SOLVING BLOCK LINEAR SYSTEMS WITH LOW-RANK OFF-DIAGONAL BLOCKS IS EASILY PARALLELIZABLE

V. MEŃKOV
DEPARTMENT OF COMPUTER SCIENCE
INDIANA UNIVERSITY – BLOOMINGTON*

Abstract. An easily and efficiently parallelizable direct method is given for solving a block linear system $Bx = y$, where $B = D + Q$ is the sum of a non-singular block diagonal matrix D and a matrix Q with low-rank blocks. This implicitly defines a new preconditioning method with an operation count close to the cost of calculating a matrix-vector product Qw for some w, plus at most twice the cost of calculating $D^{-1}w$ for some w. When implemented on a parallel machine the processor utilization can be as good as that of those operations. Order estimates are given for the general case, and an implementation is compared to block SSOR preconditioning.

1. Introduction. Solving a large linear system of equations

$$Ax = y \tag{1}$$

is a common subproblem encountered in the numerical solution of differential and integral equations. Such linear systems are usually solved with preconditioned conjugate gradient (PCG) like methods, which involve solving an auxiliary system

$$Cx = y, \tag{2}$$

with a preconditioner C on each iteration.

This paper examines parallel preconditioners with low-rank off-diagonal blocks: the BSSOR LOB preconditioner

$$C_1 = (D + Q_L)D^{-1}(D + Q_U), \tag{3}$$

and the direct LOB preconditioner

$$C_2 = D + Q. \tag{4}$$

Here D is a non-singular block diagonal matrix; Q_L and Q_U are block strictly lower and upper triangular matrices composed of low-rank blocks; and Q has zero diagonal blocks and low-rank off-diagonal blocks (ODBs).

Both preconditioners require solving one or two block linear systems of the form

$$Bx = y, \tag{5}$$

where the non-singular $n \times n$ matrix B has the splitting $B = D+Q$ with a non-singular block-diagonal matrix $D = \mathrm{diag}\{D_{11}, D_{22}, \ldots, D_{pp}\}$ and some block matrix Q. [1]

* WORK SUPPORTED BY NSF GRANTS CDA-9303189, ASC-9502292 AND ROME LABS AF 30602-92-C-0135

[1] In fact, the preconditioner C_1 (3) can be more efficiently dealt with as a product of two matrices ($C_1 = B_1B_2$, with $B_1 = (D + Q_L)D^{-1}$ and $B_2 = D + Q_U$), only one of which (B_2) is of the form $D + Q$. But the system with the matrix B_1 (which has the form $(D + Q)D^{-1}$, rather than $D + Q$) can be solved in almost the same way as (5); one only needs to remove step 5 from the SMW solve algorithm (Sec. 5, p. 5).

Block diagonal preconditioning (C_2 with $Q = 0$) is typically used in parallel computations. If diagonal blocks are assigned to processors, the preconditioner can be applied without interprocessor synchronization or communication. Unfortunately, block diagonal preconditioned systems often fail to converge, especially for non-symmetric A. Block symmetric Gauss-Seidel (C_1 with Q_L and Q_U the corresponding parts of A) is significantly more robust, but the block lower and upper triangular system solves are inherently sequential. One way to recover parallelism is to increase the number of blocks in the partitioning of A and use a level scheduling [1, 2]. however, this also causes the quality of preconditioning to fall, and in the limiting case becomes pointwise symmetrized Gauss-Seidel, which typically provides poor quality of preconditioning.

In this paper we borrow an idea from the solution of integral equations, and use low-rank approximations (LRA) of the off-diagonal blocks. In [3] the characteristics of such an approximation are analyzed; this paper shows how such a preconditioner can be implemented with parallelism almost as good as that of block diagonal preconditioning. By varying the rank of the ODBs the new method can be parametrized to vary in quality between that of block diagonal and block SSOR [11] preconditioner.

We assume that the n variables are divided into p blocks (so that B and other matrices of the same dimension consist of $p \times p$ blocks). A typical source of the blocking would be from domain decomposition, with $p \ll n$.

Let $r_{kl} = \operatorname{rank} Q_{kl}$, let $M = \sum_{k,l} r_{kl}$ be the sum of ranks of all blocks of Q, let $m_l = \sum_k r_{kl}$ be the sum of ranks of the blocks of Q in the l-th block column, and let m be the maximum of the sum of the ranks of the blocks in any one block column or block row of Q. If no block of Q has rank higher than one, then M becomes the number of non-zero blocks in Q, and m the maximum number of non-zero blocks in any one row or column of Q. Let $\mathcal{P}$ be the set of all the pairs (k, l) such that $Q_{kl} \neq 0$.

Each block of Q can be represented as

$$Q_{kl} = \sum_{i=1}^{r_{kl}} u_{kl}^i {v_{kl}^i}^{\mathrm{T}} = U_{kl} V_{kl}^T, \tag{6}$$

with $U_{kl} = [\, u_{kl}^1 \quad u_{kl}^2 \quad \cdots \quad u_{kl}^{r_{kl}} \,]$ and $V_{kl} = [\, v_{kl}^1 \quad v_{kl}^2 \quad \cdots \quad v_{kl}^{r_{kl}} \,]$.

Since (5) is used in iterative solving of (1), we assume that problem (5) needs to be solved for many right-hand sides y, which are not available at the same time. Blocks of D and Q are constructed and represented to allow: (1) solving systems with the diagonal blocks D_{kk}, and (2) performing matrix-vectors multiplications with the blocks Q_{kl}. E.g. each block of D is represented as a product of a lower and an upper triangular matrices obtained by incomplete LU factorization of the diagonal blocks of A.

The goal is a direct preconditioner whose operation count is comparable to that of any method that uses the two above-mentioned operations (such as, e.g., the usual block elimination method, in the case when B is block-triangular), but which can be efficiently parallelized.

2. Data representation and distribution. The data are distributed among the processors with

- D_{kk} stored on processor k;
- U_{kl} (composed of the vectors $\{u_{kl}^s\}_{s=1,\ldots,r_{kl}}$) stored on processor k;
- V_{kl} (composed of the vectors $\{v_{kl}^s\}_{s=1,\ldots,r_{kl}}$) is stored on processor l.

Here processor k may be a virtual processor, and if fewer than p processors are available, several virtual processors are mapped to one physical processor.

3. Representation of B^{-1}. From (6), $Q = UV^{\mathrm{T}}$, with M-column matrices U and V:

$$(7)\quad U = \begin{bmatrix} U_{11} & 0 & \cdots & 0 & U_{12} & 0 & \cdots & 0 & \cdots & U_{1p} & 0 & \cdots & 0 \\ 0 & U_{21} & \cdots & 0 & 0 & U_{22} & \cdots & 0 & \cdots & 0 & U_{2p} & \cdots & 0 \\ \vdots & & \ddots & \vdots & \vdots & & \ddots & \vdots & \cdots & \vdots & & \ddots & \vdots \\ 0 & & 0 & U_{p1} & 0 & & 0 & U_{p2} & \cdots & 0 & & 0 & U_{pp} \end{bmatrix}$$

$$(8)\quad V = \begin{bmatrix} V_{11} & V_{21} & \cdots & V_{p1} & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & V_{12} & V_{22} & \cdots & V_{p2} & \cdots & \vdots & 0 & & \vdots \\ \vdots & & & & \vdots & & & & \cdots & 0 & & & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & V_{1p} & V_{2p} & \cdots & V_{pp} \end{bmatrix}$$

The block column ordering shown above is chosen so that G will have the same block structure as Q, see Section 4. Block columns in (7) and (8) corresponding to zero blocks Q_{kl} are absent (have no columns).

Since $B = D + UV^{\mathrm{T}}$, the Sherman–Morrison–Woodbury formula [10] gives

$$(9)\qquad B^{-1} = (D + UV^{\mathrm{T}})^{-1} = D^{-1} - D^{-1}U(I + G)^{-1}V^{\mathrm{T}}D^{-1},$$

with $G = V^{\mathrm{T}}D^{-1}U$ of order M.

Since the non-zero eigenvalues of G are the same as those of $D^{-1}UV^{\mathrm{T}}$,

$$\sigma(I + G) \setminus \{1\} = \sigma(D^{-1}B) \setminus \{1\},$$

where $\sigma(H)$ is the set of eigenvalues of the square matrix H. Since B is non-singular, so is $I+G$. Furthermore, if $M < n$ (i.e. if the ranks of the blocks of Q are low enough), then $\operatorname{rank}(UV^{\mathrm{T}}) \le M < n$, and $1 \in \sigma(D^{-1}B)$. Therefore $\sigma(I + G) \subset \sigma(D^{-1}B)$, and $\operatorname{cond}(I + G) \le \operatorname{cond}(D^{-1}B)$.

4. Structure of G. The following proposition shows that the block sparsity pattern of G is contained in that of Q:

PROPOSITION 4.1. *If $Q_{kl} = 0$, then $G_{kl} = 0$.*

Proof. G can be partitioned as

$$G = \begin{bmatrix} G_{11} & G_{12} & \cdots & G_{1p} \\ \vdots & & \vdots & \\ G_{p1} & G_{p2} & \cdots & G_{pp} \end{bmatrix},$$

where an $m_k \times m_l$ block G_{kl} has the structure

$$\begin{aligned} G_{kl} &= \begin{bmatrix} 0 & \cdots & 0 & V_{1k}^{\mathrm{T}} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & V_{pk}^{\mathrm{T}} & 0 & \cdots & 0 \end{bmatrix} \cdot \begin{bmatrix} D_{11}^{-1}U_{11} & 0 & \cdots & 0 \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & D_{pp}^{-1}U_{p1} \end{bmatrix} \\ &= \begin{bmatrix} 0 & \cdots & 0 & H_{1kl} & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & H_{pkl} & 0 & \cdots & 0 \end{bmatrix}, \end{aligned} \qquad (10)$$

with zeros in the first $k-1$ block columns, and the last $p-k$ block columns, and $H_{jkl} = V_{jk}^{\mathrm{T}} D_{kk}^{-1} U_{kl}$ of size $r_{jk} \times r_{kl}$. The equation for H_{jkl} shows that the block G_{kl} can be non-zero only if U_{kl} is non-zero, which, in its turn, can be non-zero only if Q_{kl} is non-zero. $\square$

If we use *no-cancellation assumption* [6, Section 2.2.2] (i.e., disregard the possibility that a potentially non-zero element becomes zero simply because a given dot product happens to be zero), it follows from Proposition 4.1 that G has the same non-zero block structure as Q.

COROLLARY 4.2. *If Q is strictly block upper or lower triangular, then so is G.*

Since H_{jkl} is $r_{jk} \times r_{kl}$, the total number of non-zero elements in G is $\mathrm{nnz}\,(G) = \sum_k \left((\sum_j r_{jk})(\sum_l r_{kl}) \right) = \sum_{kl} (r_{kl} m_k) \leq mM$.

If no block of Q has rank higher than one, then $mM \leq p^3$. If Q is block-tridiagonal, with all non-zero blocks of rank one, then $mM < 3p$.

5. Outline of the solution method. The parallel solution method for solving $Bx = y$ is based on (9), and will be referred to later as the SMW-based method.

Computing the preconditioner.
begin
1. **foreach** $k = 1 : p$ *(in parallel)*
Preprocess D_{kk} for solving $D_{kk} x_k = y_k$ (if necessary)
endfor
2. **foreach** $k = 1 : p$ *(in parallel)* */* Compute entries of G */*
for $l = 1 : p$ *such that* $(k,l) \in \mathcal{P}$
2a. *Solve $D_{kk} \tilde{U}_{kl} = U_{kl}$*
for $j = 1 : p$ *such that* $(j,k) \in \mathcal{P}$
2b. *Set $H_{jkl} = V_{jk}^{\mathrm{T}} \tilde{U}_{kl}$*
endfor
endfor
endfor
2c. If desired, gather all block rows of G on a single processor
3. LU-factor $I + G$
end

Computing G consists of two steps. On step 2a, processor k computes $\tilde{U}_{kl}$ for all l's by solving the linear system

$$D_{kk} \left[\tilde{U}_{k1} \quad \tilde{U}_{k2} \quad \cdots \quad \tilde{U}_{kp} \right] = \left[U_{k1} \quad U_{k2} \quad \cdots \quad U_{kp} \right]$$

with $\sum_l r_{kl}$ right-hand sides. On step 2b, processor k computes the blocks G_{kl} for all values l, i.e. the blocks H_{jkl} in the matrix

$$\begin{bmatrix} H_{1k1} & \cdots & H_{1kp} \\ H_{2k1} & \cdots & H_{2kp} \\ \vdots & & \vdots \\ H_{pk1} & \cdots & H_{pkp} \end{bmatrix} = \begin{bmatrix} V_{1k}^{\mathrm{T}} \\ V_{2k}^{\mathrm{T}} \\ \vdots \\ V_{pk}^{\mathrm{T}} \end{bmatrix} \cdot \left[\tilde{U}_{k1} \quad \tilde{U}_{k2} \quad \cdots \quad \tilde{U}_{kp} \right]$$

As a result of this step, G is distributed among the processors by block row. A gather step is needed if one wants G to be stored on a single processor.

Unless G is triangular, some LU-factorization of $I + G$ is needed (step 3). If G is stored on a single processor, factoring is done by that processor; otherwise, it is done in parallel.

Applying the preconditioner. Applying the preconditioner (4) or (3) involves solving one or two systems of the form (5), using formula (9):

begin **Solving** $Bx = y$

1. **foreach** $k = 1 : p$ *(in parallel)* /* $z = D^{-1}y$ */
  Solve $D_{kk}z_k = y_k$
endfor

2. **foreach** $l = 1 : p$ *(in parallel)* /* $t = V^{\mathrm{T}}z$ */
  for $k = 1 : p$
    Set $t_{kl} = V_{kl}^{\mathrm{T}} z_l$
  endfor
endfor

3. Solve $(I + G)s = t$ /* $s = (I + G)^{-1}\ t$ */

4. **foreach** $k = 1 : p$ *(in parallel)* /* $y' = y - Us$ */
  Set $y'_k = y_k - \sum_l U_{kl}s_{kl}$
endfor

5. **foreach** $k = 1 : p$ *(in parallel)* /* $x = D^{-1}y'$ */
  Solve $D_{kk}x_k = y'_k$
endfor

end

Step 3 solves

$$(I + G)s = t, \tag{11}$$

where s is partitioned conformally with t.

Only step 3 involves interprocessor communications. On step 2, the j-th processor produces all components of t in its block column (t_{kj} for all k); on step 4, the j-th processor needs all components of s in its block row (s_{jl} for all l). Depending on how (11) is solved, step 3 can be implemented in different ways:

1. Store the entire factors of $I+G$ on a single processor, which solves the system (11). First, t is gathered on this processor; after solving, components of s are scattered to the processors that will need them at the next step. Both steps 2 and 4 send M floating-point values between the processors.
   What processor should be used for solving (11)? Normally, it can be one of the processors $1, \ldots, p$ that perform all other operations. However, if minimizing the memory allocation per processor is a priority, and there are extra processors available, it may be expedient to use $p + 1$ processors, the extra $(p + 1)$-st processor being used to store the factors of $I + G$ and solve (11).
2. Have each processor, redundantly, solve (11). (This option was used in the test.)
3. Use a parallel method for solving (11), distributing the factors of $I + G$ among all processors. At most pM floating-point numbers are sent between processors. Factors of $I + G$ can be distributed among the p processors by block row. If this scheme results in poor data balance (as in the case of a block-triangular or block-tridiagonal Q), linear speed-up can be achieved by distributing the factors of $I + G$ cyclically by row.

The preconditioner can be applied with one D-solve per iteration instead of two, by replacing steps 4 and 5 with

4′. **foreach** $k = 1 : p$ *(in parallel)* /* $x = z - \tilde{U}\ s$ */
  Set $x_k = z_k - \sum_l \tilde{U}_{kl}s_{kl}$

endfor

where $\tilde{U} = D^{-1}U$ (see step 2a of the preconditioner-preparing algorithm). The disadvantage is that the DAXPY operation in $4'$ is done with dense vectors, while that in 5 uses sparse ones. In the remainder of the article, we use 4 and 5 instead of $4'$.

6. Operation count analysis. Solving (5) requires the following number of operations:

$$C_B = C_D + C_V + C_G + C_U + C_D = 2C_D + C_m + C_G.$$

The five terms of the first sum correspond to the five steps of the algorithm. C_D is for solving $Dz = w$, C_V is for one local dot product for each column of the blocks V_{kl}, C_U is for one DAXPY with each column of the blocks U_{kl}'s, and C_G is for solving (11). The sum $C_m = C_V + C_U$ is the number of operations needed to perform a matrix-vector multiplication $Qw = UV^{\mathrm{T}}w$.

7. Timing models. During solving $Bx = y$, only solving system (11) requires interprocessor communications; all other operations are parallel. From the operation count data in Section 6, the time t_B the method needs to solve (5) is approximately the following:

$$t_B = 2t_D + t_V + t_U + t_G = 2t_D + t_m + t_G. \tag{12}$$

The three main terms here, t_D, t_V, and t_U, are times this parallel computer will take, respectively

t_D: to solve $Dz = w$,

t_V: to calculate a dot product with each column of each block V_{kl},

t_U: to perform a DAXPY operation with each column of each block U_{kl},

with the matrices distributed as described in Section 2. The sum $t_m = t_V + t_U$ is the time needed to calculate a matrix-vector product $Qw = UV^{\mathrm{T}}w$.

Cost of solving (11). The term t_G is the time needed to solve (11). When Q is strictly block triangular, so is G, and $t_G = O(Mm)$.

When the matrix G is not triangular, it still does not depend on the right-hand side y, and so $I+G$ needs to be LU-factored once for all future B-solves. The sequential solve time t_G^{seq} is $O(M^2)$; parallel solve time $t_G^{\mathrm{par}} = O(mM)$. This t_G can be reduced if the block structure of Q is such such that fill-in is limited in some way. E.g., for block-tridiagonal Q, t_G^{seq} is $O(mM)$ and $t_G^{\mathrm{par}} = O(M \max\{1, m/p\})$.

Costs of preparing G. Computing G involves solving M linear systems of the form $D_{jj}\tilde{u}_{ji} = u_{ji}$, and then calculating no more than Mm dot products of the form $v_{kj}^{\mathrm{T}}\tilde{u}_{ji}$. The total number of operations involved is under $m(C_D + C_U) \approx \frac{m}{2}C_B$. Since both D-solves and dot products can be done in parallel,

$$t_{\mathrm{prepG}} \approx \frac{m}{2}t_B. \tag{13}$$

The cost of LU-factoring the matrix $I+G$ depends greatly on the structure of this matrix (and thus on the structure of Q). For a general Q, the operation count for this factoring is, is the worst case, $C_{\mathrm{fact(I+G)}} = \mathcal{O}(M^3)$, which implies $t^{\mathrm{seq}}_{\mathrm{fact(I+G)}} = \mathcal{O}(M^3)$ for sequential factoring, and

$$t^{\mathrm{par}}_{\mathrm{fact(I+G)}} = \mathcal{O}(mM^2) \tag{14}$$

for factoring on p processors in parallel (the p-th processor calculates the p-th block row of the factors). This result may be better if Q has some special structure making LU-factoring G less expensive than $\mathcal{O}(M^3)$ operations. For example, if Q is block-tridiagonal, $t^{\text{seq}}_{\text{fact(I+G)}} = \mathcal{O}(m^2 M)$ and

$$t^{\text{par}}_{\text{fact(I+G)}} = \mathcal{O}(mM \max 1, m/p). \tag{15}$$

The timing model provides a practical definition of "low-rank" for blocks of Q:

(a) The time spent preparing and factoring $I+G$ must be small compared to the iterative solve time for (1), and

(b) t_G must be small relative to $t_D + t_U$, so that $t_B \approx 2t_D + t_m$.

Below we will show what restrictions this conditions impose on m, for a general matrix Q, as well as for the special cases of block-triangular and block-tridiagonal Q. We suppose that:

1. p processors are used.
2. The maximum dimension of a block D_{kk} is $\mathcal{O}(n/p)$.
3. $m = \mathcal{O}(M/p)$, i.e. the maximum sum of block ranks in a block row of Q is of the same order as the average sum.
4. Factoring $I+G$ and solving (11) is done in parallel on p processors. (Estimates for sequential operations can be obtained in a similar way.)
5. t_D is the dominant term in $t_D + t_m$, i.e. $t_D + t_m = \mathcal{O}(t_D)$. This is usually the case with many matrices; and if off-diagonal block computations are expensive relative to the diagonal block computations, i.e. if $t_m > pt_D$, then parallelizing the solution of the system (5) is easy.

The time t_D spent solving the system $Dz = w$ depends greatly on the structure of D. The two obvious limiting cases are

- $t_D = \Theta(n/p)$ (D_{kk} is a diagonal matrix, or a product of LU-factors with few non-zeros per row);
- $t_D = \Theta(n^2/p^2)$ (D_{kk} is a product of dense L- and U-factors).

Estimates below will be made for a Q of arbitrary block structure, as well as for block-triangular and block-tridiagonal Q.

Equation (13) indicates that t_{prepG} is on the order of the time taken by m iterations. Thus the criterion (a) implies that m needs to be small in comparison to the total number of iteration.

In order for the term $t_{\text{fact(I+G)}}$ not to make this ratio much worse, it needs to be of the same order as t_{prepG}, i.e. the condition

$$t_{\text{fact(I+G)}} = \mathcal{O}(mt_D) \tag{16}$$

should hold. For a block-triangular Q, this is not an issue (no factoring is needed); for a general Q, the parallel LU-factoring time estimate (14) is used to rewrite the condition (16) as

$$m = \begin{cases} \mathcal{O}\left(\frac{n^{1/2}}{p^{3/2}}\right), & \text{if } t_D = \Theta\left(n/p\right), \\ \mathcal{O}\left(\frac{n}{p^2}\right), & \text{if } t_D = \Theta\left(n^2/p^2\right) \end{cases} \tag{17}$$

For a block-tridiagonal Q, the parallel LU-factoring time estimate (15) gives

$$m = \begin{cases} \mathcal{O}\left(\min\{\frac{n}{p^2}, \frac{n^{1/2}}{p^{1/2}}\}\right), & \text{if } t_D = \Theta\left(n/p\right), \\ \mathcal{O}\left(\min\{\frac{n^2}{p^3}, \frac{n}{p}\}\right), & \text{if } t_D = \Theta\left(n^2/p^2\right). \end{cases} \tag{18}$$

If Q is ...	To ensure that...	If $t_D = \Theta(n/p)$	If $t_D = \Theta(n^2/p^2)$
arbitrary	$t_{\text{fact(I+G)}} = \mathcal{O}(mt_D)$	$m = \mathcal{O}\left(\frac{n^{1/2}}{p^{3/2}}\right)$	$m = \mathcal{O}\left(\frac{n}{p^2}\right)$
	$t_G = o(t_D + t_m)$	$m = o\left(\frac{n^{1/2}}{p}\right)$	$m = o\left(\frac{n}{p^{3/2}}\right)$
blk-tridiag.	$t_{\text{fact(I+G)}} = \mathcal{O}(mt_D)$	$m = \mathcal{O}\left(\min\{\frac{n}{p^2}, \frac{n^{1/2}}{p^{1/2}}\}\right)$	$m = \mathcal{O}\left(\min\{\frac{n^2}{p^3}, \frac{n}{p}\}\right)$
	$t_G = o(t_D + t_m)$	$m = o\left(\min\{\frac{n}{p^2}, \frac{n^{1/2}}{p^{1/2}}\}\right)$	$m = o\left(\min\{\frac{n^2}{p^3}, \frac{n}{p}\}\right)$
blk-triang.	$t_G = o(t_D + t_m)$	$m = o\left(\min\{\frac{n}{p^2}, \frac{n^{1/2}}{p^{1/2}}\}\right)$	$m = o\left(\min\{\frac{n^2}{p^3}, \frac{n}{p}\}\right)$

TABLE 1

Low-rank criteria (in the case of parallel solving of $(I + G)s = t$).

If Q is ...	To ensure that...	If $t_D = \Theta(n/p)$	If $t_D = \Theta(n^2/p^2)$
arbitrary	$t_{\text{fact(I+G)}} = \mathcal{O}(mt_D)$	$m = \mathcal{O}\left(\frac{n^{1/2}}{p^2}\right)$	$m = \mathcal{O}\left(\frac{n}{p^{5/2}}\right)$
	$t_G = o(t_D + t_m)$	$m = o\left(\frac{n^{1/2}}{p^{3/2}}\right)$	$m = o\left(\frac{n}{p^2}\right)$
block-tridiagonal	$t_{\text{fact(I+G)}} = \mathcal{O}(mt_D)$	$m = \mathcal{O}\left(\frac{n^{1/2}}{p}\right)$	$m = \mathcal{O}\left(\frac{n}{p^{3/2}}\right)$
	$t_G = o(t_D + t_m)$	$m = o\left(\frac{n^{1/2}}{p}\right)$	$m = o\left(\frac{n}{p^{3/2}}\right)$
block-triangular	$t_G = o(t_D + t_m)$	$m = o\left(\frac{n^{1/2}}{p}\right)$	$m = o\left(\frac{n}{p^{3/2}}\right)$

TABLE 2

Low-rank criteria (in the case of sequential solving of $(I + G)s = t$).

Restrictions on m that are necessary to satisfy criterion (b) can be derived from the estimates of t_G given earlier in this section.

These results for parallel factoring of $I + G$ and parallel $(I + G)$-solve are summarized in Table 1. Table 2 presents analogously obtained results for sequential LU-factoring and $(I + G)$-solve.

If the symbol o in the restrictions on m is replaced with $\mathcal{O}$, then instead of $t_B \approx 2t_D + t_m$ we will have $t_B = \mathcal{O}(t_D + t_m)$.

8. Preliminary computational results. The direct LOB preconditioner, applied using the SMW-based algorithm in conjugate gradients stabilized [4] iteration was implemented in pC++ [5] on an SGI Power Challenge. Only sequential solving of $(I + G)s = t$ was implemented in this version.

Two test problems with $p = 8$, coming from a 2D finite element problem in CFD, were solved using a number of preconditioners:

1. BSSOR LOB ($C = (Q_L + D)D^{-1}(Q_U + D)$. Blocks of the strict lower and upper triangular Q_L and Q_U are obtained by a lumping-based low-rank approximation method [3] of the respective blocks of A, with rank $Q_{kl} \leq 3$). The preconditioner is applied using a conventional LU-solve method.
2. Direct LOB ($C = D + (Q_L + Q_U)$, with the same Q_L and Q_U as above), applied using the SMW method.
3. BSSOR with original ODB ($C = (L + D)D^{-1}(U + D)$; strict lower and upper triangular L and U are composed from the ODBs of A);
4. Direct LOB preconditioner with original ODBs ($C = D + (L + U)$ with the same L and U as above), applied using the SMW-based method.

In all preconditioners, blocks of D were obtained by an incomplete LU-factorization [9] of the diagonal blocks of A with 3 levels of fill allowed. The system was first

partitioned into eight subdomains using a spectral method from Chaco [7, 8].

In Tables 3 and 4, the first number in each cell is the total time spent to solve the problem on the specified number of processors with the specified preconditioner; it is followed by the preconditioner preparation time (LU-factoring D, generating Q, generating and factoring $I + G$) plus the number of iterations times the time per iteration. All times are in seconds.

Table 5 presents partial breakdown of SMW iteration time. For four iterative processes, it gives the total time spent during the iterations, as well as the amount of time that was spent solving $Dz = w$ and solving $(I + G)s = t$. (The rest of the time was spent mainly doing matrix-vector multiplications).

Data in Tables 3 and 4 show that in the low rank case, LRA SMW has a parallel efficiency of 0.91 or better, compared to 0.19 of block SSOR. For higher ranks, LRA SMW only slightly increases the parallel efficiency from 0.20 to 0.22 or 0.26; the main reason for this is that, as Table 5 shows, while D-solve time decreases as the inverse of the number of processors, the $(I + G)$-solve time stays constant. Thus, when M is large, parallel $(I + G)$-solve needs to be implemented.

Further note that fewer iterations are required for LRA SMW, indicating that it provides a better quality preconditioner. These results confirm that LRA SMW can provide parallel efficiency while maintaining and even improving the quality of preconditioning that block SSOR provides.

REFERENCES

[1] F. Alvarado, *Ordering schemes for partitioned sparse inverses*, 1989. SIAM Symposium on Sparse Matrices, Salishan Lodge, Gleneden Beach, OR, May 22–24 1989.

[2] E. Anderson, *Parallel implementation of preconditioned conjugate gradient methods for solving sparse systems of linear equations*, Master's thesis, Univ. of Illinois at Urbana-Champaign, Center for Supercomputing Res. & Dev., 1988.

[3] R. Bramley and V. Meñkov, *Low rank off-diagonal block preconditioners for solving sparse linear systems on parallel computers*, Tech. Rep. 446, Department of Computer Science, Indiana University, Bloomington, IN, 1996. (To be submitted).

[4] H. V. der Vorst, *Bi–CGSTAB: A fast and smoothly converging variant of Bi–CG for the solution of nonsymmetric linear systems*, SIAM Journal of Scientific and Statistical Computing, 13 (1992), pp. 631–644.

[5] D. Gannon, S. X. Yang, and P. Beckman, *User Guide for a Portable Parallel C++ Programming System, pC++*, Department of Computer Science and CICA, Indiana University, Bloomington, IN, 1994. Available via World Wide Web at http://www.extreme.indiana.edu/sage/pcxx_ug/pcxx_ug.html.

[6] A. George and W.-H. Liu, *The Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Engelwood Cliffs, NJ, 1981.

[7] B. Hendrickson and R. Leland, *The Chaco User's Guide Version 1.0*, Tech. Rep. SAND 93-2339, Sandia National Laboratories, Albuquerque, N.M., October 1993.

[8] ———, *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*, SIAM Journal on Scientific Computing, (1995), pp. 452–469.

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

[10] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, New York, 1970.

[11] A. Yeremin and L. Kolotilina, *On a family of two–level preconditionings of the incomplete block factorization type*, Sov. J. Numer. Anal. Math. Modeling, 1 (1986), pp. 293–320.

Num. proc.	SSOR, low-r. ODB (M=84)	LRA SMW, low-r. ODB (M=84)	SSOR, orig. ODB	LRA SMW, orig. ODB (M=1234)
1	$95.3 = 8.6 + +67 \cdot 1.29$	$78.8 = 10.8 + +52 \cdot 1.31$	$40.9 = 7.9 + +23 \cdot 1.44$	$78.3 = 64.7 + +6 \cdot 2.26$
2	$67.5 = 4.7 + +60 \cdot 1.05$	$39.1 = 5.9 + +53 \cdot 0.63$	$28.7 = 4.1 + +23 \cdot 1.07$	$55.8 = 46.3 + +6 \cdot 1.59$
4	$64.2 = 3.6 + +66 \cdot 0.92$	$20.6 = 4.1 + +51 \cdot 0.32$	$24.1 = 3.0 + +22 \cdot 0.96$	$47.1 = 39.6 + +6 \cdot 1.24$
8	$60.8 = 2.7 + +66 \cdot 0.88$	$12.4 = 3.2 + +51 \cdot 0.18$	$21.9 = 2.0 + +22 \cdot 0.90$	$42.2 = 35.7 + +6 \cdot 1.09$

TABLE 3

Timing results for Problem `Str389`. $n = 9275$.

Num. proc.	SSOR, low-r. ODB (M=66)	LRA SMW, low-r. ODB (M=66)	SSOR, orig. ODB	LRA SMW, orig. ODB (M=2214)
1	$230.2 = 18.9 + +72 \cdot 2.93$	$223.7 = 22.6 + +68 \cdot 2.96$	$141.4 = 16.1 + +39 \cdot 3.21$	$453.1 = 349.0 + +19 \cdot 5.48$
2	$181.9 = 10.3 + +70 \cdot 2.45$	$112.4 = 12.1 + +68 \cdot 1.47$	$105.4 = 8.2 + +39 \cdot 2.49$	$357.7 = 282.5 + +19 \cdot 3.95$
4	$167.7 = 6.2 + +77 \cdot 2.10$	$54.5 = 7.5 + +68 \cdot 0.69$	$95.4 = 4.2 + +43 \cdot 2.12$	$315.2 = 251.1 + +19 \cdot 3.38$
8	$156.1 = 4.2 + +77 \cdot 1.97$	$28.3 = 5.0 + +68 \cdot 0.34$	$85.9 = 2.2 + +43 \cdot 1.95$	$359.6 = 301.3 + +19 \cdot 3.07$

TABLE 4

Timing results for Problem `20284.mlp`, $n = 20284$.

Num	$n = 9275$ $M = 84$			$n = 9275$ $M = 1234$			$n = 20284$ $M = 66$			$n = 20284$ $M = 2214$		
proc	D	I+G	Tot	D	I+G	Tot	D	I+G	Tot	D	I+G	Tot
1	47.7	0.21	68.0	6.1	4.83	13.5	142.3	0.18	201.1	40.6	46.49	104.1
2	22.5	0.18	33.2	3.0	4.82	9.6	68.4	0.18	100.3	19.8	46.05	75.1
4	10.8	0.17	16.5	1.5	4.82	7.4	30.7	0.16	47.0	9.7	46.08	64.1
8	5.0	0.17	9.2	0.7	4.80	6.6	14.3	0.16	23.3	4.5	46.08	58.3

TABLE 5

Breakdown of iteration time. "D" is the total time spent solving $Dz = w$; "I+G" is the total time spent solving $(I + G)s = t$; "Tot" is the total time spent during iterations.