SIAM J. Sci. Comput.
Vol. 32, No. 3, pp. 1201–1216

WEIGHTED MATRIX ORDERING AND PARALLEL BANDED PRECONDITIONERS FOR ITERATIVE LINEAR SYSTEM SOLVERS[*]

MURAT MANGUOGLU[†], MEHMET KOYUTÜRK[‡], AHMED H. SAMEH[†], AND ANANTH GRAMA[†]

Abstract. The emergence of multicore architectures and highly scalable platforms motivates the development of novel algorithms and techniques that emphasize concurrency and are tolerant of deep memory hierarchies, as opposed to minimizing raw FLOP counts. While direct solvers are reliable, they are often slow and memory-intensive for large problems. Iterative solvers, on the other hand, are more efficient but, in the absence of robust preconditioners, lack reliability. While preconditioners based on incomplete factorizations (whenever they exist) are effective for many problems, their parallel scalability is generally limited. In this paper, we advocate the use of banded preconditioners instead and introduce a reordering strategy that enables their extraction. In contrast to traditional bandwidth reduction techniques, our reordering strategy takes into account the magnitude of the matrix entries, bringing the heaviest elements closer to the diagonal, thus enabling the use of banded preconditioners. When used with effective banded solvers—in our case, the Spike solver—we show that banded preconditioners (i) are more robust compared to the broad class of incomplete factorization-based preconditioners, (ii) deliver higher processor performance, resulting in faster time to solution, and (iii) scale to larger parallel configurations. We demonstrate these results experimentally on a large class of problems selected from diverse application domains.

Key words. spectral reordering, weighted reordering, banded preconditioner, incomplete factorization, Krylov subspace methods, parallel preconditioners

AMS subject classifications. 65F10, 65F50, 15A06

DOI. 10.1137/080713409

1. Introduction.

Solving sparse linear systems constitutes the dominant cost in diverse computational science and engineering applications. Direct linear system solvers are generally robust (i.e., they are guaranteed to find the solution of a nonsingular system in a precisely characterizable number of arithmetic operations); however, their application is limited by their relatively high memory and computational requirements. The performance of iterative methods, on the other hand, is more dependent on the properties of the linear system at hand. Furthermore, these iterative schemes often require effective preconditioning strategies if they are to obtain good approximations of the solution in a relatively short time.

In iterative sparse linear system solvers, preconditioners are used to improve the convergence properties. In general, a preconditioner (M) transforms the linear system ($Ax = f$) into another ($M^{-1}Ax = M^{-1}f$) in which the coefficient matrix ($M^{-1}A$) has a more favorable eigenvalue distribution, thus reducing the number of iterations required for convergence. This often comes at the expense of an increase in the number of FLOPs per iteration, in addition to the overhead of computing the preconditioner. Consequently, a desirable preconditioner is one that compensates for this overhead by a significant reduction in the required number of iterations.

[*]Received by the editors January 15, 2008; accepted for publication (in revised form) February 8, 2010; published electronically April 16, 2010. This work was partially supported by grants from NSF (NSF-CCF-0635169) and DARPA/AFRL (FA8750-06-1-0233) and by a gift from Intel.
http://www.siam.org/journals/sisc/32-3/71340.html

[†]Department of Computer Science, Purdue University, West Lafayette, IN 47907 (mmanguog@cs.purdue.edu, sameh@cs.purdue.edu, ayg@cs.purdue.edu).

[‡]Department of Electrical Engineering and Computer Science, Case Western Reserve University, Cleveland, OH 44106 (koyuturk@eecs.case.edu).

Multicore and petascale architectures provide computational resources for the solution of very large linear systems. Since direct solvers do not generally scale to large problems and machine configurations, efficient application of preconditioned iterative solvers is warranted. The primary advantage of preconditioners on such architectures is twofold: (i) their ability to enhance convergence of the outer iterations, and (ii) their scalability on large number of nodes/cores. Traditional preconditioners based on incomplete (triangular) factorizations often improve convergence properties whenever such approximate LU factors exist. However, their parallel scalability is often limited. On the other hand, banded linear system solvers such as Spike [29], used in each Krylov iteration utilizing banded preconditioners, are shown to provide excellent parallel scalability. Unlike sparse LU-factorization-based preconditioners, banded preconditioners do not require excessive indirect memory accesses—making them more amenable to compiler optimizations. In this context, the key questions are (i) how can one derive effective banded preconditioners, and (ii) how do these preconditioners perform in terms of enhancing convergence, processor performance, concurrency, and overall time-to-solution? These issues form the focus of our study.

In order to use banded preconditioners, we must first deal with the extraction of a "narrow-banded" matrix that serves as a good preconditioner. While traditional bandwidth reduction techniques such as reverse Cuthill–McKee (RCM) [14] and Sloan [36] reordering schemes are useful for general bandwidth reduction, they rarely yield effective preconditioners that are sufficiently narrow-banded. Furthermore, since symmetric reordering alone does not guarantee a nonsingular banded preconditioner, nonsymmetric reordering is needed to bring as many large nonzero entries to the diagonal as possible. To enhance effectiveness of banded preconditioners, we develop a bandwidth reduction technique aimed at encapsulating a significant part of the matrix into a "narrow band." Our approach is to reorder the matrix so as to move *heavy* (large magnitude) entries closer to the diagonal. This makes it possible to extract a narrow-banded preconditioner by dropping the entries that are outside the band. To solve this *weighted bandwidth reduction* problem, we use a *weighted spectral reordering* technique that provides an optimal solution to a continuous relaxation of the corresponding optimization problem. This technique is a generalization of spectral reordering, which has also been effectively used for reducing the bandwidth and envelope of sparse matrices [6]. To alleviate problems associated with symmetric reordering, we couple this method with nonsymmetric reordering techniques, such as the maximum traversal algorithm [18], to make the main diagonal free of zeros and place the largest entries on the main diagonal [19].

The resulting algorithm, which we refer to as WSO, can be summarized as follows: (i) use nonsymmetric reordering to move the largest entries to the diagonal; (ii) use weighted spectral reordering to move larger elements closer to the diagonal; (iii) extract a narrow central band to use as a preconditioner for Krylov subspace methods; and (iv) use our highly efficient Spike scheme for solving systems involving the banded preconditioner in each Krylov iteration. Our results show that this procedure yields a fast, highly parallelizable preconditioner with excellent convergence characteristics and parallel scalability. We demonstrate experimentally that WSO is more robust than incomplete factorization-based preconditioners for a broad class of problems.

The rest of this paper is organized as follows. In section 2, we discuss background and related work. In section 3, we present the weighted bandwidth reduction approach, describe our solution of the resulting optimization problem, and provide a brief overview of our chosen parallel banded solver infrastructure used in each outer

Krylov subspace iteration. In section 4, we present experimental results and comparisons with incomplete factorization-based preconditioners.

2. Background and related work. Preconditioning is a critical step in enhancing efficiency of iterative solvers for linear systems $Ax = f$. Often, this is done by transforming the system to

$$M^{-1}Ax = M^{-1}f. \tag{2.1}$$

Here, the left preconditioner M is designed so that the coefficient matrix $M^{-1}A$ has more desirable properties. For simplicity, throughout this paper, we use the term *preconditioning* to refer to left preconditioning. The discussion, however, also applies to right, or two-sided, preconditioners with minor modifications.

The inverse of the matrix M is chosen as an approximation to the inverse of A in the sense that the eigenvalues of $M^{-1}A$ are clustered around 1. Furthermore, M is chosen such that the action of M^{-1} on a vector is inexpensive, and is highly scalable on parallel architectures. A class of preconditioners that has received considerable attention is based on approximate LU-factorizations of the coefficient matrix. These methods compute an incomplete LU-factorization (ILU) by controlling fill-in as well as dropping small entries to obtain approximate LU factors. These ILU preconditioners have been applied successfully to various classes of linear systems [13, 43, 21]. High-quality software packages [37, 26, 25] that implement various forms of ILU preconditioning are also available. Recently, Benzi et al. investigated the role of reordering on the performance of ILU preconditioners [8, 7].

In this paper, we propose the use of banded preconditioners as an effective alternative to ILU preconditioners on parallel computing platforms with deep memory hierarchies.

2.1. Banded preconditioners. A preconditioner M is banded if

$$k_l < i - j \quad \text{or} \quad k_u < j - i \Rightarrow m_{ij} = 0, \tag{2.2}$$

where k_l and k_u are referred to as the lower and upper bandwidths, respectively. Half-bandwidth, k, is defined as the maximum of k_l and k_u, and the bandwidth of the matrix is equal to $k_l + k_u + 1$. Narrow-banded preconditioners have been used in a variety of applications, especially when the systems under considerations are diagonally dominant; see, e.g., [32, 41, 35]. However, in the absence of diagonal dominance, such banded preconditioners are not robust.

In general, reordering algorithms such as RCM [14] seek permutations of rows and columns that minimize matrix bandwidth. We propose an approach based on a generalization of spectral reordering aimed at minimizing the bandwidth that encapsulates a significant portion of the dominant nonzeros of the matrix, rather than all the nonzero entries.

3. Weighted bandwidth reduction. Since we are concerned with solving general nonsymmetric linear systems, we propose nonsymmetric as well as symmetric ordering techniques.

3.1. Nonsymmetric reordering. We first apply a nonsymmetric row permutation as follows:

$$QAx = Qf. \tag{3.1}$$

Here, Q is the row permutation matrix that maximizes either the number of nonzeros on the diagonal of A or the product of the absolute values of the diagonal entries [19]. The first algorithm is known as *maximum traversal search*, while the second scheme provides scaling factors so that the absolute values of the diagonal entries are equal to one, and all other elements are less than or equal to one. Such scaling is applied as follows:

$$(QD_2AD_1)(D_1^{-1}x) = (QD_2f). \tag{3.2}$$

Both algorithms are implemented in subroutine MC64 [17] of the HSL library [23].

3.2. Symmetric reordering. Following the above nonsymmetric reordering and optional scaling, we apply the symmetric permutation P as follows:

$$(PQD_2AD_1P^T)(PD_1^{-1}x) = (PQD_2f). \tag{3.3}$$

In this paper, we use weighted spectral reordering to obtain the symmetric permutation, P, that minimizes the bandwidth encapsulating a specified fraction of the total magnitude of nonzeros in the matrix. This permutation is determined from the symmetrized matrix $|A| + |A^T|$ as described in the next section.

3.2.1. Weighted spectral reordering. Traditional reordering algorithms, such as Cuthill–McKee [14] and spectral reordering [6], aim to minimize the bandwidth of a matrix. The half-bandwidth of a matrix A is defined as

$$BW(A) = \max_{i,j:A(i,j)\neq 0} |i-j|, \tag{3.4}$$

i.e., the maximum distance of a nonzero entry from the main diagonal. These methods pack the nonzeros of a sparse matrix into a band around the diagonal, but do not attempt to pack the heaviest elements into as narrow a central band as possible. To realize such a reordering strategy that enables the extraction of effective narrow-banded preconditioners, we introduce the weighted bandwidth reduction problem. Given a matrix A and a specified error bound ϵ, we define the weighted bandwidth reduction problem as one of finding a matrix M with minimum bandwidth, such that

$$\frac{\sum_{i,j} |A(i,j) - M(i,j)|}{\sum_{i,j} |A(i,j)|} \leq \epsilon \tag{3.5}$$

and

$$\begin{aligned} &M(i,j) = A(i,j) && \text{if } |i-j| \leq k, \\ &M(i,j) = 0. \end{aligned} \tag{3.6}$$

The idea behind this formulation is that if a significant part of the matrix is packed into a narrow band, then the rest of the nonzeros can be dropped to obtain an effective preconditioner. In order to find a heuristic solution to the weighted bandwidth reduction problem, we use a generalization of spectral reordering. Spectral reordering is a linear algebraic technique that is commonly used to obtain approximate solutions to various intractable graph optimization problems [22]. It has also been successfully applied to the bandwidth and envelope reduction problems for sparse matrices [6]. The core idea of spectral reordering is to compute a vector x that minimizes

$$\sigma_A(x) = \sum_{i,j:A(i,j)\neq 0} (x(i) - x(j))^2, \tag{3.7}$$

subject to $||x||_2 = 1$ and $x^T e = 0$. Here, it is assumed that the matrix A is real and symmetric. The vector x that minimizes $\sigma_A(x)$ under constraints provides a mapping of the rows (and columns) of matrix A to a one-dimensional Euclidean space, such that pairs of rows that correspond to nonzeros are located as close as possible to each other. Consequently, the ordering of the entries of the vector x provides an ordering of the matrix that significantly reduces the bandwidth.

Fiedler [20] first showed that the optimal solution to this problem is given by the eigenvector corresponding to the second smallest eigenvalue of the Laplacian matrix (L) of A,

$$\begin{aligned} &L(i,j) = -1 && \text{if } i \neq j \wedge A(i,j) \neq 0, \\ &L(i,i) = |\{j : A(i,j) \neq 0\}|. \end{aligned} \tag{3.8}$$

Note that the matrix L is positive semidefinite, and the smallest eigenvalue of this matrix is equal to zero. The eigenvector x that minimizes $\sigma_A(x) = x^T L x$, such that $||x||_2 = 1$ and $x^T e = 0$, is known as the Fiedler vector. The Fiedler vector is the eigenvector corresponding to the second smallest eigenvalue of the symmetric eigenvalue problem

$$Lx = \lambda x. \tag{3.9}$$

The Fiedler vector of a sparse matrix can be computed efficiently using iterative methods [28].

While spectral reordering is shown to be effective in bandwidth reduction, the classical approach described above ignores the magnitude of nonzeros in the matrix. Therefore, it is not directly applicable to the weighted bandwidth reduction problem. However, Fiedler's result can be directly generalized to the weighted case [11]. More precisely, the eigenvector x that corresponds to the second smallest eigenvalue of the weighted Laplacian $\bar{L}$ minimizes

$$\bar{\sigma}_A(x) = x^T \bar{L} x = \sum_{i,j} |A(i,j)|(x(i) - x(j))^2, \tag{3.10}$$

where $\bar{L}$ is defined as

$$\begin{aligned} &\bar{L}(i,j) = -|A(i,j)| && \text{if } i \neq j, \\ &\bar{L}(i,i) = \sum_j |A(i,j)|. \end{aligned} \tag{3.11}$$

We now show how weighted spectral reordering can be used to obtain a continuous approximation to the weighted bandwidth reduction problem. For this purpose, we first define the relative *bandweight* of a specified band of the matrix as follows:

$$w_k(A) = \frac{\sum_{i,j:|i-j|<k} |A(i,j)|}{\sum_{i,j} |A(i,j)|}. \tag{3.12}$$

In other words, the bandweight of a matrix A, with respect to an integer k, is equal to the fraction of the total magnitude of entries that are encapsulated in a band of half-width k.

For a given α, $0 \leq \alpha \leq 1$, we define α-bandwidth as the smallest half-bandwidth that encapsulates a fraction α of the total matrix weight, i.e.,

$$BW_\alpha(A) = \min_{k:w_k(A) \geq \alpha} k. \tag{3.13}$$

Observe that α-bandwidth is a generalization of half-bandwidth; i.e., when $\alpha = 1$, the α-bandwidth is equal to the half-bandwidth of the matrix.

Now, for a given vector $x \in \mathbb{R}^n$, define the injective permutation function $\pi(i) : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$, such that, for $1 \le i, j \le n$, $x(\pi(i)) \le x(\pi(j))$ iff $i \le j$. Here, n denotes the number of rows (columns) of the matrix A. Moreover, for a fixed k, define the function $\delta_k(i, j) : \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\} \to \{0, 1\}$, which quantizes the difference between $\pi(i)$ and $\pi(j)$ with respect to k, i.e.,

$$\delta_k(i,j) = \begin{cases} 0 & \text{if } |\pi(i) - \pi(j)| \le k, \\ 1 & \text{else.} \end{cases} \tag{3.14}$$

Let $\bar{A}$ be the matrix obtained by reordering the rows and columns of A according to π, i.e.,

$$\bar{A}(\pi(i), \pi(j)) = A(i,j) \text{ for } 1 \le i, j \le n. \tag{3.15}$$

Then $\delta_k(i, j) = 0$ indicates that $A(i, j)$ is inside a band of half-width k in the matrix A, while $\delta_k(i, j) = 1$ indicates that it is outside this band. Defining

$$\hat{\sigma}_k(A) = \sum_{i,j} |A(i,j)| \delta_k(i,j), \tag{3.16}$$

we obtain

$$\hat{\sigma}_k(A) = (1 - w_k(\bar{A})) \sum_{i,j} |A(i,j)|. \tag{3.17}$$

Therefore, for a fixed α, the α-bandwidth of the matrix $\bar{A}$ is equal to the smallest k that satisfies $\hat{\sigma}_A(k) / \sum_{i,j} |A(i,j)| \le 1 - \alpha$.

Observe that the problem of minimizing $\bar{\sigma}_x(A)$ is a continuous relaxation of the problem of minimizing $\hat{\sigma}_k(A)$ for a given k. Therefore, the Fiedler vector of the weighted Laplacian $\bar{L}$ provides a good basis for reordering A to minimize $\hat{\sigma}_k(A)$. Consequently, for a fixed ϵ, this vector provides a heuristic solution to the problem of finding a reordered matrix $\bar{A}$ with minimum $(1 - \epsilon)$-bandwidth. Once this matrix is obtained, we extract the banded preconditioner M as follows:

$$M = \{M(i,j) : M(i,j) = \bar{A}(i,j) \text{ if } |i - j| \le BW_{1-\epsilon}(\bar{A}),\ 0 \text{ else}\}. \tag{3.18}$$

Clearly, M satisfies (3.5) and is of minimal bandwidth.

Note that spectral reordering is defined specifically for symmetric matrices, and the resulting permutation is symmetric as well. Since the main focus of this study concerns general matrices, we apply spectral reordering to nonsymmetric matrices by computing the Laplacian matrix of $|A| + |A^T|$ instead of $|A|$. We note that this formulation results in a symmetric permutation for a nonsymmetric matrix, which may be considered overconstrained.

3.3. Summary of banded solvers. A number of banded solvers have been proposed and implemented in software packages such as LAPACK [4] for uniprocessors, and ScaLAPACK [10] and Spike [38, 12, 29, 9, 16, 39] for parallel architectures. Since Spike is shown to have excellent scalability [33, 34], we adopt it in this paper as the parallel banded solver of choice. The central idea of Spike is to partition the matrix so that each processor can work on its own part of the matrix, with the processors communicating only at the end of the process to solve a common reduced system. The size of the reduced system is determined by the bandwidth of the matrix and the number partitions.

Unlike classical sequential LU-factorization of the coefficient matrix A, for solving a linear system $Ax = f$, the Spike scheme employs the factorization

$$A = DS, \tag{3.19}$$

where D is the block diagonal of A. The factor S, given by $D^{-1}A$ (assuming D is nonsingular), called the spike matrix, consists of the block diagonal identity matrix modified by "spikes" to the right and left of each partition. The case of nonsingular diagonal blocks of A can be overcome as outlined later in section 4.1. The process of solving $Ax = f$ reduces to a sequence of steps that are ideally suited for parallel execution and performance. Furthermore, each step can be accomplished using one of several available methods, depending on the specific parallel architecture and the linear system at hand. This gives rise to a family of optimized variants of the basic Spike scheme [38, 12, 29, 9, 16, 39].

4. Numerical experiments.

We perform extensive experiments to establish the runtime and convergence properties of banded preconditioners for a wide class of problems. We also compare our preconditioner with other popular "black-box" preconditioners. The first set of problems consists of systems whose number of unknowns varies from 7,548 to 116,835. The second set contains systems from the University of Florida Sparse Matrix Collection with more than 500,000 unknowns. In addition, we provide parallel scalability results of our solver on the larger test cases.

4.1. Robustness of WSO on a uniprocessor.

This set of problems consists of symmetric and nonsymmetric general sparse matrices from the University of Florida [15] and a linear system obtained from Dr. Olaf Schenk (mips80S-1). In this set of experiments, we use only the sequential version of our algorithm on a 2.66GHz Intel Xeon processor. The test problems are presented in Table 1. For each matrix, we generate the corresponding right-hand side using a solution vector of all ones to ensure that $f \in span(A)$. The number k represents the bandwidth of the preconditioner detected by our method. The condition number is estimated via MATLAB's *condest* function for small problems (<500,000 unknowns) and via MUMPS [3, 2, 1] for larger problems. In all of the numerical experiments, we use the Bi-CGSTAB [42] iterative scheme with a narrow-banded preconditioner, and terminate the iterations when the relative residual satisfies

$$||r_k||_\infty / ||r_0||_\infty \leq 10^{-5}. \tag{4.1}$$

Implementation of ILU-based preconditioners. For establishing a competitive baseline, we use ILUT preconditioners as well as the enhanced version, ILUTI, proposed by Benzi et al. [8, 7]. To implement this enhanced ILUT scheme, we use the nonsymmetric reordering algorithm available in HSL's MC64 to maximize the absolute value of the product of diagonal entries of the coefficient matrix. This is followed by scaling, and the symmetric RCM reordering, and finally the ILUT factorization. We refer to this enhanced ILUT factorization as ILUTI. We use ILUT from the SPARSEKIT software package [37] with dual numerical dropping and fill-in threshold strategy. In the following experiments we allow fill-in of no more than 20% of the number of nonzeros per row and a relative drop tolerance of 10^{-1}.

Table 1
Properties of test matrices.

Number	Name	Condest	k	Dimension (N)	Nonzeros (nnz)	nnz/N	Type
1	FINAN512	9.8×10^1	50	74,752	596,992	8.05	Financial optimization
2	FEM_3D_THERMAL1	1.7×10^3	50	17,880	430,740	24.09	3D thermal FEM
3	RAJAT31	4.4×10^3	30	4,690,002	20,316,253	4.33	Circuit simulation
4	H2O	4.9×10^3	50	67,024	2,216,736	33.07	Quantum chemistry
5	APPU	1.0×10^4	50	14,000	1,853,104	132.36	NASA benchmark
6	BUNDLE1	1.3×10^4	50	10,581	770,811	72.84	3D computer vision
7	MIPS80S-1	3.3×10^4	30	986,552	7,630,280	7.73	Nonlinear optimization
8	RAJAT30	8.7×10^4	30	643,994	6,175,244	9.59	Circuit simulation
9	DW8192	1.5×10^7	182	8,192	41,746	5.1	Dielectric waveguide
10	DC1	1.1×10^{10}	50	116,835	766,396	6.5	Circuit simulation
11	MSC23052	1.4×10^{12}	50	23,052	1, 154, 814	50.1	Structural mechanics
12	FP	8.2×10^{12}	2	7,548	834,222	110.52	Electromagnetics
13	PRE2	3.3×10^{13}	30	659,033	5,959,282	9.04	Harmonic balance method
14	KKT_POWER	4.2×10^{13}	30	2,063,494	14,612,663	7.08	Nonlinear optimization (KKT)
15	RAEFSKY4	1.5×10^{14}	50	19,779	1,328,611	67.12	Structural mechanics
16	ASIC_680k	9.4×10^{19}	3	682,862	3,871,773	5.66	Circuit simulation
17	2D_54019_HIGHK	8.1×10^{32}	2	54,019	996,414	18.45	Device simulation

Implementation of weighted bandwidth reduction-based preconditioners. To implement our WSO scheme, we first use spectral reordering of $|A| + |A^T|$ via HSL's MC73 [24] using $TOL = 1.0 \times 10^{-1}$ (tolerance for the Lanczos algorithm on the coarsest grid), $TOL1 = 5.0 \times 10^{-1}$ (tolerance for the Rayleigh quotient iterations at each refinement step), and $RTOL = 5.0 \times 10^{-1}$(SYMMLQ tolerance for solving linear systems in the Rayleigh quotient iterations). We then apply MC64 to maximize the absolute value of the product of diagonal entries of the coefficient matrix. All the other parameters of MC73 are kept at the default setting.

After obtaining the reordered system, we determine the $(1-\epsilon)$-bandwidth. Note that the computation of the $(1-\epsilon)$-bandwidth requires $O(nnz) + O(n)$ time and $O(n)$ storage. To achieve this, we first create a work array, $w(1:n)$, of dimension n. Then, for each nonzero, a_{ij}, we update $w(abs(i-j)) = w(abs(i-j)) + abs(a_{ij})$. Finally, we compute a cumulative sum in w starting from index 0 until the $(1-\epsilon)$-bandwidth is reached. The time for this process is included in the total time reported in our experiments. In this set of experiments, we choose $\epsilon = 10^{-4}$. In general, the corresponding $(1-\epsilon)$-bandwidth (or k) can be as large as n. To prevent this, we place an upper limit $k \leq 50$ if the matrix dimension is greater than 10,000 and an upper limit $k \leq 30$ if the matrix dimension is larger than 500,000.

After determining k, the WSO scheme proceeds with the extraction of the banded preconditioner with the bandwidth $2k+1$. We then use the factorization of the banded system to precondition the Bi-CGSTAB iterations.

While the nonsymmetric ordering via HSL's MC64 minimizes the occurrence of small pivots when factoring the banded preconditioner, the banded preconditioner M resulting from the WSO process may still be singular due to the dropping of a large number of elements outside the band. This can be detected during the banded solve, and the preconditioner is updated as follows:

$$M = M + \alpha I, \tag{4.2}$$

where α is selected as $\alpha = 10^{-5}||M||_\infty$. Our timing results include this extra factorization time, should a singular M be encountered after the use of HSL's MC64.

In addition to WSO, we provide timings obtained if one uses RCM reordering on the system and uses a banded preconditioner, which we will refer to as RCMB.

Solution of linear systems. For both ILUTI and WSO, we solve (3.3) by considering the following reordered and scaled system:

$$\tilde{A}\tilde{x} = \tilde{f}. \tag{4.3}$$

Here, $\tilde{A} = PQD_2AD_1P^T$, $\tilde{x} = PD_1^{-1}x$, and $\tilde{f} = PQD_2f$. The corresponding scaled residual is given by $\tilde{r} = \tilde{f} - \tilde{A}\tilde{x}$, where $\tilde{x}$ is the computed solution. One can recover the residual of the original system, $r = f - Ax$, as follows:

$$\tilde{r} = PQD_2f - PQD_2AD_1P^TPD_1^{-1}x \tag{4.4}$$
$$\Rightarrow \tilde{r} = PQD_2f - PQD_2Ax \tag{4.5}$$
$$\Rightarrow \tilde{r} = PQD_2(f - Ax) \tag{4.6}$$
$$\Rightarrow \tilde{r} = PQD_2r \tag{4.7}$$
$$\Rightarrow r = D_2^{-1}Q^TP^T\tilde{r}. \tag{4.8}$$

At each iteration, we check the unscaled relative residuals using (4.8). Note that a permutation of the inverted scaling matrix is computed once and used for recovering the unscaled residual at each iteration. This process is included in the total time, but the cost is minimal compared to sparse matrix-vector multiplication and triangular solves.

We use PARDISO with the built in METIS [27] reordering and enable the nonsymmetric ordering option for indefinite systems. For ILUPACK [26], we use the PQ reordering option and a relative drop tolerance of 10^{-1} with a bound on the norm of the inverse triangular factors 50 (recommended in the user documentation for general problems). In addition, we enable options in ILUPACK to use nonsymmetric permutation and scaling (similar to MC64). We use the same stopping criterion as in the previous section (1.0×10^{-5}).

Table 2
Number of iterations for ILUTI, WSO, ILUPACK, PARDISO, and RCMB methods.

		Banded		*LU based*	
Number	Condest	WSO	RCMB	ILUTI(*,10^{-1})	ILUPACK
1	9.8×10^{1}	8	8	6	7
2	1.7×10^{3}	37	22	26	12
3	4.4×10^{3}	286	F1	501	F2
4	4.9×10^{3}	147	144	40	93
5	1.0×10^{4}	23	28	15	44
6	1.3×10^{4}	19	18	12	13
7	3.3×10^{4}	224	F1	247	372
8	8.7×10^{4}	613	F1	81	F1
9	1.5×10^{7}	23	1	F2	F2
10	1.1×10^{10}	24	25	164	15
11	1.4×10^{12}	F2	F2	F2	F2
12	8.2×10^{12}	1	1	F2	26
13	3.3×10^{13}	2	F1	F2	F2
14	4.2×10^{13}	F2	F1	F2	F2
15	1.5×10^{14}	4	4	48	F2
16	9.4×10^{19}	7	F1	10	F2
17	8.1×10^{32}	1	1	1	73

The number of iterations for iterative methods is presented in Table 2. A failure during the factorization of the preconditioner is denoted by F1, while F2 denotes stagnation of the iterative scheme. In Table 3, we demonstrate the total solution time of each algorithm, including the direct solver PARDISO.

In typical science and engineering applications, one selects a solver and preconditioner for the problem at hand, as opposed to trying a large number of combinations and then selecting the best one to solve the problem (again). This is because the performance characteristics of preconditioned iterative solvers are sensitive to the nature of the linear system; consequently, performance on one system may not reveal insights into its performance on other linear systems. Choosing ILU-based preconditioners, e.g., ILUTI or ILUPACK, for the system collection in Table 1 results in failure in 29.4% and 47.1% of the cases, respectively. Choosing the WSO-based banded preconditioner results in failure in only 11.8% of the cases. This shows the significant improvement in robustness of the WSO scheme, as compared to the ILU-based preconditioners. Not surprisingly, the direct solver, PARDISO, is the most robust in the set. However, in terms of total solve time, it is faster than WSO only in two cases. We also note from the table that ILU-factorization-based preconditioners are faster than

TABLE 3
Total solve time for ILUTI, WSO, ILUPACK, PARDISO, and RCMB methods.

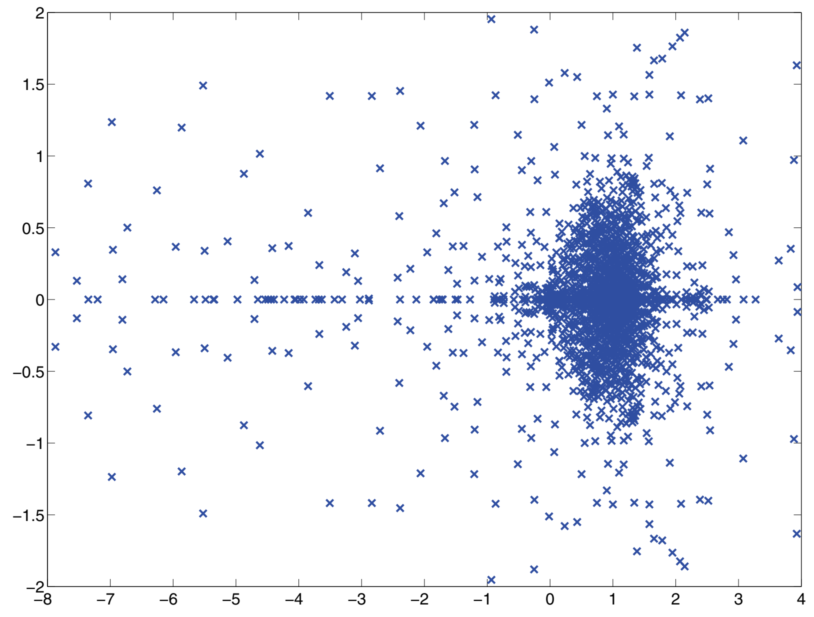
		Banded		*LU based*		
Number	Condest	WSO	RCMB	ILUTI(*,10^{-1})	PARDISO	ILUPACK
1	9.8×10^1	0.87	0.29	0.14	1.28	0.5
2	1.7×10^3	0.41	0.19	0.19	1.48	0.23
3	4.4×10^3	457.2	F1	193.5	91.6	F2
4	4.9×10^3	5.63	4.89	0.84	450.73	2.86
5	1.0×10^4	0.82	0.51	0.88	270.17	44.6
6	1.3×10^4	0.7	0.14	0.27	0.88	0.27
7	3.3×10^4	77.3	F1	26.2	1,358.7	330
8	8.7×10^4	205.4	F1	459.1	7.6	F1
9	1.5×10^7	0.33	0.07	F2	0.78	F2
10	1.1×10^{10}	1.95	8.99	15.8	3.14	2.07
11	1.4×10^{12}	F2	F2	F2	1.07	F2
12	8.2×10^{12}	0.42	0.04	F2	1.74	0.46
13	3.3×10^{13}	7.0	F1	F2	45.3	F2
14	4.2×10^{13}	F2	F1	F2	449.1	F2
15	1.5×10^{14}	0.27	0.09	0.59	1.28	F2
16	9.4×10^{19}	2.0	F1	239.0	27.2	F2
17	8.1×10^{32}	0.21	0.06	0.11	0.95	1.44

banded preconditioners (WSO and RCMB) for problems that are well conditioned (condest $\leq 3.3 \times 10^4$). However, for moderate to ill-conditioned problems (condest $> 3.3 \times 10^4$), WSO performs better.

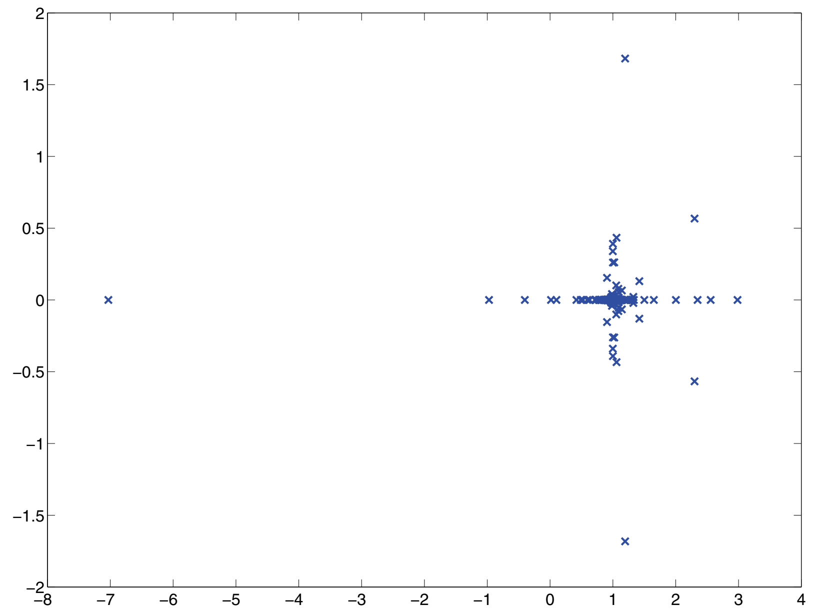
We now examine causes for failure of incomplete factorization preconditioners and WSO more closely. For this study, we primarily focus on the ILUTI preconditioner, since its performance is superior to that of ILUPACK for our test problems. Of the five failures for ILUTI, four are attributed to failure of convergence, as opposed to problems in factorization. To understand these problems with convergence, we consider a specific problem instance, dw8192 (matrix #9). We select this since it is the smallest matrix for which we observe failure for both ILU-based solvers. This allows us to examine the spectrum of the preconditioned matrix. In Figures 1(a) and (b), we show the spectrum of the preconditioned matrix $M^{-1}A$ for ILUTI and WSO. The figure clearly explains the superior robustness of WSO—the eigenvalues have a much better clustering around 1 than with the ILU-preconditioned matrix.

The remaining case of ILUTI failure, rajat30 (matrix #8), is due to a zero pivot encountered during the ILU-factorization. Even in cases where ILUTI and WSO require a similar number of iterations, WSO often outperforms ILUTI preconditioned solvers because the ILU factors are denser, and hence both triangular factorization and solves are more costly. This happens for matrix appu (matrix #5). In yet another case, asic_680k (matrix #16), which comes from circuit simulation, the number of iterations is comparable for the ILUTI and WSO solvers, the fill during incomplete factorization is low, but the reordering time in ILUTI is very high. For this reason, WSO preconditioned solve is over two orders of magnitude faster than the corresponding ILUTI preconditioned solve.

Our examination of performance characteristics of ill-conditioned matrices in our test set therefore reveals the following insights: (i) WSO preconditioned matrices have better clustering of the spectra than ILUTI preconditioned matrices for the



(a) Spectrum of the ILUTI preconditioned matrix.



(b) Spectrum of the WSO preconditioned matrix.

FIG. 1. *Spectrum of $M^{-1}A$, where M is the ILUTI preconditioner* (a) *or the WSO preconditioner* (b) *for dw*8192 *(close-up view around* 1*). The spectra clearly demonstrate superior preconditioning by WSO.*

small matrix (for which we could compute the spectra); (ii) in some cases, the failure of ILUTI was due to problems in factorization; and (iii) even in cases where iteration counts of ILUTI and WSO preconditioned solvers were similar, ILUTI solve time is higher because of factorization or reordering time.

We note that all of the iterative solvers (WSO and all variants of ILU) fail for kkt_power (matrix #14). This matrix corresponds to a saddle point problem. For saddle point problems, other specifically designed algorithms are more suitable. We refer the reader to [31, 30, 5] for a more detailed discussion of saddle point problems and nested iterative schemes.

4.2. Parallel scalability of WSO. In this section, we demonstrate the parallel scalability of our solver on a cluster of Intel multicore (2.66GHz) processors with an Infiniband interconnect. We select the largest three problems from Table 1 in which WSO succeeds: rajat31, asic_680k, and mips80S-1 (matrices #3, #16, and #7). We compare the scalability of our solver to two other existing distributed memory direct sparse solvers, namely, SuperLU and MUMPS. We use METIS reordering for MUMPS and ParMETIS [40] for SuperLU. In order to solve the banded systems we use the truncated variation of the Spike algorithm [34]. For the largest problem, namely, rajat31 (matrix #3), we use a maximum of 64 cores. For the other two smaller problems we use a maximum of 32 cores. In Figures 2, 3, and 4, we show the total time of each solver. In the total time, we include any preprocessing, reordering, factorization, and triangular solve times. For rajat31, we note that the scalability of MUMPS and SuperLU is limited, while WSO scales well and solves the linear system faster using 64 cores. For asic680_k, WSO is the fastest compared to MUMPS and SuperLU. Similarly for mips80s-1, WSO is faster than MUMPS and scales well, while SuperLU fails due to excessive fill-in. We note that the majority of the total time is spent in METIS and ParMETIS reorderings when using MUMPS and SuperLU for asic680_k. This portion of the total time can be amortized if one needs to solve several linear systems with the same coefficient matrix but with many different right-hand sides. Similarly, a significant portion of the total time spent in WSO reordering can be amortized.

5. Conclusion.

In this paper, we present banded preconditioners as and attractive alternative to ILU-type preconditioners. We show the robustness and parallel scalability of banded preconditioners on parallel computing platforms. In addition, we identify the central role of reordering schemes in extracting prominent narrow central bands as effective preconditioners. We propose the use of nonsymmetric reordering to enhance the diagonal of the coefficient matrix, and the use of an approximation of the symmetric weighted spectral reordering to encapsulate most of the Frobenius norm of the matrix in as narrow a central band as possible. We demonstrate that the resulting banded preconditioner, used in conjunction with a Krylov subspace iterative scheme such as Bi-CGSTAB, is much more robust than almost all of the optimized versions of ILU preconditioners. Not only is this type of preconditioner more robust, often resulting in the least sequential time-to-solution, but it is also scalable on a variety of parallel architectures. Such parallel scalability is directly inherited from the use of the underlying Spike solver for the linear systems involving the banded preconditioner in each Krylov iteration.

Acknowledgments. We would like to thank Michaele Benzi, Jennifer Scott, and Eric Polizzi for many useful discussions and for the software used in our experiments.

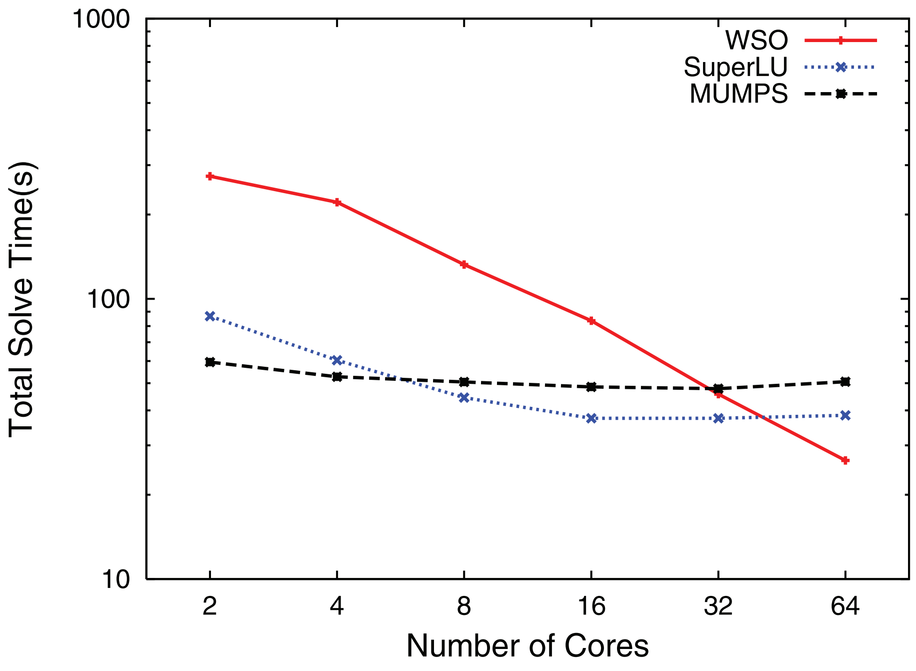


FIG. 2. *Total solve time for WSO, SuperLU, and MUMPS for solving RAJAT*31.

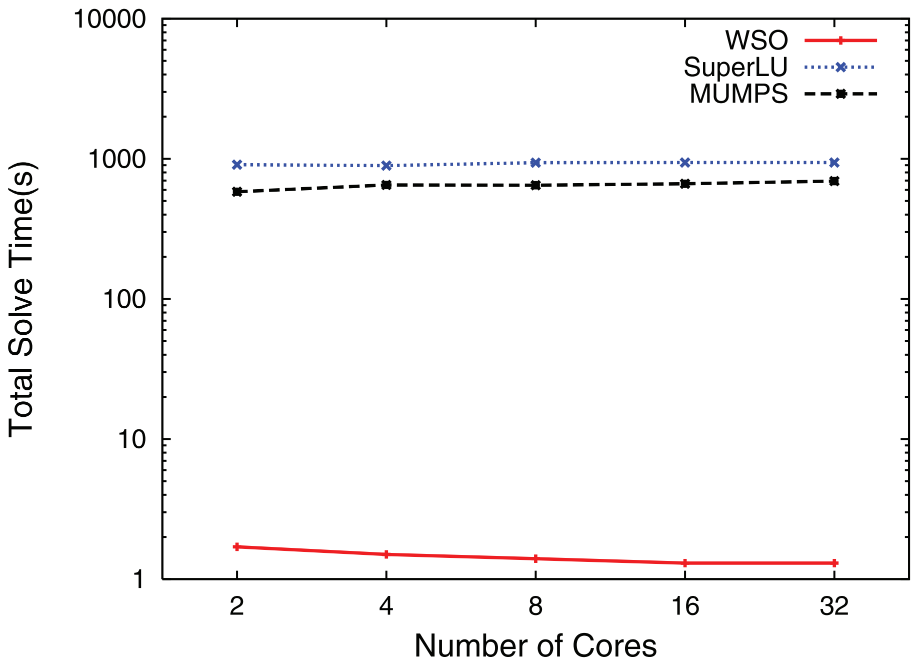


FIG. 3. *Total solve time for WSO, SuperLU, and MUMPS for solving ASIC*680*k*.

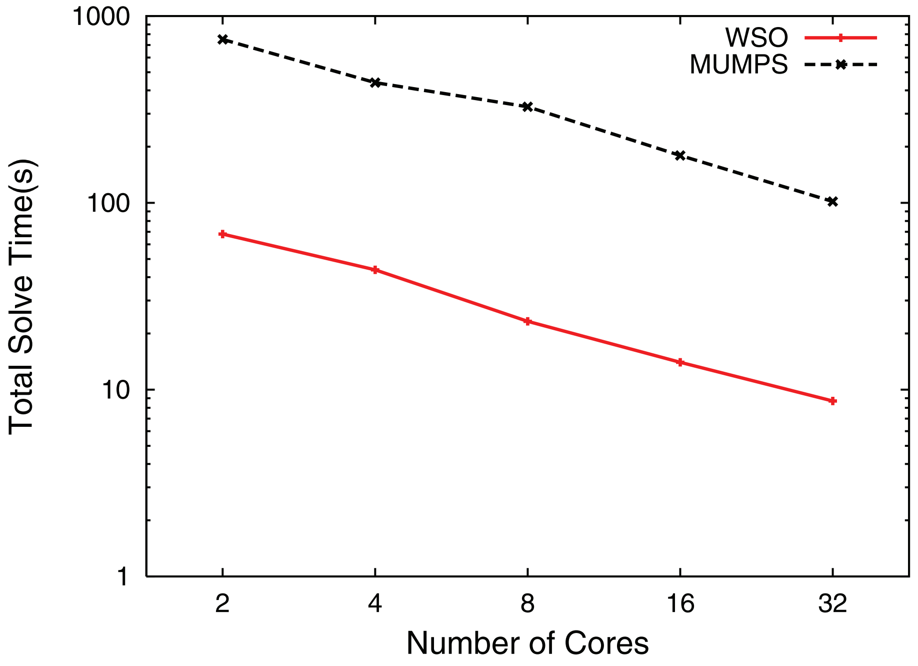


Fig. 4. *Total solve time for WSO and MUMPS for solving mips*80*S*-1.

REFERENCES

[1] P. R. Amestoy and I. S. Duff, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Comput. Methods Appl. Mech. Engrg., 184 (2000), pp. 501–520.

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

[3] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet, *Hybrid scheduling for the parallel solution of linear systems*, Parallel Comput., 32 (2006), pp. 136–156.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK: A Portable Linear Algebra Library for High-Performance Computers*, Tech. report UT-CS-90-105, University of Tennessee, Knoxville, TN, 1990.

[5] A. Baggag and A. Sameh, *A nested iterative scheme for indefinite linear systems in particulate flows*, Comput. Methods Appl. Mech. Engrg., 193 (2004), pp. 1923–1957.

[6] S. T. Barnard, A. Pothen, and H. Simon, *A spectral algorithm for envelope reduction of sparse matrices*, Numer. Linear Algebra Appl., 2 (1995), pp. 317–334.

[7] M. Benzi, J. C. Haws, and M. Tůma, *Preconditioning highly indefinite and nonsymmetric matrices*, SIAM J. Sci. Comput., 22 (2000), pp. 1333–1353.

[8] M. Benzi, D. B. Szyld, and A. van Duin, *Orderings for incomplete factorization preconditioning of nonsymmetric problems*, SIAM J. Sci. Comput., 20 (1999), pp. 1652–1670.

[9] M. W. Berry and A. Sameh, *Multiprocessor schemes for solving block tridiagonal linear systems*, Internat. J. Supercomput. Appl., 1 (1988), pp. 37–57.

[10] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK: A linear algebra library for message-passing computers*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997), CD-ROM, SIAM, Philadelphia, PA, 1997.

[11] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, *Spectral k-way ratio-cut partitioning and clustering*, IEEE Trans. Computer-Aided Design Integrated Circuits Systems, 13 (1994), pp. 1088–1096.

[12] S. C. Chen, D. J. Kuck, and A. H. Sameh, *Practical parallel band triangular system solvers*, ACM Trans. Math. Software, 4 (1978), pp. 270–277.

[13] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.
[14] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th National Conference, ACM Press, New York, 1969, pp. 157–172.
[15] T. A. Davis, *University of Florida sparse matrix collection*, NA Digest, 97 (23) (1997).
[16] J. J. Dongarra and A. H. Sameh, *On some parallel banded system solvers*, Parallel Comput., 1 (1984), pp. 223–235.
[17] I. S. Duff and J. Koster, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. Matrix Anal. Appl., 22 (2001), pp. 973–996.
[18] I. S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.
[19] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. Matrix Anal. Appl., 20 (1999), pp. 889–901.
[20] M. Fiedler, *Algebraic connectivity of graphs*, Czechoslovak Math. J., 23 (1973), pp. 298–305.
[21] J. R. Gilbert and S. Toledo, *An assessment of incomplete-LU preconditioners for nonsymmetric linear systems*, Informatica (Slovenia), 24 (2000), pp. 409–425.
[22] B. Hendrickson and R. Leland, *An improved spectral graph partitioning algorithm for mapping parallel computations*, SIAM J. Sci. Comput., 16 (1995), pp. 452–469.
[23] *The HSL Mathematical Software Library*; see http://www.hsl.rl.ac.uk/index.html.
[24] Y. F. Hu and J. A. Scott, *HSL_MC73: A Fast Multilevel Fiedler and Profile Reduction Code*, Tech. report RAL-TR-2003-036, Rutherford Appleton Laboratory, Oxfordshire, UK, 2003.
[25] *ILU++*, http://www.iluplusplus.de.
[26] *ILUPACK*, http://www.math.tu-berlin.de/ilupack/.
[27] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392.
[28] N. P. Kruyt, *A conjugate gradient method for the spectral partitioning of graphs*, Parallel Comput., 22 (1997), pp. 1493–1502.
[29] D. H. Lawrie and A. H. Sameh, *The computation and communication complexity of a parallel banded system solver*, ACM Trans. Math. Software, 10 (1984), pp. 185–195.
[30] M. Manguoglu, F. Saied, A. H. Sameh, T. E. Tezduyar, and S. Sathe, *Preconditioning techniques for nonsymmetric linear systems in computation of incompressible flows*, J. Appl. Mech., 76 (2009), article 021204.
[31] M. Manguoglu, A. H. Sameh, T. E. Tezduyar, and S. Sathe, *A nested iterative scheme for computation of incompressible flows in long domains*, Comput. Mech., 43 (2008), pp. 73–80.
[32] M. K. Ng, *Fast iterative methods for symmetric sinc-Galerkin systems*, IMA J. Numer. Anal., 19 (1999), pp. 357–373.
[33] E. Polizzi and A. H. Sameh, *A parallel hybrid banded system solver: The Spike algorithm*, Parallel Comput., 32 (2006), pp. 177–194.
[34] E. Polizzi and A. H. Sameh, *Spike: A parallel environment for solving banded linear systems*, Computers & Fluids, 36 (2007), pp. 113–120.
[35] M. S. Reeves, D. C. Chatfield, and D. G. Truhlar, *Preconditioned complex generalized minimal residual algorithm for dense algebraic variational equations in quantum reactive scattering*, J. Chem. Phys., 99 (1993), pp. 2739–2751.
[36] J. K. Reid and J. A. Scott, *Implementing Hager's exchange methods for matrix profile reduction*, ACM Trans. Math. Software, 28 (2002), pp. 377–391.
[37] Y. Saad, *SPARSKIT: A Basic Tool Kit for Sparse Matrix Computations*, Tech. report 90-20, NASA Ames Research Center, Moffett Field, CA, 1990.
[38] A. H. Sameh and D. J. Kuck, *On stable parallel linear system solvers*, J. ACM, 25 (1978), pp. 81–91.
[39] A. H. Sameh and V. Sarin, *Hybrid parallel linear system solvers*, Int. J. Comput. Fluid Dyn., 12 (1999), pp. 213–223.
[40] K. Schloegel, G. Karypis, and V. Kumar, *Parallel static and dynamic multi-constraint graph partitioning*, Concurrency and Computation: Practice and Experience, 14 (2002), pp. 219–240.
[41] Z. Tong and A. Sameh, *On optimal banded preconditioners for the five-point Laplacian*, SIAM J. Matrix Anal. Appl., 21 (1999), pp. 477–480.
[42] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–644.
[43] J. Zhang, *On preconditioning Schur complement and Schur complement preconditioning*, Elect. Trans. Numer. Anal., 10 (2000), pp. 115–130.