architectures. This indicates that the KSR1 and the Intel Delta are similar in floating point capability and interprocessor data communication.

We also compare the performance of PSOR on the Intel Paragon versus the IBM SP2 in Figure 5.8. The Intel Paragon is the commercial successor to the Delta with a similar two-dimensional mesh, but with higher performance than the Delta. Each of its i860 processors has 32 Mbytes memory. The IBM SP2 is the latest distributed memory machine. It consists of nodes (i.e., processors with associated memory and disk) connected by an ethernet and a high-performance switch, where each node is one standard POWER2 RS/6000 processor together with associated memory and disk (64 to 512 Mbytes). Figure 5.8 shows that the SP2 is about three times faster than the Paragon in floating point operations and more efficient in the interprocessor message passing. On the other hand, Figure 5.8 shows that with the increase of the number of processors the CPU time is reduced more smoothly on the Paragon than on the SP2.

5.4. PSOR vs. multicolor SOR for strips. We considered the model problems (2.1) and (4.5) with $f = 1$ and $h = 1/513$ for the five-point and nine-point stencils, respectively. We set $\omega = 1.99$ and fixed the total number of iterations as 1000 for both PSOR and multicolor SOR. Numerical results are reported in Tables 5.2 and 5.3. The term *Residual* in Tables 5.2 and 5.3 denotes the value of $\|f - Au^{(1000)}\|_2$.

The R/B and R/B/G/O SOR methods were programmed in an efficient way. For illustration, we present a piece of the R/B SOR program for strip partitions in Figure 5.10. With our strip partition approach, each R/B and R/B/G/O SOR iteration require four and eight send messages (send south and north each color) per iteration, respectively. In contrast, each PSOR iteration only requires two send messages (send south and north) for both five-point and nine-point stencils (see Figure 5.9). Hence, PSOR can spend much less CPU time on interprocessor data communication than multicolor SOR. This is confirmed in Tables 5.2 and 5.3.

Tables 5.2 and 5.3 also indicate that PSOR spends much less CPU time on computation than the multicolor SOR method on the Paragon. Theoretically, the PSOR and multicolor SOR methods have the same number of floating point operations to

TABLE 5.2

A comparison of PSOR with R/B SOR for solving the five-point stencil (2.1) *with* $f = 1$ *and* $h = 1/513$ *on Paragon. Here the* Residual *of R/B SOR is* 2.57×10^{-5}.

Processor	Total time		Comm. time		Residual (PSOR)
	PSOR	RB-SOR	PSOR	RB-SOR	
1	241.2	346.1	0	0	3.07×10^{-5}
2	121.7	175.2	1.03	1.94	2.76×10^{-5}
4	62.1	89.9	1.68	3.18	2.16×10^{-5}
8	32.0	46.8	1.71	3.19	1.75×10^{-5}
16	16.9	25.1	1.74	3.29	2.05×10^{-5}
32	9.43	14.2	1.74	3.26	2.84×10^{-5}
64	5.68	8.87	1.79	3.31	3.93×10^{-5}
128	3.78	6.11	1.79	3.35	1.10×10^{-5}
256	2.97	5.16	2.00	3.86	1.19×10^{-5}

TABLE 5.3

A comparison of PSOR with R/B/G/O SOR for solving the nine-point stencil (4.5) *with* $f = 1$ *and* $h = 1/513$ *on Paragon. Here the* Residual *of the R/B/G/O SOR is* 4.88×10^{-6}.

Processor	Total time		Comm. time		Residual (PSOR)
	PSOR	RBGO-SOR	PSOR	RBGO-SOR	
1	330.7	573.4	0	0	8.54×10^{-6}
2	166.2	290.5	1.06	3.7	6.77×10^{-6}
4	84.4	150.0	1.78	6.21	4.24×10^{-6}
8	43.2	78.5	1.76	6.5	2.61×10^{-6}
16	22.5	42.6	1.77	6.58	2.38×10^{-6}
32	12.2	24.6	1.75	6.78	3.07×10^{-6}
64	7.1	15.8	1.83	6.5	4.27×10^{-6}
128	4.4	11.2	1.82	6.52	5.65×10^{-6}
256	3.33	10.8	2.02	8.64	6.96×10^{-6}

be calculated, but multicolor SOR requires more integer arithmetic and memory accesses. In fact, the results show that traversing the data structure only once during calculation, as PSOR does, can have a big advantage. (Note: R/B and R/B/G/O SOR require two and four memory accesses, respectively.) Table 5.3 also shows that the residuals for PSOR and R/B/G/O SOR are comparable as expected from our last theorem.

6. Conclusions. We have presented and analyzed an efficient parallel version of SOR called the PSOR method. Since it is defined by using domain decomposition and interprocessor data communication techniques, PSOR is simple to use. For strips, PSOR needs only two send messages per iteration and accesses the local data structure more efficiently on current computers than does the multicolor SOR method.

For the five-point stencil model problem, we have proved for both strip and block partitions that both PSOR and SOR have the same asymptotic convergence rate as the rowwise ordering. We also proved a theorem that shows, in general, that $\hat{A}$ will be consistently ordered whenever A is consistently ordered. Here A and $\hat{A}$ are defined in (3.1) and (4.1), respectively. For the nine-point stencil model problem, we also proved a theorem that shows that PSOR has the same asymptotic rate of convergence as both the rowwise ordering and the R/B/G/O ordering for the strip partitions. Since PSOR requires less communication and accesses the local data structure more efficiently, it can be used as an alternative to R/B/G/O as a parallel method.

```
c- Strip partition in Y-axis direction
c  m: even number
c  p: the number of processors
      h = 1 / (m + 1)
      m_p = m / p

c- Update the first grid line of strip
      j = 1
      do i = 1, m
        u(i,j) = ......
      enddo

c- Exchange the updated u on the
c  first grid line of strip
      .......

c- Update the remainder of strip
      do j = 2, m_p
       do i = 1, m
        u(i,j) = ......
       enddo
      enddo

c- Exchange the updated u on the
c  last grid line of strip
      .......
```

FIG. 5.9. *Illustration of PSOR program.*

```
c-- On red points
      jrb = 1
      do j = 1, m_p
       jrb = -jrb
       idel = (jrb + 1)/2
       i0 = idel + 1
       do i = i0, m, 2
        u(i,j) = ......
       enddo
      enddo
c- Exchange the updated u on the first
c  and last grid lines of strip.
      ......
c-- On black points
      jrb = -1
      do j = 1, m_p
       jrb = -jrb
       idel = (jrb + 1)/2
       i0 = idel + 1
       do i = i0, m, 2
        u(i,j) = ......
       enddo
      enddo
c- Exchange the updated u on the first
c  and last grid lines of strip.
      ......
```

FIG. 5.10. *Illustration of R/B SOR program.*

We demonstrated the parallel performance of the PSOR method for the five-point stencil on four distinct MIMD multiprocessor computers (a KSR1, the Intel Delta, an Intel Paragon, and an IBM SP2). Numerical results showed that PSOR is very efficient on these machines. They also confirm that PSOR, either on a strip partition or a block partition, has the same asymptotic rate of convergence as the natural rowwise ordering.

We compared the parallel performance of the PSOR method versus the R/B SOR method for the five-point stencil and the R/B/G/O SOR method for the nine-point stencil on the Paragon. Numerical results point to the effectiveness of PSOR in both computation and interprocessor data communication. The results also showed that PSOR and R/B/G/O SOR on strip partitions have the same asymptotic rate of convergence as expected from our last theorem.

Finally, PSOR may be advantageous for dealing with complicated scientific and engineering problems such as irregular geometries, high orders of discretization, and local grid refinement because it is based on a domain decomposition. Also, since PSOR can use an ordering within each subdomain that closely resembles the natural rowwise ordering (especially for strip partitions), we can expect a parallel SSOR defined by using PSOR to be an effective parallel preconditioner for the conjugate gradient method. We intend to compare PSOR–SSOR with SSOR based on the natural ordering in a preconditioning context in a subsequent paper.

Acknowledgments. The first author would like to thank his advisor, Professor L. Ridgway Scott, for valuable discussions and his constant support. He is also grateful to Professor Tamar Schlick for her support. Access to the KSR1, the Intel Delta and Paragon, and an IBM SP2 has been provided, respectively, by the Texas Center for Advanced Molecular Computation at the University of Houston, the Center for Advanced Computing Research at Caltech, and the Cornell Theory Center at Cornell University.

REFERENCES

[1] L. M. Adams and J. M. Ortega, *A multi-color SOR method for parallel computation*, in Proceedings of the 1982 International Conference on Parallel Processing, Bellaire, MI, pp. 53–58.

[2] L. M. Adams and H. F. Jordan, *Is SOR color-blind?* SIAM J. Sci. Statist. Comput., 7 (1986), pp. 490–506.

[3] L. M. Adams, R. J. LeVeque, and D. M. Young, *Analysis of the SOR iteration for the 9-point Laplacian*, SIAM J. Numer. Anal., 25 (1988), pp. 1156–1180.

[4] B. Bagheri, T. W. Clark and L. R. Scott, *A parallel dialect of* Fortran, Fortran Forum, 11 (1992), pp. 20–31.

[5] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[6] U. Block, A. Frommer, and G. Mayer, *Block coloring schemes for the SOR method on local memory parallel computers*, Parallel Comput., 14 (1990), pp. 61–75.

[7] S. C. Eisenstat, *Comments on scheduling parallel iterative methods on multiprocessor systems* II, Parallel Comput., 11 (1989), pp. 241–244.

[8] L. Elsner, *Comparisons of weak regular splittings and multisplitting methods*, Numer. Math., 56 (1989), pp. 283–289.

[9] D. J. Evans, *Parallel SOR iterative methods*, Parallel Comput., 1 (1984), pp. 3–18.

[10] L. A. Hageman and D. M. Young, *Applied iterative methods*, Academic Press, New York, 1981.

[11] W. Kahan, *Gauss–Seidel Methods of Solving Large Systems of Linear Equations*, Ph.D. thesis, University of Toronto, Toronto, ON, Canada, 1958.

[12] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, translated and revised from the 1991 German original, Appl. Math. Sci. 95, Springer-Verlag, New York, 1994.

[13] D. L. Harrar II, *Orderings, multicoloring, and consistently ordered matrices,* SIAM J. Matrix Anal. Appl., 14 (1993), pp. 259–278.

[14] D. P. O'Leary and R. E. White, *Multi-splittings of matrices and parallel solution of linear systems,* SIAM J. Alg. Discrete Methods, 6 (1985), pp. 630–640.

[15] R. G. Melhem, *Determination of stripe structures for finite element matrices*, SIAM J. Numer. Anal., 24 (1987), pp. 1419–1433.

[16] R. G. Melhem and K. V. S. Ramarao, *Multicolor reordering of sparse matrices resulting from irregular grids*, ACM Trans. Math. Software, 14 (1988), pp. 117–138.

[17] N. M. Missirlis, *Scheduling parallel iterative methods on multiprocessor systems,* Parallel Comput., 5 (1987), pp. 295–302.

[18] M. Neumann and R. J. Plemmons, *Convergence of parallel multisplitting iterative methods for M-matrices*, Linear Algebra Appl., 88/89 (1987), pp. 559–573.

[19] W. Niethammer, *The SOR method on parallel computers.* Numer. Math, 56 (1989), pp. 247-254.

[20] J. M. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM J. Optim., 1 (1991), pp. 565–582.

[21] J. M. Ortega and R. G. Voigt, *Solution of partial differential equations on vector and parallel computers*, SIAM Rev., 27 (1985), pp. 149–270.

[22] A. M. Ostrowski, *On the linear iteration procedures for symmetric matrices*, Rend. Mat. Appl., 14 (1954), pp. 140–163.

[23] N. R. Patel and H. F. Jordan, *A parallelized point row-wise successive over-relaxation method on a multiprocessor*, Parallel Comput., 1 (1984), pp. 207–222.

[24] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, 1987.

[25] Y. Robert and D. Trystram, *Comments on scheduling parallel iterative methods on multiprocessor systems,* Parallel Comput., 7 (1988), pp. 253–255.

[26] K. Stüen and U. Trottenberg, *Multigrid methods: fundamental algorithms, model problem analysis and applications,* in Multigrid Methods, Lecture Notes in Math. 960, Springer-Verlag, Berlin, New York, 1982, pp. 1–176.

[27] University of Tennessee, *MPI: A Message-Passing Interface Standard*, Version 1.0, May 5, 1994.

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

[29] J. Wang, *Convergence analysis without regularity assumptions for multigrid algorithms based on SOR smoothing*, SIAM J. Numer. Anal., 29 (1992), pp. 987–1001.

[30] R. E. White: *Multisplittings and parallel iterative methods*, Comput. Methods Appl. Mech. Engrg., 64 (1987), pp. 567–577.

[31] R. E. White, *Multisplitting with different weighting schemes,* SIAM J. Matrix Anal. Appl., 10 (1989), pp. 481–493.
[32] R. E. White, *Multisplitting of a symmetric positive definite matrix*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 69–82.
[33] D. Xie and L. R. Scott, *The Parallel U-Cycle Multigrid Method*, UH/MD Technical report 240, University of Houston, Houston, TX, 1997.
[34] D. Xie, *New Parallel Iteration Methods, New Nonlinear Multigrid Analysis, and Application in Computational Chemistry*, Ph.D. thesis, UH/MD Research report 208, University of Houston, Houston, TX, 1995.
[35] I. Yavneh, *On red-black SOR smoothing in multigrid*, SIAM J. Sci. Comput., 17 (1996), pp. 180–192.
[36] D. M. Young, *Iterative Solution of Large Linear System*, Academic Press, New York, 1971.