SIAM J. SCI. COMPUT.
Vol. 37, No. 2, pp. C217–C246
© 2015 Society for Industrial and Applied Mathematics

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

COMMUNICATION AVOIDING ILU0 PRECONDITIONER*

LAURA GRIGORI† AND SOPHIE MOUFAWAD†‡

Abstract. In this paper we present a communication avoiding ILU0 preconditioner for solving large linear systems of equations by using iterative Krylov subspace methods. Recent research has focused on communication avoiding Krylov subspace methods based on so-called s-step methods. However, there are not many communication avoiding preconditioners yet, and this represents a serious limitation of these methods. Our preconditioner allows us to perform s iterations of the iterative method with no communication, through ghosting some of the input data and performing redundant computation. To avoid communication, an alternating reordering algorithm is introduced for structured and well partitioned unstructured matrices, which requires the input matrix to be ordered by using a graph partitioning technique such as k-way or nested dissection. We show that the reordering does not affect the convergence rate of the ILU0 preconditioned system as compared to k-way or nested dissection ordering, while it reduces data movement and is expected to reduce the time needed to solve a linear system. In addition to communication avoiding Krylov subspace methods, our preconditioner can be used with classical methods such as GMRES to reduce communication.

Key words. minimizing communication, linear algebra, iterative methods, incomplete LU preconditioner

AMS subject classifications. 65F08, 65Y05, 68W10

DOI. 10.1137/130930376

1. Introduction.

Many scientific problems require the solution of systems of linear equations of the form $Ax = b$, where the input matrix A is very large and sparse. We focus in this paper on solving such systems using Krylov subspace methods, such as GMRES [31] and CG [23]. These methods iterate from a starting vector y until convergence or stagnation, by using projections onto the Krylov subspace $K(A, y) = span[y, Ay, A^2y, \ldots]$. In the parallel case, the input matrix is distributed over processors, and each iteration involves multiplying the input matrix with a vector, followed by an orthogonalization process. Both these operations require communication among processors. Since A is usually very sparse, the communication dominates the overall cost of the iterative methods when the number of processors is increased to a large number. While the matrix-vector product can be performed by using point-to-point communication routines between subsets of processors, the orthogonalization step requires the usage of collective communication routines, and these routines are known to scale poorly. More generally, on current machines the cost of communication, the movement of data, is much higher than the cost of arithmetic operations, and this gap is expected to continue to increase exponentially. As a result, communication is often the bottleneck in numerical algorithms.

In a quest to address the communication problem, recent research has focused on reformulating linear algebra operations such that the movement of data is significantly reduced or even minimized, as in the case of dense matrix factorizations [12, 20, 3]. Such algorithms are referred to as communication avoiding. The communication avoiding Krylov subspace methods [29, 24, 6] are based on s-step methods

*Submitted to the journal's Software and High-Performance Computing section July 23, 2013; accepted for publication (in revised form) December 1, 2014; published electronically April 14, 2015.
http://www.siam.org/journals/sisc/37-2/93037.html
†INRIA Paris-Rocquencourt, Alpines, and UPMC - Univ Paris 6, CNRS UMR 7598, Laboratoire Jacques-Louis Lions, Paris, France (laura.grigori@inria.fr, smm33@aub.edu.lb).
‡Corresponding author.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

[9, 33], which reformulate the iterative method such that the dependencies between certain iterations are eliminated. In short, the reformulation allows us, by unrolling s iterations of the Krylov subspace method and ghosting some of the data, to compute s vectors of the basis without communication, followed by an orthogonalization step. The orthogonalization step is performed by using TSQR, a communication optimal QR factorization algorithm [12]. Another recent approach introduced in [18] focuses on hiding the reduction latency in pipelined GMRES by overlapping the global communication phase with other useful computations and local communications. For $s = 1$, where s-step methods are equivalent to classical methods, there are many available preconditioners. One of them, block Jacobi, is a naturally parallelizable and communication avoiding preconditioner. However, except for a discussion in [24], there are no available preconditioners that avoid communication and can be used with s-step methods for $s > 1$. This is a serious limitation of these methods, since for difficult problems, Krylov subspace methods without preconditioner can be very slow or even might not converge.

Our goal is to design communication avoiding preconditioners that should be efficient in accelerating the iterative method and should also minimize communication. In other words, given a preconditioner M, the preconditioned system with its communication avoiding version $M_{ca}^{-1}Ax = M_{ca}^{-1}b$ should have the same order of convergence as the original preconditioned system $M^{-1}Ax = M^{-1}b$, and also reduce communication. This is a challenging problem, since applying a preconditioner on its own may, and in general will, require extra communication. Since the construction of M represents typically a large part of the overall runtime of the linear solver, we focus on both minimizing communication during the construction of M and during its application to a vector at each iteration of the linear solver. The incomplete LU factorization (ILU) is a widely used black-box preconditioner, which can be used on its own or as a building block of other preconditioners as domain decomposition methods. The ILU preconditioner is written as $M = LU$, where L and U are the incomplete factors of the input matrix A. This preconditioner is obtained by computing a direct LU factorization of the matrix A, and by dropping some of the elements during the decomposition, based on either their numerical value or their relation with respect to the graph of the input matrix A [28, 30]. For a historical overview on preconditioning techniques and incomplete factorizations, refer to [4, 32].

In this paper we introduce CA-ILU0, a communication avoiding ILU0 preconditioner for left preconditioned systems. With a few modifications discussed in section 5, the CA-ILU0 preconditioner can be applied to right and split preconditioned systems. We first start by introducing definitions, concepts, and algorithms used in this paper in section 2. Then we adapt the matrix powers kernel to the ILU preconditioned system to obtain the ILU matrix powers kernel in section 3. Each vector of this kernel is obtained by computing $((LU)^{-1}Ax)$; that is, in addition to the matrix-vector multiplication Ax, it uses a forward and a backward substitution. The ILU matrix power kernel, which is designed for any given LU decomposition, does not by itself allow one to avoid communication. That is, if we want to compute s vectors of this kernel with no communication through ghosting some of the data, there are cases when one processor performs an important part of the entire computation. We restrain then our attention to the ILU0 factorization, which has the property that the L and U factors have the same sparsity pattern as the lower triangular part of A and the upper triangular part of A, respectively. To obtain a communication avoiding ILU0 preconditioner, we introduce in section 4 a reordering, alternating min-max layers Amml(s), of the input matrix A, which is reflected in the L and U matrices.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

The AMML(1) reordering allows one to avoid communication when computing $s = 1$ matrix vector multiplication $(LU)^{-1}Ax$. In other words, the matrix vector multiplication Ax, and the backward and forward substitution, are parallelized with only one communication before starting the whole computation. Thus, the CA-ILU0 preconditioner can be used with classical preconditioned Krylov methods like GMRES to reduce communication.

In general, the AMML(s) reordering allows the avoidance of communication for s-steps of the matrix vector multiplication $((LU)^{-1}Ax)$. In other words, s backward and forward solves corresponding to a submatrix of A can be performed when $s \geq 1$, without needing any data from other submatrices. Thus with our reordering it is sufficient to communicate once at the beginning of the first multiplication. This is possible since the CA-ILU0 $(L)^{-1}$ and $(U)^{-1}$ are sparse, unlike those of ILU0 (see Figure 4). Thus, the CA-ILU0 preconditioner can also be used with s-step Krylov methods like s-step GMRES and CA-GMRES to reduce communication. In section 6.2, we discuss the reduction in communication introduced by our method for $s \geq 1$.

In this paper we portray our CA-ILU0 preconditioner (section 5) and its performance (section 6) using GMRES, but it can be used with other Krylov subspace methods as well. Although we focus on structured matrices arising from the discretization of partial differential equations on regular grids, it must be noted that the method also works for sparse unstructured matrices whose graphs can be well partitioned (small edge or vertex separators). The AMML(s) reordering can be used to avoid communication not only in parallel computations (between processors, shared-memory cores or between CPU and GPU) but also in sequential computations (between different levels of the memory hierarchy). Thus in this paper we will use the term *processor* to indicate the component performing the computation and *fetch* to indicate the movement of data (read, copy, or receive message). In section 6 we show that our reordering does not affect the convergence of ILU0 preconditioned GMRES as compared to k-way reordering, and we model the expected performance of our preconditioner based on the needed memory and the redundant flops introduced to reduce the communication.

2. Preliminaries. In this section we give the definitions of the concepts and the terms that we use in this paper as graphs, nested dissection, and k-way partitioning. Then we briefly introduce CA-GMRES, the matrix powers kernel, and associated graph partitioning problems.

2.1. Graphs and partitioning. The structure of an unsymmetric $n \times n$ matrix A can be represented by using a directed graph $G(A) = (V, E)$, where V is a set of vertices and E is a set of edges. A vertex v_i is associated with each row/column i of the matrix A. An oriented edge $e_{j,i}$ from vertex j to vertex i is associated with each nonzero element $A(j, i) \neq 0$. A weight w_i and a cost $c_{j,i}$ are assigned to every vertex v_i and edge $e_{j,i}$, respectively. Let B be a subgraph of $G(A)$ ($B \subset G(A)$); then $V(B)$ is the set of vertices of B, $V(B) \subset V(G(A))$, and $E(B)$ is the set of edges of B, $E(B) \subset E(G(A))$. Let i and j be two vertices of $G(A)$. The vertex j is reachable from the vertex i if and only if there exists a path of directed edges from i to j. The length of the path is equal to the number of visited vertices excluding i. Let S be any subset of vertices of $G(A)$. The set $R(G(A), S)$ denotes the set of vertices reachable from any vertex in S and includes S ($S \subset R(G(A), S)$). The set $R(G(A), S, m)$ denotes the set of vertices reachable by paths of length at most m from any vertex in S. The set $R(G(A), S, 1)$ is the set of adjacent vertices of S in the graph of A, and we denote it by $Adj(G(A), S)$. The set $Adj(G(A), S) - S$ is the open set of adjacent vertices of S in

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the graph of A, and we denote it by $opAdj(G(A), S)$. Note that an undirected graph of a symmetric matrix is a special case of directed graphs where all the edges are bidirectional. Since there is no need to specify a direction, the edges are undirected.

We use MATLAB notation for matrices and vectors. For example, given a vector y of size $n \times 1$ and a set of indices α (which correspond to vertices in the graph of A), $y(\alpha)$ is the vector formed by the subset of the entries of y whose indices belong to α. For a matrix A, $A(\alpha, :)$ is a submatrix formed by the subset of the rows of A whose indices belong to α. Similarly, $A(:, \alpha)$ is a submatrix formed by the subset of the columns of A whose indices belong to α. We have $A(\alpha, \beta) = [A(\alpha, :)](:, \beta)$, the β columns of the submatrix $A(\alpha, :)$.

To exploit parallelism and reduce communication when solving a linear system $Ax = b$ using an iterative solver, the input matrix A is often reordered using graph partitioning techniques such as nested dissection [17, 27] or k-way graph partitioning [26]. These techniques assume that the matrix A is symmetric and its graph is undirected. In case A is unsymmetric, then the undirected graph of $A + A^t$ is used to define a partition for the matrix A. Graph partitioning techniques can be applied on both graphs and hypergraphs (see, e.g., [8, 2]), and they rely on identifying either edge/hyperedge separators or vertex separators. Since our preconditioner can be used with both techniques, we describe them briefly in the following in the context of undirected graphs.

Nested dissection [17] is a divide-and-conquer graph partitioning strategy based on vertex separators. For undirected graphs, at each step of dissection, a set of vertices that forms a separator is sought, which splits the graph into two disjoint subgraphs once the vertices of the separator are removed. We refer to the two subgraphs as $\Omega_{1,1}$ and $\Omega_{1,2}$, and to the separator as $\Sigma_{1,1}$. The vertices of the first subgraph are numbered first, then those of the second subgraph, and finally those of the separator. The corresponding matrix has the following structure:

$$A = \begin{pmatrix} A_{11} & & A_{13} \\ & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}.$$

The algorithm then continues recursively on the two subgraphs $\Omega_{1,1}$ and $\Omega_{1,2}$. The separators' subgraphs and the subdomains' subgraphs introduced at level i of the nested dissection are denoted by $\Sigma_{i,j}$ and $\Omega_{i,l}$, respectively, where $j \leq 2^{i-1}$, $l \leq 2^i$, $i \leq t$, and $t = \log(P)$ (P is the number of processors). The vertices of the separators and the final subdomains are denoted by $S_{i,j} = V(\Sigma_{i,j})$ and $D_l = V(\Omega_{t,l})$, respectively. Thus, at level i we introduce 2^{i-1} new separators and 2^i new subdomains. We illustrate nested dissection in Figure 2, which displays the graph of a two-dimensional (2D) five-point stencil matrix A, where the vertices are represented by their indices. For clarity of the figure, the edges are not shown in the graph, but it must be noted that there are oriented edges connecting each vertex to its north, south, east, and west neighbors. This corresponds to a symmetric matrix with a maximum of five nonzeros per row. All the following figures of graphs have the same format. In addition, the directed graphs of L and U coincide with that of A, where the L and U matrices are obtained from the ILU0 factorization of A. By disregarding the colored lines, Figure 2 presents the subdomains and the separators obtained by using three levels of nested dissection.

K-way graph partitioning by edge separators aims at partitioning a graph $G = (V, E)$ into $k > 1$ parts $\pi = \{\Omega_1, \Omega_2, \ldots, \Omega_{k-1}, \Omega_k\}$, where the k parts are nonempty

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

($\Omega_i \neq \phi$, $\Omega_i \subset V$ for $1 \leq i \leq k$, and $\cup_{i=1}^{k} \Omega_i = V$) and the partition π respects the balance criterion

$$W_i \leq (1+\epsilon) W_{avg}, \tag{2.1}$$

where $W_i = \sum_{v_j \in \Omega_i} w_j$ is the weight associated to each part Ω_i, $W_{avg} = (\sum_{v_i \in V} w_i)/k$ is the perfect weight, ϵ is the maximum allowed imbalance ratio, and w_j is the weight associated to vertex v_j. In addition, k-way graph partitioning minimizes the cutsize of the partition π,

$$\chi(\pi) = \sum_{e_{i,j} \in \mathcal{E}_E} c_{i,j}, \tag{2.2}$$

where $\mathcal{E}_E$ is the set of external edges of π, $\mathcal{E}_E \subset E$, and $c_{i,j}$ is the cost of the edge $e_{i,j}$, where $i < j$. In graph partitioning, the term edge-cut indicates that the two vertices that are connected by an edge belong to two different partitions. The edge which is cut is referred to as an external edge. In this paper, we use $w_j = 1$ and $c_{i,j} = 1$.

2.2. CA-GMRES and the matrix powers kernel. Communication avoiding GMRES [24, 29] is based on s-step GMRES (see, e.g., [33, 15] and references therein), where several changes have been introduced such that the communication is minimized. First, the Arnoldi(s) algorithm, which restarts after computing and orthogonalizing s vectors of the Krylov subspace basis, is replaced by the communication avoiding Arnoldi(s,t) algorithm, which restarts after t calls of the matrix powers kernel. Second, the s vectors obtained from the matrix powers kernel are orthogonalized against the previously computed vectors using block Gram–Schmidt and against each others using TSQR (tall skinny QR) [12]. Finally the upper Hessenberg matrix is reconstructed. In total, as explained in [24], CA-GMRES communicates a factor of $\Theta(s)$ fewer messages in parallel than GMRES. In a sequential machine with two levels of fast and slow memory, it reads the sparse matrix and vectors from slow memory to fast memory a factor of $\Theta(s)$ fewer times.

To minimize communication in a parallel setting, the s monomial basis vectors of the Krylov subspace $[y, Ay, A^2y, \ldots, A^sy]$ are computed with no communication using the so-called matrix powers kernel [13]. This requires ghosting and computing redundantly on each processor the data required for computing its part of the vectors with no communication. Note that throughout this paper we use the term ghosting to denote the storage of redundant data, of vectors or matrices, that do not belong to the processor's assigned domain or part, but are needed for future computations. First, the data and the work is split between P processors. Each processor is assigned a part α of the input vector y_0 ($y_0(\alpha)$) and $A(\alpha, :)$, where $\alpha \subseteq V(G(A))$. Then, each processor has to compute the same part α of $y_1 = Ay_0$, $y_2 = Ay_1$, till $y_s = Ay_{s-1}$ without communicating with other processors. To do so, each processor fetches all the data needed from the neighboring processors, to compute its part α of the s vectors. Thus, to compute $y_s(\alpha)$, each processor should receive the missing data of $y_0(\eta_s)$ and $A(\eta_s, :)$ from its neighboring processors and store it redundantly, where $\eta_s = R(G(A), \alpha, s)$. Finally, each processor computes the set $R(G(A), \alpha, s-i)$ of the vectors y_i for $i = 1, 2, \ldots, s$ without any communication with the other processors. Figure 1 shows the needed data for each step on Domain 1 with $s = 3$ where the graph of a 2D five-point stencil matrix with $n = 200$ is partitioned into four subdomains. Since $\alpha \subseteq \eta_i = R(G(A), \alpha, i)$, it is obvious that the more steps are performed, the more redundant data is ghosted and flops are computed. In addition, partitioning

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

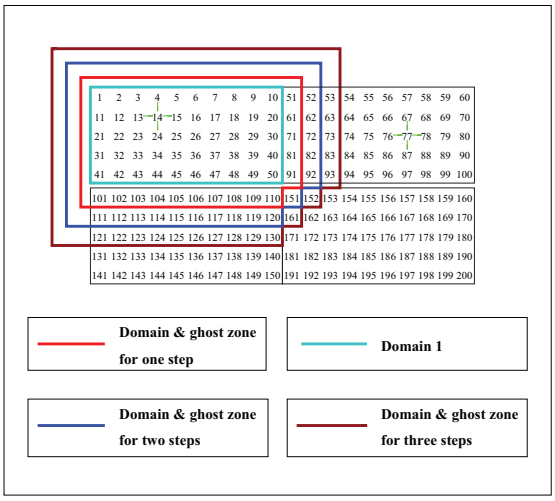


FIG. 1. *Data needed to compute three multiplications $y_i = Ay_{i-1}$ on Domain 1 using the matrix powers kernel.*

plays an important role in reducing the size of the ghost data and balancing the load among processors. In [8] and [7], hypergraph partitioning models were introduced to reduce the volume of communication in matrix vector multiplication and matrix powers kernel.

3. ILU matrix powers kernel. The algorithm for solving a left-preconditioned system by using Krylov subspace methods is the same as for a nonpreconditioned system, with the exception of the matrix vector multiplications. For example, GMRES requires computing $y = Ax$, while the preconditioned version computes $y = M^{-1}Ax$, where M is a preconditioner. Similarly, a preconditioned CA-GMRES relies on a preconditioned matrix powers kernel. And constructing a communication avoiding preconditioner is equivalent to building a preconditioned matrix powers kernel which computes the set of s basis vectors $\{M^{-1}Ay_0, (M^{-1}A)^2y_0, \ldots, (M^{-1}A)^{s-1}y_0, (M^{-1}A)^s y_0\}$ while minimizing communication, where y_0 is a starting vector and $s \geq 1$. In this section we first define the partitioning problem associated with computing s vectors of the preconditioned matrix powers kernel. Then we identify dependencies involved in the computation of the preconditioned matrix powers kernel.

In this paper we focus on the incomplete LU preconditioner $M = LU$, which consists of finding a lower triangular matrix L and an upper triangular matrix U that approximate the input matrix A up to some error, i.e., $A = LU + R$, where R is the residual matrix. There are different approaches for computing the incomplete L and U factors, one of which is by defining a drop-tolerance such that all the entries in the matrices L and U that are smaller than this drop-tolerance are dropped. Another form of incomplete LU decomposition is ILU0, which produces L and U matrices that have the same sparsity pattern as the lower and upper parts of A, respectively. The third is ILU(k), which allows some fill-in in the L and U sparsity pattern up to some level k. For more information on the different ILU decompositions refer to Chapter 10 of [30].

The partitioning problem. In the following, we consider that the matrix A and the factors L and U of the preconditioner are distributed block row-wise over P pro-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

cessors. A communication avoiding preconditioner can be obtained if we are able to ghost some data and perform some redundant computation such that s basis vectors $\{M^{-1}Ay_0, (M^{-1}A)^2y_0, \ldots, (M^{-1}A)^{s-1}y_0, (M^{-1}A)^s y_0\}$ can be computed with no communication. To obtain an efficient preconditioner, ideally we would like to minimize the size of the ghost data while balancing the load among processors. In this paper, we use Metis's edge separator (k-way) and vertex separator (nested dissection). The obtained partition satisfies the balance criterion (2.1) and minimizes the cutsize (2.2).

We are interested in ILU preconditioners where $M = LU$ is obtained from the ILU factorization of A. In practice, $(LU)^{-1}A$ is never computed explicitly. In fact, determining $y_i = (LU)^{-1}Ay_{i-1}$ during the computation of the preconditioned matrix powers kernel is equivalent to performing the following three steps:

1. Compute $f = Ay_{i-1,}$.
2. Solve $LUy_i = f$; i.e., solve $Lz = f$ by forward substitution.
3. Solve $Uy_i = z$ by backward substitution.

Hence in practice the inverse of the factors LU is never computed, and the multiplication $(LU)^{-1}A$ is not performed when computing $y_i = (LU)^{-1}Ay_{i-1}$. Thus, we use a heuristic which starts by partitioning the graph of A using either nested dissection or k-way graph partitioning, and we use the same partition for L and U. Note that other graph partitioning techniques and even hypergraph partitioning techniques can be used to partition A. But for simplicity we base our work on graphs. Then, we introduce heuristics to reduce the redundant data and computation needed to perform the forward and backward substitution by each processor without communication. However, before we discuss the heuristics, we first introduce the ILU matrix powers kernel.

ILU preconditioned matrix powers kernel. Our ILU matrix powers kernel is based on the matrix powers kernel, with the exception that A is replaced by $(LU)^{-1}A$, since we have a preconditioned system. Given a partition $\pi = \{\Omega_1, \Omega_2, \ldots, \Omega_{k-1}, \Omega_k\}$ of the graph of $(LU)^{-1}A$, let $\alpha_0 = V(\Omega_p)$ be the set of vertices assigned to processor j, where processor j must compute $\{y_1(\alpha_0), y_2(\alpha_0), \ldots, y_{s-1}(\alpha_0), y_s(\alpha_0)\}$ with $y_i = (LU)^{-1}Ay_{i-1}$.

In the following we describe an algorithm that allows a processor j to perform s steps with no communication, by ghosting parts of A, L, U, and y_0 on processor j before starting the s iterations. Consider that at some step i, processor j needs to compute $y_i(\alpha)$. The last operation that leads to the computation of $y_i(\alpha)$ is the backward substitution $Uy_i(\alpha) = z$. Due to the dependencies in the backward substitution, the equations α are not sufficient for computing $y_i(\alpha)$. Gilbert and Peierls showed in [19] that the set of equations that need to be solved in addition to α is the set of reachable vertices from α in the graph of U. Thus, the equations $\beta = R(G(U), \alpha)$ are necessary and sufficient for solving the equations α. In other words, if the processor j has in its memory $U(\beta, \beta)$ and $z(\beta)$, then it can solve with no communication the reduced system $U(\beta, \beta)y_i(\beta) = z(\beta)$. This is because by definition of reachable sets, there are no edges between the vertices in the set β and other vertices. Thus all the columns in $U(\beta, :)$ except the β columns are zero columns. To solve the reduced system $U(\beta, \beta)y_i(\beta) = z(\beta)$, processor j needs to have in his memory $z(\beta)$ beforehand. And this is equivalent to solving the set of equations $\gamma = R(G(L), \beta)$ of $Lz = f$. Similarly, processor j solves the reduced system $L(\gamma, \gamma)z_i(\gamma) = f(\gamma)$, where $f(\gamma)$ must be available. Computing $f(\gamma)$ is equivalent to computing $A(\gamma, :)y_{i-1}$. However, it must be noted that the entire vector y_{i-1} is not used, since for computing this subset of matrix vector multiplication processor j needs only $y_{i-1}(\delta)$, where $\delta = Adj(G(A), \gamma)$.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Therefore, it computes $A(\gamma, \delta)y_{i-1}(\delta)$.

Hence to compute the first step, $y_1 = (LU)^{-1}Ay_0$, processor j fetches $y_0(\delta_1)$, $A(\gamma_1, \delta_1)$, $L(\gamma_1, \gamma_1)$, and $U(\beta_1, \beta_1)$. To perform another step, we simply let $\alpha_1 = \delta_1$ and do the same analysis. This procedure is summarized in Algorithm 1. Thus to compute s steps of $y_i(\alpha_0) = [(LU)^{-1}Ay_{i-1}](\alpha_0)$, processor j fetches $y_0(\delta_s)$, $A(\gamma_s, \delta_s)$, $L(\gamma_s, \gamma_s)$, and $U(\beta_s, \beta_s)$. Note that $\alpha_{i-1} \subseteq \beta_i \subseteq \gamma_i \subseteq \delta_i \subseteq \alpha_i$, for $i = 1$ until s. After fetching all the data needed, processor j computes its part using Algorithm 2. Thus Algorithm 1 has to output all the subsets β_j, γ_j, and δ_j for $1 \leq j \leq s$, which will be used in Algorithm 2. Note that although processor j needs to compute only $y_i(\alpha_0)$, where $1 \leq i \leq s$, it computes some redundant flops in order to avoid communication.

Algorithm 1 s-step dependencies.

Input: $G(A)$, $G(L)$, $G(U)$; s, number of steps; α_0, subset of unknowns
Output: Sets β_j, γ_j, and δ_j for all $j = 1$ till s
1: **for** $i = 1$ to s
2: Find $\beta_i = R(G(U), \alpha_{i-1})$.
3: Find $\gamma_i = R(G(L), \beta_i)$.
4: Find $\delta_i = Adj(G(A), \gamma_i)$.
5: Set $\alpha_i = \delta_i$.
6: **end for**

Algorithm 2 CA-ILU s-step spmv ($A(\gamma_s, \delta_s)$, $L(\gamma_s, \gamma_s)$, $U(\beta_s, \beta_s)$, s, α_0).

Input: $A(\gamma_s, \delta_s)$, $L(\gamma_s, \gamma_s)$, $U(\beta_s, \beta_s)$, s, number of steps, α_0, subset of unknowns
Output: $y_i(\alpha_0)$, where $1 \leq i \leq s$
1: **for** $i = s$ to 1
2: Compute $f(\gamma_i) = A(\gamma_i, \delta_i)y_{j-s}(\delta_i)$.
3: Solve $L(\gamma_i, \gamma_i)z_{i-s+1}(\gamma_i) = f(\gamma_i)$.
4: Solve $U(\beta_i, \beta_i)y_{i-s+1}(\beta_i) = z(\beta_i)$.
5: Save $y_{i-s+1}(\alpha_0)$, which is the part that processor j has to compute.
6: **end for**

The ILU matrix powers kernel presented here is general and works for any matrices L and U. However, it is not sufficient to reduce or avoid communication, since the reachable sets β_i and γ_i might be much larger than α_{i-1} and β_i, respectively. A communication avoiding method is efficient if there is a good trade-off between the number of redundant flops and the amount of communication which was reduced. This reflects in the runtime of the algorithm. In other words, if while performing three or four steps of a CA-ILU preconditioned iterative solver, each processor ends up needing all the data and computing almost entirely the vectors y_i, then either we are not exploiting the parallelism of our problem efficiently or the problem is not fit for communication avoiding techniques.

This is indeed the case if Algorithm 2 is applied to the 2D five-point stencil matrix whose graph, presented in Figure 3(a), is partitioned into four subdomains by using k-way partitioning. To perform only one step of an iterative solver preconditioned by CA-ILU with no communication, processor 1 (which computes Domain 1 in the figure) ends up computing the entire vector y_i and fetching all the matrices A, L, and U, where the L and U matrices are obtained from the ILU0 factorization. This cancels any possible effect of the parallelization of the problem and shows that what works for the matrix powers kernel of the form $y_i = Ay_{i-1}$ does not work for the same kernel where the multiplication is $y_i = (LU)^{-1}Ay_{i-1}$. Nested dissection might

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

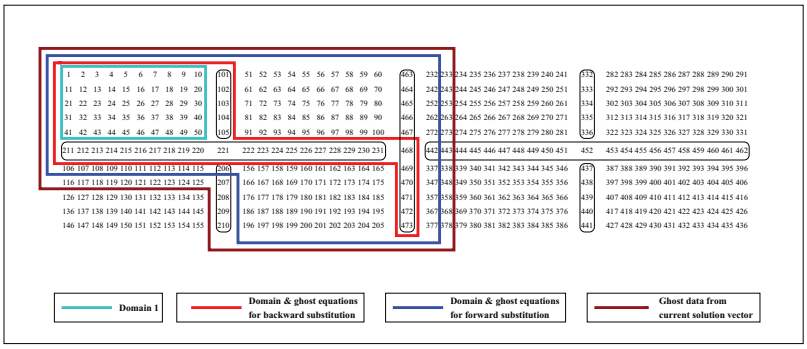


FIG. 2. *An* 11*-by-*43 5*-point stencil, partitioned into* 8 *subdomains using* 7 *separators. The data needed to compute one* $y_i = (LU)^{-1}Ay_{i-1}$ *on Domain* 1 *is shown, where* L *and* U *matrices are obtained from* $ILU(0)$.

look like a better solution since it splits the domain into independent subdomains that interact only with the separators. However, it is not sufficient either to obtain a communication avoiding preconditioner. This can be seen in Figure 2. To compute one matrix-vector multiplication of the form $y_i = (LU)^{-1}Ay_{i-1}$, processor 1 has to fetch half of matrix A, half of matrix L, half of the vector y_{i-1}, almost the quarter of matrix U, and perform the associated computation.

This shows that partitioning the graph of the input matrix A by using techniques such as nested dissection or k-way is not sufficient to reduce communication in the preconditioned matrix powers kernel. This is because both the matrix vector multiplication and the forward/backward substitutions need to be performed in a communication avoiding manner. In the next section, we introduce a new reordering that reduces the communication. Note that in Metis library [25] the subdomains with number of vertices greater than 200 do not have a natural ordering as shown in Figure 2, but they are partitioned recursively using nested dissection into smaller subdomains and separators. And this tends to reduce communication and the computed redundant flops, as we will detail later in the experimental section 6.

4. Alternating min-max layers renumbering for ILU0 matrix powers kernel. In this section we describe a reordering that allows us to compute and apply an ILU0 preconditioner in parallel in a preconditioned Krylov subspace method with no communication. Let A be a matrix whose graph is partitioned into P subdomains $\pi = \{\Omega_1, \Omega_2, \ldots, \Omega_P\}$ using k-way graph partitioning. To compute and apply in parallel the preconditioner, each processor j is assigned one subdomain Ω_j over which it should compute the s multiplications $y_{i+1}(\overline{\alpha}_0^{(j)}) = ((LU)^{-1}Ay_i)(\overline{\alpha}_0^{(j)})$, where $0 \leq i \leq s-1$, $\overline{\alpha}_0^{(j)} = V(\Omega_j)$, and L and U are obtained from the ILU0 factorization of A. The goal of our reordering algorithm is to renumber the vertices of each subdomain, $\overline{\alpha}_0^{(j)} = V(\Omega_j)$ such that processor j can compute its assigned part of the s multiplications without any communication. As explained in the previous section, k-way partitioning and nested dissection alone are not sufficient to reduce data movement and redundant flops in the ILU0 matrix powers kernel. However, our new reordering, which is applied locally on the vertices of the subdomains Ω_j, for $j = 1, 2, \ldots, P$, obtained after a graph partitioning, reduces the ghost zones in Figures 2 and 3(a) not only for performing one step, but also for performing $2, 3, \ldots, s$ steps of the ILU0 matrix powers kernel. In this section we focus on the reordering after apply-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

ing k-way graph partitioning; we refer to this reordering as alternating min-max layers (AMML(s)) reordering. In the technical report [21] a similar reordering is described for a graph partitioned using nested dissection. In this paper we focus on ILU0, since this preconditioner produces L and U matrices that have the same sparsity pattern as A. Hence the graphs of L and U are known before the numerical values of the factors L and U are computed, and this allows us to reorder these graphs in order to avoid communication during the computation of the factors. This is not the case in drop-tolerance ILU, where the graphs of L and U are not known before the numerical computation of the factors, and hence avoiding communication is a more complex task there.

Note that k-way partitioning assigns to the vertices of every subdomain a set of consecutive distinct indices or numbers, *num*. The reordering does not change the set of indices assigned to every subdomain, but it changes the order of the vertices within each subdomain.

In this paper we present two versions of AMML(s) reordering, Algorithms 3 and 4. Both algorithms take as input the graph of A; the vertices of the subdomain to be reordered, $\overline{\alpha}^{(j)} = \overline{\alpha}_0^{(j)} = V(\Omega_j)$; D, the set of d neighboring subdomains $\overline{\alpha}^{(i)}$ that depend on $\overline{\alpha}^{(j)}$ (there exists at least one directed edge connecting a vertex in $\overline{\alpha}^{(i)}$ to a vertex in $\overline{\alpha}^{(j)}$ in the graph of A); the number of steps s to be performed; the set of indices *num* assigned to $\overline{\alpha}^{(j)}$; and *evenodd*, which defines in which order we want to number our nodes "first, last, first, ..." (odd) or "last, first, last, first, ..." (even). During the first call to the algorithm to reorder a subdomain, the initial parameters are set to *evenodd* = *even* and *num* to the set of indices assigned to the subdomain by k-way partitioning. As the name indicates, the goal of the alternating min-max layers of AMML(s) reordering is to reorder each subdomain $\overline{\alpha}^{(j)}$ with $2s$ adjacent alternating layers, starting from the layer that each of the neighboring subdomains $\overline{\alpha}^{(i)}$ is adjacent to, for $i \neq j$. This first layer is assigned with the largest indices.

Note that, in this section we use the term $\overline{\alpha}_0^{(j)}$ to denote $V(\Omega_j)$, rather than $\alpha_0^{(j)}$ which is used in section 3. The reason is that after reordering some vertices of $\overline{\alpha}_0^{(j)}$, the remaining vertices $\overline{\alpha}_1^{(j)}$ are a subset of $\overline{\alpha}_0^{(j)}$ ($\overline{\alpha}_s^{(j)} \subset \cdots \subset \overline{\alpha}_1^{(j)} \subset \overline{\alpha}_0^{(j)}$), whereas in section 3, $\alpha_1^{(j)} = \delta_1^{(j)}$ is a superset of $\alpha_0^{(j)}$ ($\alpha_0^{(j)} \subset \alpha_1^{(j)} \subset \cdots \subset \alpha_s^{(j)}$).

We explain first the reordering for applying the ILU0 preconditioner during one iteration of a Krylov subspace solver ($s = 1$), where the goal is to reduce the number of vertices in the sets $\beta_1^{(i)} = R(G(U), \overline{\alpha}_0^{(i)})$ and $\gamma_1^{(i)} = R(G(L), \beta_1^{(i)})$ for each subdomain Ω_i, where $\overline{\alpha}_0^{(i)} = V(\Omega_i)$. In the following, Figures 3(c) and 3(b) are used to explain the alternating reordering for one step ($s = 1$), while Figure 3(d) is used to display the reordering for two steps ($s = 2$). Further, Figure 4 shows the sparsity pattern of the matrix A in its natural ordering, with k-way reordering, and with k-way+AMML(2) reordering with the corresponding L^{-1} matrices, where L is obtained from the ILU0 factorization of A. To reduce globally the number of reachable vertices of interest in the graphs of L and U, the alternating reordering renumbers the vertices of each subdomain Ω_j such that locally on this subdomain the set of reachable vertices $\beta_1^{(i)} \cap \overline{\alpha}_0^{(j)}$ and $\gamma_1^{(i)} \cap \overline{\alpha}_0^{(j)}$ from all the other subdomains Ω_i is reduced. To do this, Algorithms 3 and 4 identify first the boundary vertices of each neighboring subdomain Ω_i in subdomain Ω_j, $bvU_0^{(j,i)} = \overline{\alpha}_0^{(j)} \cap Adj(G(A), \overline{\alpha}_0^{(i)})$, and assign to the sets $bvU_0^{(j,i)} = \overline{\alpha}_0^{(j)} \cap Adj(G(A), \overline{\alpha}_0^{(i)})$ the largest possible numbers in *num*. Then the algorithms identify the adjacent vertices of $bvU_0^{(j,i)}$ in the graph of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

A, $bvL_0^{(j,i)} = \overline{\alpha}_0^{(j)} \cap opAdj(G(A), bvU_0^{(j,i)})$, and assign to these sets the smallest numbers possible in num. Figure 3(b) displays the reordered graph obtained after this reordering, where the vertices of the sets $bvU_0^{(j,i)}$ and $bvL_0^{(j,i)}$ are kept in their natural ordering. The set $\beta_1^{(i)}$ is the set of vertices bounded by the red polygon, and the set $\gamma_1^{(i)}$ is the set of vertices bounded by the blue polygon for $i = 1$. In the worst case, the reachable set $\beta_1^{(i)}$ is equal to the union of the set $\overline{\alpha}_0^{(i)}$ with all the sets $bvU_0^{(j,k)}$ for $i \neq j$ and $j \neq k$. Similarly, the reachable set $\gamma_1^{(i)}$ is equal to $\beta_1^{(i)}$ and all the $bvL_0^{(j,k)}$. That is,

$$\beta_1^{(i)} \subseteq \overline{\alpha}_0^{(i)} \cup \bigcup_{\substack{j=1 \\ j \neq i}}^{P} bvU_0^{(j)} \quad \text{and} \quad \gamma_1^{(i)} \subseteq \beta_1^{(i)} \cup \bigcup_{\substack{j=1 \\ j \neq i}}^{P} bvL_0^{(j)},$$

where

$$bvU_0^{(j)} = \bigcup_{\substack{k=1 \\ k \neq j}}^{P} bvU_0^{(j,k)} \quad \text{and} \quad bvL_0^{(j)} = \bigcup_{\substack{k=1 \\ k \neq j}}^{P} bvL_0^{(j,k)}.$$

However, for each subdomain Ω_i, the reachable sets can be further reduced by reordering the vertices within the sets $bvU_0^{(j,i)}$ and $bvL_0^{(j,i)}$ for all the neighboring subdomains Ω_j. Algorithms 3 and 4 differ only in the approach used for reordering the vertices within the sets $bvU_0^{(j,i)}$ and $bvL_0^{(j,i)}$.

The remaining numbers in num are assigned to the remaining vertices $\overline{\alpha}_1^{(j)} = \overline{\alpha}_0^{(j)} - bvL_0^{(j)} - bvU_0^{(j)}$, where the $\alpha_1^{(j)}$ vertices are kept in their natural ordering as shown in Figure 3(c) (the black vertices), the $bvU_0^{(j)}$ vertices are the vertices in subdomain Ω_j that all the other subdomains $i \neq j$ depend on (the red and magenta vertices), and $bvL_0^{(j)} = Adj(G(A), bvU_0^{(j)})$ (the blue vertices). Then we get

$$\beta_1^{(i)} \subset Adj(G(A), \overline{\alpha}_0^{(i)}) \cup \zeta \quad \text{and} \quad \gamma_1^{(i)} \subset Adj(G(A), \beta_1^{(i)}) \cup \zeta^*,$$

where $\beta_1^{(i)} = R(G(U), \overline{\alpha}_0^{(i)})$, $\gamma_1^{(i)} = R(G(L), \beta_1^{(i)})$, $|\zeta| \ll |Adj(G(A), \overline{\alpha}_0^{(i)})|$, and $|\zeta^*| \ll |Adj(G(A), \beta_1^{(i)}|$. The sets ζ and ζ^* represent additional vertices that belong to the reachable set in addition to the adjacent set. For example, in Figure 3(c), the vertices 200 and 151 belong to the sets $\beta_1^{(1)} = R(G(U), \overline{\alpha}_0^{(1)})$ and $\gamma_1^{(1)} = R(G(L), \beta_1^{(1)})$, respectively. However, these vertices do not belong to the sets $Adj(G(A), \overline{\alpha}_0^{(1)})$ and $Adj(G(A), \beta_1^{(1)})$. Note that for matrices arising from 1D three-point stencil, 2D nine-point stencil, and 3D 27-point stencil discretizations, we have $\zeta = \zeta^* = \phi$.

When computing $s > 1$ multiplications of the form $y_i = (LU)^{-1} A y_{i-1}$ for $i = 1, 2, \ldots, s$, the goal of the alternating reordering is to reduce the number of vertices not only in the sets $\beta_1^{(j)}$ and $\gamma_1^{(j)}$, but also in the sets $\beta_i^{(j)}$ and $\gamma_i^{(j)}$, for $i = 1, 2, \ldots, s$. Thus we perform the same analysis as for the case of $s = 1$. We obtain a recursive reordering on the given set of vertices $\overline{\alpha}_0^{(j)}$ such that the two layers $bvU_0^{(j,i)} = \overline{\alpha}_0^{(j)} \cap Adj(G(A), \overline{\alpha}_0^{(i)})$ and $bvL_0^{(j,i)} = \overline{\alpha}_0^{(j)} \cap opAdj(G(A), bvU_0^{(j,i)})$ for all $i \neq j$ are assigned with the largest and smallest numbers, respectively. The remaining numbers are assigned to the unnumbered vertices $\overline{\alpha}_1^{(j)} = \overline{\alpha}_0^{(j)} - bvU_0^{(j)} - bvL_0^{(j)}$. But unlike the case of $s = 1$, the vertices $\overline{\alpha}_1^{(j)}$ are reordered recursively, to minimize the cardinality of

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 3 AMML ($\overline{\alpha}^{(j)}$, $D, s, evenodd, G(A), num$).

Input: $\overline{\alpha}^{(j)}$, the set of vertices of the subdomain to be reordered; G(A), the graph of A
Input: $D = \{\overline{\alpha}^{(i)} | Adj(G(A), \overline{\alpha}^{(i)}) \cap \overline{\alpha}^{(j)} \neq \phi, i = 1 : P, i \neq j\}$, set of d neighboring subdomains; s, the number of steps to be performed in the ILU matrix powers kernel
Input: *evenodd*, a tag that can be "even" or "odd"; *num*, set of indices assigned to $\overline{\alpha}^{(j)}$

Let $d = |D|$ be the cardinality of the set D.
if $s == 0$ **then**
  Number $\overline{\alpha}^{(j)}$ in any order, preferably in the natural order.
else
  for $i = 1$ to d **do** Find the vertices $bv_i = \overline{\alpha}^{(j)} \cap Adj(G(A), \overline{\alpha}^{(i)})$.
  for $i = 1$ to d **do** let $corners_i = \cup_{k=1}^{d} (bv_i \cap bv_k)$.
  for $i = 1$ to d **do**
    if evenodd = odd **then**
      Assign to the unnumbered vertices of bv_i the smallest numbers in *num*, num_{bv_i}
    else Assign to the unnumbered vertices of bv_i the largest numbers in *num*, num_{bv_i}
    end if
    Remove the numbers num_{bv_i} from *num* ($num = num - num_{bv_i}$).
    if $corners_i = \phi$ **then**
      Number the unnumbered vertices of bv_i with the indices num_{bv_i}, in any order.
    else
    Call AMML ($\boldsymbol{bv_i}$, $\boldsymbol{D, s, evenodd}$, $\boldsymbol{G(A)}$,$\boldsymbol{num_{bv_j}}$).
    end if
  end for
  Let $\overline{\alpha}^{(j)} = \overline{\alpha}^{(j)} - \cup_{i=1}^{d} bv_i$.
  if evenodd = even **then** Call AMML ($\boldsymbol{\overline{\alpha}^{(j)}}$, $\boldsymbol{\{bv_i \mid i = 1 : d\}, s, odd, G(A)num}$).
  else Call AMML ($\boldsymbol{\overline{\alpha}^{(j)}}$, $\boldsymbol{\{bv_i \mid i = 1 : d\}, s - 1, even, G(A), num}$).
  end if
end if

Algorithm 4 AMML**V2** ($\overline{\alpha}^{(j)}$, $D, s, evenodd, G(A), num$)

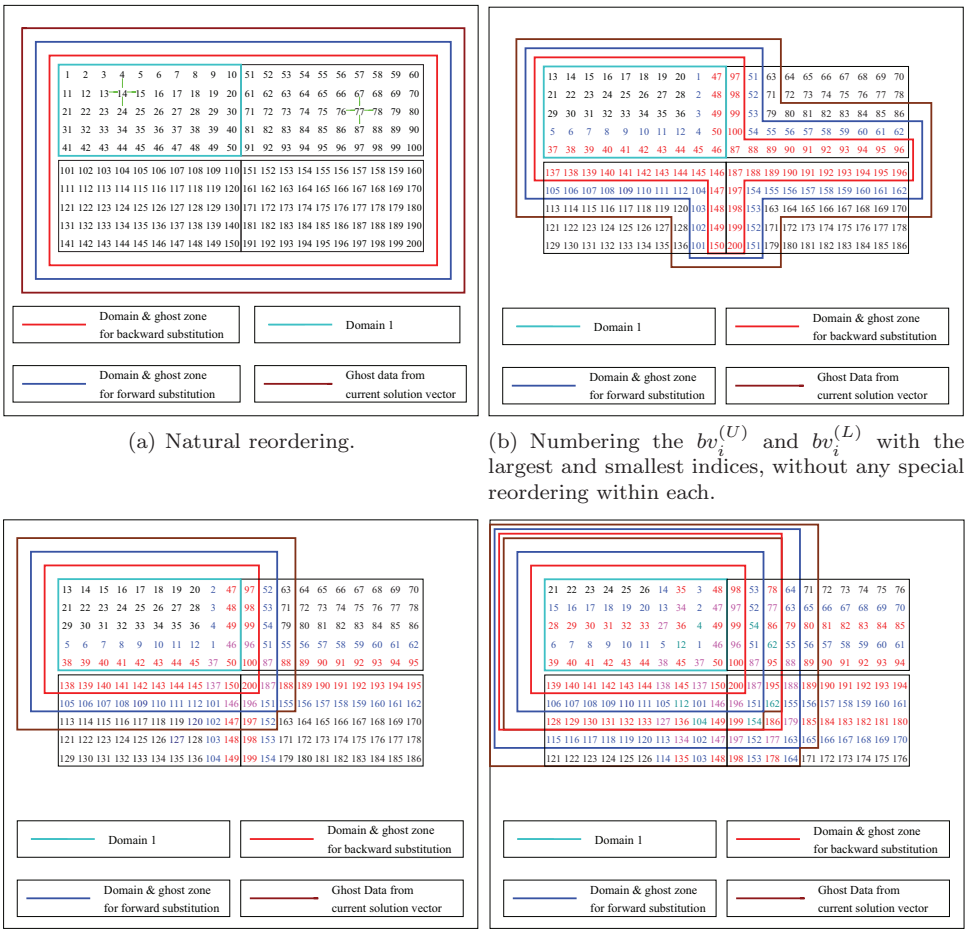
Input: $\overline{\alpha}^{(j)}$, the set of vertices of the subdomain to be reordered; G(A), the graph of A
Input: $D = \{\overline{\alpha}^{(i)} | Adj(G(A), \overline{\alpha}^{(i)}) \cap \overline{\alpha}^{(j)} \neq \phi, i = 1 : P, i \neq j\}$, set of d neighboring subdomains; s, the number of steps to be performed in the ILU matrix powers kernel
Input: *evenodd*, a tag that can be "even" or "odd"; *num*, set of indices assigned to $\overline{\alpha}^{(l)}$

Let $d = |D|$ be the cardinality of the set D
if $s == 0$ **then**
  Number $\overline{\alpha}^{(j)}$ in any order, preferably in the natural order.
else
  for $i = 1$ to d **do** Find the vertices $bv_i = \overline{\alpha}^{(j)} \cap Adj(G(A), \overline{\alpha}^{(i)})$
  Let $bv_j = \cup_{i=1}^{d} bv_i$
  if evenodd = odd **then**
    Assign to the vertices of bv_j, the smallest numbers in *num*, num_{bv_j}
  else Assign to the vertices of bv_j, the largest numbers in *num*, num_{bv_j}
  end if
  Remove the numbers num_{bv_j} from *num* ($num = num - num_{bv_j}$)
  Reorder bv_j using Nested Dissection to obtain an alternating reordering
  Let $\overline{\alpha}^{(j)} = \overline{\alpha}^{(j)} - bv_j$
  if evenodd = even **then** Call AMML**V2** ($\boldsymbol{\overline{\alpha}^{(j)}}$, $\boldsymbol{bv_j, s, odd, G(A), num}$)
  else Call AMML**V2** ($\boldsymbol{\overline{\alpha}^{(j)}}$, $\boldsymbol{bv_j, s - 1, even, G(A), num}$)
  end if
end if

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php



(a) Natural reordering.

(b) Numbering the $bv_i^{(U)}$ and $bv_i^{(L)}$ with the largest and smallest indices, without any special reordering within each.

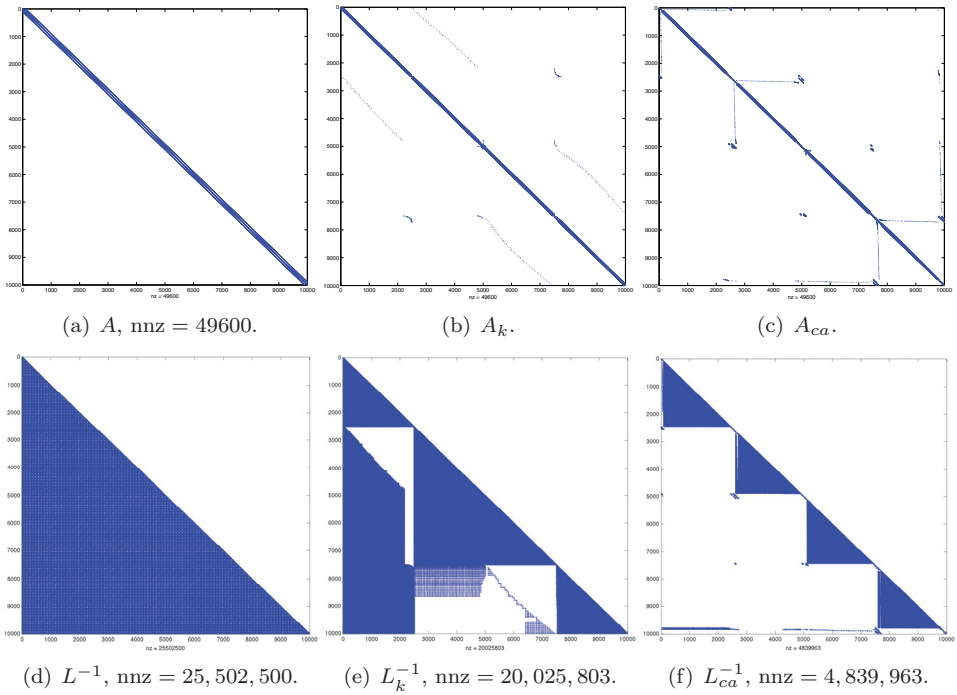
(c) AMML(1) reordering.

(d) AMML(2) reordering.

FIG. 3. *Data needed to compute* $y_i = (LU)^{-1}Ay_{i-1}$ *on Domain* 1 *using ILU matrix powers kernel for different reorderings where* $i = 1$ *in panels* (a), (b), *and* (c) *and* $i = 1, 2$ *in panel* (d). *The red and blue layers of vertices are numbered with the largest and smallest available indices for their corresponding domains. The magenta vertices belong to the red layer but are numbered with the smallest vertices to reduce the reachable set in graph of* L. *The cyan vertices belong to the blue layer but are numbered with the largest vertices to reduce the reachable set in graph of* U.

the sets $\beta_i^{(j)}$ and $\gamma_i^{(j)}$, for $i = 2, 3, \ldots, s$. First $bvU_1^{(j,i)} = \overline{\alpha}_1^{(j)} \cap opAdj(G(A), bvL_0^{(j,i)})$ and $bvL_1^{(j,i)} = \overline{\alpha}_1^{(j)} \cap opAdj(G(A), bvU_1^{(j,i)})$ for all $i \neq j$ are assigned with the largest and smallest numbers, respectively. Then if $s = 2$, the remaining numbers are assigned to $\alpha_2^{(j)} = \overline{\alpha}_1^{(j)} - bvU_1^{(j)} - bvL_1^{(j)}$, where the order of the vertices $\alpha_2^{(j)}$ is unchanged. Note that it is possible to reorder the vertices $\alpha_2^{(j)}$ in any manner. For example, Figure 4(c) shows the sparsity pattern of the k-way+AMML(2) reordered matrix A, where the $\alpha_2^{(j)}$ vertices are unchanged with $j = 1, 2, 3, 4$. And Figure 4(f) shows the sparsity pattern of the corresponding L^{-1}, where the diagonal blocks are dense. It is possible to reorder the vertices of $\alpha_2^{(j)}$ using nested dissection, to reduce the fill-ins in the diagonal blocks of L^{-1}; however, this will slow down the convergence of the preconditioner. That is why we prefer to leave the $\alpha_s^{(j)}$ vertices unchanged.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) A, nnz $= 49600$. (b) A_k. (c) A_{ca}.

(d) L^{-1}, nnz $= 25,502,500$. (e) L_k^{-1}, nnz $= 20,025,803$. (f) L_{ca}^{-1}, nnz $= 4,839,963$.

FIG. 4. *The fill-ins in the inverse of L obtained from the $ILU(0)$ factorization of a 2D five-point stencil matrix, in its natural ordering A, in its k-way reordered matrix A_k, and in its k-way+*AMML*(2) reordered version A_{ca}, where $U^{-1} = (L^{-1})^t$ and nnz is the number of nonzero values.*

If $s > 2$, then $\alpha_2^{(j)}$ is also reordered recursively for $s-2$ to minimize the cardinality of the sets $\beta_i^{(j)}$ and $\gamma_i^{(j)}$ for $i = 3, 4, \ldots, s$. The reordering is performed in s recursive steps. At each step, two layers,

$$bvU_t^{(j)} = \bigcup_{\substack{i=1 \\ i \neq j}}^{P} bvU_t^{(j,i)} \quad \text{and} \quad bvL_t^{(j)} = \bigcup_{\substack{i=1 \\ i \neq j}}^{P} bvL_t^{(j,i)},$$

are reordered, and this produces an alternating min-max $2s$ layer reordering. In Figure 3(d), where the graph is reordered in order to perform two multiplications, there are four alternating layers starting from the boundary vertices in every subdomain. Note that, similarly to the case of $s = 1$, the vertices of $bvU_t^{(j,i)}$ and $bvU_t^{(j,i)}$ for $t = 0, 1, \ldots, s-1$ have to be reordered to reduce the addition of unnecessary vertices to the reachable sets.

At each recursive call in Algorithms 3 and 4, either $bvU_t^{(j,i)}$ or $bvL_t^{(j,i)}$, denoted by bv_i, is reordered. The tag *evenodd* is used to decide which one to reorder. If $evenodd = even$, then the largest available numbers in the set *num* are assigned to $bvU_t^{(j,i)}$. Otherwise, the smallest numbers in the set *num* are assigned to $bvL_t^{(j,i)}$. In Algorithm 3, the $bvU_t^{(j,i)}$ and the $bvL_t^{(j,i)}$ are reordered by calling the algorithm recursively to ensure an alternating reordering within each. In case the vertices of $bvU_t^{(j,i)}$ ($bvL_t^{(j,i)}$) do not belong to any other $bvU_t^{(j,k)}$ ($bvL_t^{(j,k)}$), where $k \neq i$, i.e., $corners_j = \phi$, then the vertices of $bvU_t^{(j,i)}$ ($bvL_t^{(j,i)}$) are kept in their natural ordering. The reordering for one and two steps is shown in Figures 3(c) and 3(d). The only

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

difference in Algorithm 4 is that we let

$$bvU_t^{(j)} = \bigcup_{\substack{i=1 \\ i \neq j}}^{P} bvU_t^{(j,i)} \quad \text{and} \quad bvL_t^{(j)} = \bigcup_{\substack{i=1 \\ i \neq j}}^{P} bvL_t^{(j,i)},$$

denoted by bv_j, and then reorder each using nested dissection. Since nested dissection assigns to the vertices of the separators larger numbers than the two subdomains, and then continues partitioning each till the final subdomains are very small, then the obtained reordering of the $bvU_t^{(j)}$ and $bvL_t^{(j)}$ is very similar to an alternating reordering. If $s > 1$, the remaining vertices of $\overline{\alpha}^{(j)}$ are reordered recursively, as shown in Algorithms 3 and 4.

4.1. Complexity of AMML(s) reordering. We define the complexity of our alternating min-max layers reordering as being the number of times the vertices and the edges of the graph of A are visited in order to perform the reordering. The complexity of Algorithms 3 and 4 for reordering the vertices of a subdomain Ω_j for s steps is equivalent to the complexity of finding the alternating layers from vertices $\overline{\alpha}^{(i)}$ of each neighboring subdomain Ω_i and then reordering each of these layers, where $\overline{\alpha}^{(i)} = V(\Omega_i)$.

Both algorithms take as input all the sets of vertices of the d neighboring subdomains $\overline{\alpha}^{(i)}$ that depend on the vertices $\overline{\alpha}^{(j)}$ and find $bv_i = Adj(G(A), \overline{\alpha}^{(i)}) \cap \overline{\alpha}^{(j)}$, where $i = 1, 2, \ldots, d$. This means that for finding the sets bv_i for each neighboring subdomain Ω_i, all the vertices and edges of the subdomain Ω_i are visited. In other words, for each subdomain Ω_j, $d \times |V(\Omega_{max})|$ vertices and $d \times |E(\Omega_{max})|$ edges have to be visited, where $|\overline{\alpha}^{(i)}| \leq |V(\Omega_{max})|$ and $|E(\Omega_i)| \leq |E(\Omega_{max})|$ for $i \neq j$, and d is the number of neighboring subdomains. However, this is not necessary. We can perform a preprocessing step in which each processor Ω_i finds its subdomain's boundary vertices, S_i, that depend on some other vertex in $\overline{\alpha}^{(j)}$, where $i \neq j$. Then the algorithms can use S_i instead of $\overline{\alpha}^{(i)}$, where $|S_i| \ll |\overline{\alpha}^{(i)}|$. But to keep the presentation of the algorithms simple, $\overline{\alpha}^{(i)}$ was used instead. To find S_i for each subdomain Ω_i, $|\overline{\alpha}^{(i)}|$ vertices and $|E(\Omega_i)|$ edges are visited.

For each subdomain Ω_j, finding the alternating layers from S_i for each neighboring subdomain Ω_i requires visiting at most $\sum_{i=1}^{d} |R(G(A), S_i, 2s)|$ vertices and their associated edges. In Algorithm 3, in case $corners_i \neq \phi$, the vertices of the set bv_i and associated edges need to be visited again. So some fraction of $|\cup_{i=1}^{d}(R(G(A), S_i, 2s) \cap \overline{\alpha}^{(j)})|$ vertices and their associated edges are visited in this case. In the worst case, where $corners(i) \neq \phi$, for all the d neighboring subdomains $i = 1 : d$, $i \neq j$, the algorithm visits at most $|\cup_{i=1}^{d}(R(G(A), S_i, 2s) \cap \overline{\alpha}^{(j)})|$ vertices and associated edges. Hence reordering the vertices of a subdomain Ω_j, $\overline{\alpha}^{(j)}$ requires visiting at most

$$2\sum_{i=1}^{d} |R(G(A), S_i, 2s) \cap \overline{\alpha}^{(j)}| + \sum_{i=1}^{d} |S_i| \ll 2|\overline{\alpha}^{(j)}| + \sum_{i=1}^{d} \xi |\overline{\alpha}^{(i)}| \ll (2 + d\xi)|V(\Omega_{max})|$$

vertices and $(2 + d\xi)|E(\Omega_{max})|$ edges. Note that $|S_i| = \xi|\overline{\alpha}^{(i)}|$, where $0 \leq \xi \ll 1$ is the ratio of the cardinality of the boundary vertices to the cardinality of the subdomain's vertices. The quantity ξ should be very small since the number of boundary vertices in a subdomain is at most equal to the edge-cuts of that subdomain, and k-way partitioning aims at minimizing the edge-cuts.

Thus, to reorder $\overline{\alpha}^{(j)}$, in total at most $(3 + d\xi)|V(\Omega_{max})|$ vertices and $(3 + d\xi)|E(\Omega_{max})|$ edges are visited. Since the AMML(s) reordering is done in parallel

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

on P processors, its parallel complexity is upper bounded by $(3 + d\xi)(|V(\Omega_{max})| + |E(\Omega_{max})|)$. Hence our algorithm is of linear complexity with respect to $(|V(\Omega_{max})| + |E(\Omega_{max})|)$. In case the AMML(s) reordering is done sequentially, one processor loops over the vertices of the subdomains and reorders them, then the complexity is upper bounded by $(3+d\xi)P(|V(\Omega_{max})|+|E(\Omega_{max})|)$, where P is the number of subdomains.

5. CA-ILU0 preconditioner. In this section we summarize the different steps required for constructing the CA-ILU0 preconditioner, presented in Algorithm 5. The algorithm first reorders the input matrix by using a graph partitioning technique, and in this paper we consider the usage of k-way partitioning. The obtained matrix is further reordered using AMML(s) reordering. Note that the permutations applied to the matrix A are also applied to the vector b. Then, the redundant data needed by each processor is identified using Algorithm 1. The final step is to compute the ILU0 factorization of the reordered matrix to obtain the L and U matrices for preconditioning the system $Ax = b$. The ILU0 factorization can be done sequentially on one processor, where the needed parts of the L and U matrices have to be fetched by the processors before starting the computations in a Krylov subspace solver, or it can be done in parallel where each processor performs the ILU0 factorization of the augmented part of A to obtain the needed parts of L and U.

The ILU0 factorization of $A(\gamma_s^{(j)}, :)$ can be performed in parallel without any communication for the following reasons. Performing the ILU0 factorization of $A(\rho, :)$ requires computing the factorization of $A(\omega, :)$ beforehand, where $\omega = R(G(L), \rho)$. And by the definition of reachable sets, we have that $R(G(L), \gamma_s^{(j)}) = \gamma_s^{(j)}$. Hence, processor j has all the needed rows of A to perform the ILU0 factorization in parallel without any synchronization or communication, at the expense of doing some redundant computation.

Algorithm 5 Construction of CA-ILU0 preconditioner.

1: Partition the graph of A into P subdomains by using k-way graph partitioning.
2: Find a permutation using AMML(s) reordering (Algorithm 3 or 4).
3: Apply the permutation to matrix A.
4: Find the redundant/ghost data that each processor needs using Algorithm 1.
5: Have each processor i fetch its corresponding $A(\gamma_s^{(i)}, :)$.
6: Have each processor i perform the CA-ILU(0) factorization of $A(\gamma_s^{(i)}, :)$ to obtain the corresponding $L(\gamma_s^{(i)}, :)$ and $U(\gamma_s^{(i)}, :)$ matrices.

All the steps of the CA-ILU0 preconditioner construction are done in parallel. In addition, steps 2 and 6 in Algorithm 5 can be done in parallel without communication. Note that, after fetching the matrix $A(\gamma_s^{(j)}, :)$ and $y_0(\delta_s^{(j)})$, each processor can compute its part of the L and U factors of the preconditioner and the first s multiplication without any communication with other processors.

The CA-ILU0 preconditioner can be used with a classic Krylov subspace solver, allowing us to apply the left-preconditioner without any communication. In this case AMML(s) reordering has the parameter s set to 1.

On the other hand, the CA-ILU0 preconditioner can be used for ILU0 right-preconditioned systems by slightly modifying the ILU matrix powers kernel where AMML(s) reordering is unchanged. As for the split preconditioned system of the form $L^{-1}AU^{-1}y = L^{-1}b$ with $x = U^{-1}y$, CA-ILU0 preconditioner can also be used by slightly modifying the ILU matrix powers kernel and AMML(s) reordering. In this

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

case, when the tag *evenodd* = *even*, we assign to the layer at hand the smallest indices in *num* and the largest indices otherwise. This produces $2s$ alternating layers starting from small indices.

6. Expected numerical efficiency and performance of CA-ILU0 preconditioner. The numerical efficiency and performance of CA-ILU0 preconditioner depends on the convergence of GMRES for the CA-ILU0 preconditioned system, on the complexity of Amml(s) reordering of the input matrix A, and on the additional memory requirements and redundant flops of the ILU0 matrix powers kernel. We discuss the convergence of the CA-ILU0 preconditioned system using GMRES in section 6.1, the memory requirements and redundant flops in section 6.2. While we do not present here results of a parallel implementation of CA-ILU0, the results presented in this section show that CA-ILU0 can be expected to be faster in practice than implementations of the classic ILU0 based on different reordering strategies.

We use in our experiments Metis [25] for graph partitioning purposes. Metis provides two versions of multilevel k-way partitioning. We use the PartGraphKway version that minimizes edge-cuts, which is referred to as Kway in the plots of iterations, redundant flops, and memory. In addition, Metis provides a multilevel nested dissection version called NodeNDP. The Kway version has the fastest convergence but requires more memory and redundant flops than nested dissection versions. Hence the Kway version is a good candidate for our CA-ILU0 preconditioner if we choose appropriately the number of partitions and number of steps to obtain a good trade-off between the amount of communication reduced and the amount of redundant computation. For this reason, in sections 6.1 and 6.2 the plots correspond to the CA-ILU0 preconditioner based on Kway partitioning.

The difference between the CA-ILU0 preconditioner based on nested dissection and that based on k-way partitioning is the local reordering. When using nested dissection to partition the graph of A, the obtained subdomains are reordered with a modified Amml(s) reordering that produces $2s-1$ layers rather than $2s$ since the separators are numbered larger than the subdomains. This modified Amml(s) algorithm is the same as the Amml(s) algorithms (3 or 4). But instead of setting the tag *evenodd* = *even* initially, it is set to *odd*. The separators are also reordered in an alternating format with an additional step to Algorithms 3 and 4, as shown in the technical report [21].

Finally, we compare our CA-ILU0 preconditioner with the block Jacobi preconditioner in section 6.3.

6.1. Convergence. It is known that the convergence of ILU0 preconditioned systems depends on the ordering of the input matrix. The best convergence is often observed when the matrix is ordered using reverse Cuthill–McKee (RCM) or natural ordering, while the usage of k-way partitioning or nested dissection tends to lead to a slower convergence (see, for example, [14]). Hence, we first discuss the effect of our reordering on the convergence of ILU0 preconditioned system. Our goal is to study the convergence of the CA-ILU0 preconditioner. Since s-step GMRES can lead by itself to a slower convergence than that of a classic GMRES method, we use in our experiments the classic GMRES method. Table 1 gives a brief description of the test matrices used and the nature of the problems they arise from.

The first four matrices, referred to as Nh2D1, Nh2D2, Sky2D, and Sky3D, arise from the following boundary value problems discretized using FreeFem++ [22] with

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 1
The test matrices.

Matrix	Size	Nonzeros	Symmetric	2D/3D	Problem
NH2D1	40000	199200	Yes	2D	Boundary value
NH2D2	160000	798400	Yes	2D	Boundary value
SKY2D	40000	199200	Yes	2D	Skyscraper
SKY3D	64000	438400	Yes	3D	Skyscraper
BO1	1800	11670	Yes	3D	Black oil reservoir
BO2	14400	97080	Yes	3D	Black oil reservoir
UTM3060	3060	42211	No	3D	Electromagnetics
BCSSTK18	11948	149090	Yes	3D	Structural (stiffness matrix)
WATT2	1856	11550	No	3D	Computational fluid dynamics

a finite element P1 scheme:

$$\begin{aligned} -div(\kappa(x)\nabla u) &= f \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \partial\Omega_D, \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial\Omega_N, \end{aligned}$$

where $\Omega = [0,1]^n$ ($n = 2$ or 3) and $\partial\Omega_N = \partial\Omega \setminus \partial\Omega_D$. The tensor κ is the given coefficients of the partial differential operator. In the 2D case, we have $\partial\Omega_D = [0,1] \times \{0,1\}$, and in the 3D case, we have $\partial\Omega_D = [0,1] \times \{0,1\} \times [0,1]$. For a detailed description refer to [1]. We focus on the following two cases:

- NH2D1 and NH2D2: A nonhomogeneous problem with large jumps in the coefficients. The tensor κ is isotropic and discontinuous. It jumps from the constant value 10^3 in the ring $\frac{1}{2\sqrt{2}} \leq |x - c| \leq \frac{1}{2}$, $c = (\frac{1}{2}, \frac{1}{2})^T$, to 1 outside.
- SKY2D and SKY3D skyscraper problems: The tensor κ is isotropic and discontinuous. The domain contains many zones of high permeability which are isolated from each other. For SKY2D, we have

$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1) & \text{if } [10x_i] \text{ is odd, } i = 1, 2, \\ 1 & \text{otherwise,} \end{cases}$$

  where we note $[x]$ as the integer value of x.

The BO1 and BO2 matrices are from a simulation of a black oil reservoir model, based on a compositional triphase Darcy flow simulator (oil, water, and gas).[1] The permeability is heterogeneous, with jumps on the order of 2^8.

NH2D1 and SKY2D are discretized on a 200×200 2D Cartesian grid. NH2D2 is discretized on a 400×400 Cartesian grid. BO1, BO2, and SKY3D are discretized on a $15 \times 15 \times 8$ grid, a $30 \times 30 \times 16$ grid, and a $40 \times 40 \times 40$ grid, respectively. As for the matrices UTM3060, BCSSTK18, and WATT2, their full description can be found in [10]. Seven other matrices that arise from the boundary value problem of the convection-diffusion equations, used in [5, 11] for testing preconditioners, are described in Appendix A.

We compare the convergence of GMRES for the ILU0 preconditioned system where the matrix A is reordered using k-way + AMML(s) version1 reordering (Algorithm 3), k-way + AMML(s) version2 reordering (Algorithm 4), k-way reordering, and

[1] These matrices were provided to us by R. Masson, at that time at IFP Energies Nouvelles.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

the natural ordering of A for different number of partitions and for $s = 1, 2, 5, 10$. We set the GMRES tolerance to 10^{-8} and the maximum number of iterations to 500. Figure 5 shows the convergence behavior for the first eight matrices.

As expected, the ILU0 preconditioned system with the natural ordering of A converges faster than when A is reordered using k-way and the two versions of Amml(s). The convergence of the RCM ILU0 preconditioned system is not shown in the plots, but for the symmetric matrices Nh2D1 , Bo1 , Bo2 , Bcsstk18, and Sky2D it has the same convergence as natural ordering, while for Utm3060 it converges with two iterations fewer than natural ordering. For the matrix Nh2D1 , when A is reordered using k-way and k-way plus Amml(1), the preconditioned GMRES has the same rate of convergence. But as s increases and the more we reorder the matrix, the more iterations are needed for convergence. We notice the same behavior for the matrices Utm3060, Bo1 , and Bo2 (Figures 5(b), 5(c), and 5(d)). But for the matrix Bcsstk18, GMRES converges in a maximum of nine iterations for the different reorderings of A, as shown in Figure 5(e). It must be noted that without preconditioning, the Bcsstk18 system does not converge for the given tolerance, while for $tol = 10^{-6}$ it converges in 909 iterations.

For the matrices Bo1 and Bo2 , k-way plus Amml(s) version1 (Algorithm 3) has a better convergence than version 2 (Algorithm 4). However, for the matrices Nh2D1 and Utm3060 there is no clear winner. But for $s = 1$, version1 converges slightly better, since in version2 nested dissection reorders two layers of vertices.

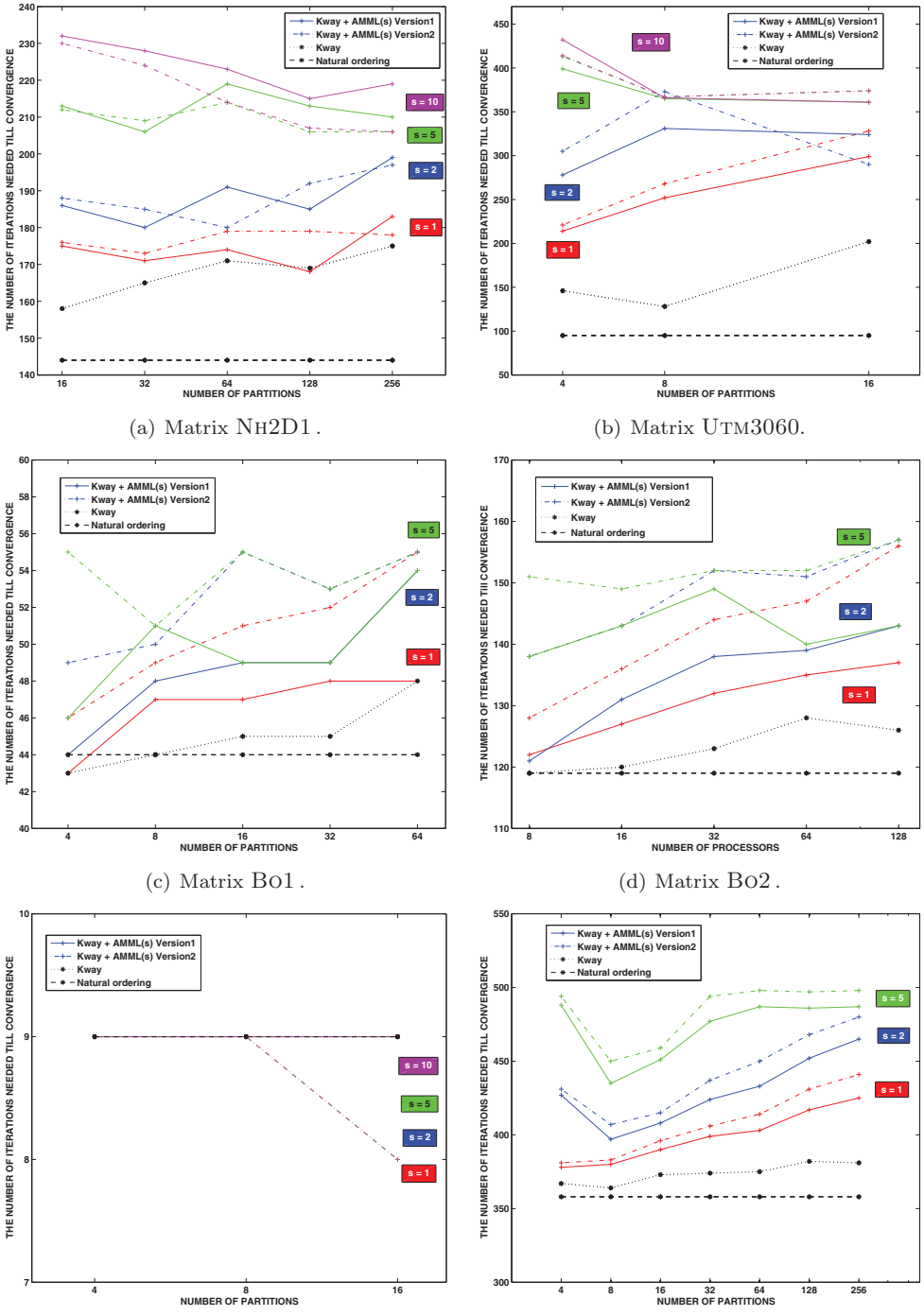
Table 6 in Appendix A shows the convergence behavior of ILU0 preconditioned GMRES with the different reorderings for Watt2 and the other seven matrices. We observe similar convergence behaviors with respect to number of partitions, steps s, and the two versions of Amml(s).

We can conclude that our CA-ILU0 preconditioned system where the matrix A is reordered using k-way and Amml(s) has a very similar convergence behavior to the ILU0 preconditioned system where the matrix A is only reordered using the k-way graph partitioning technique. Thus, our additional Amml(s) reordering of the matrix does not much affect its convergence, while it enhances its communication avoiding parallelizability.

6.2. Avoided communication versus memory requirements and redundant flops of the ILU0 matrix powers kernel. The ILU0 matrix powers kernel avoids communication by performing redundant flops and storing more vectors and data. Table 2 compares the needed memory and performed flops for s matrix vector multiplications on one subdomain/processor when using the nonpreconditioned CA-GMRES $(Ax, A^2x, \ldots, A^sx)$ and the CA-ILU0 preconditioned CA-GMRES $((LU)^{-1}Ax, \ldots, ((LU)^{-1}A)^sx)$ on 2D nine-point stencils and 3D 27-point stencils. We assume that each processor j has to compute the part $\alpha_0^{(j)} = V(\Omega_j)$ of the s matrix vector multiplication, where A is an $n \times n$ matrix, $|\alpha_0^{(j)}| \approx n/P = w^d$, $d = 2$ for 2D matrices and 3 for 3D matrices, and $w = (n/P)^{\frac{1}{d}}$ is the width of the square or cube subdomain. For simplicity, we refer to $\alpha_0^{(j)}$ as α in Table 2.

CA-GMRES requires storing s vectors of size $|R(G(A), \alpha_0^{(j)}, i)| \approx |(w + 2i)^d|$, $i = 1, 2, \ldots, s$, one vector of size $|\alpha_0^{(j)}|$, and the corresponding $|R(G(A), \alpha_0^{(j)}, s-1)|$ rows of the matrix A. Then it performs $\sum_{i=1}^{s}((w + 2(i-1))^d)(2 \times nnz - 1)$ flops, where nnz is the number of nonzeros per row (9 and 27). CA-ILU0 preconditioned CA-GMRES requires storing s vectors of size $|R(G(A), \alpha_0^{(j)}, 2(i-1))| \approx |(w + 4(i-1))^d|$, $i = 1, 2, \ldots, s$, one vector of size $|R(G(A), \alpha_0^{(j)}, 2s+1)|$, one vector of size

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) Matrix NH2D1.

(b) Matrix UTM3060.

(c) Matrix BO1.

(d) Matrix BO2.

(e) Matrix BCSSTK18.

(f) Matrix SKY2D.

FIG. 5. *The number of iterations needed for convergence for* six *matrices as a function of the number of partitions and steps s. The matrices are either in natural ordering or reordered using k-way partitioning (*Kway*), or k-way partitioning followed by* AMML*(s) based on Algorithm* 3 *(Version* 1*) or Algorithm* 4 *(Version* 2*). The number of partitions varies from* 4, 8, 16, 32, 64, 128 *to* 256 *depending on the size of the matrix. The number of steps s is either* 1 *(red),* 2 *(blue),* 5 *(green), or* 10 *(magenta).*

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

TABLE 2

Memory requirement and computational cost for performing s matrix vector multiplications on one subdomain α, for the nonpreconditioned CA-GMRES and for the CA-ILU0 preconditioned CA-GMRES.

Stencil		CA-GMRES	CA-ILU0 CA-GMRES
2D 9-pt	Memory	$(s+10)\|\alpha\| + 2(s^2+19s-18)\|\alpha\|^{\frac{1}{2}} + \frac{4}{3}s^3 + 38s^2 - \frac{214}{3}s + 36$	$(s+21)\|\alpha\| + 4(s^2+41s-4)\|\alpha\|^{\frac{1}{2}} + \frac{16}{3}s^3 + 328s^2 - \frac{184}{3}s + 24$
	Flops	$17s\|\alpha\| + 34(s^2-s)\|\alpha\|^{\frac{1}{2}} - 34s^2 + \frac{17}{3}(4s^3+2s)$	$35s\|\alpha\| + 4s(35s+26)\|\alpha\|^{\frac{1}{2}} + \frac{560}{3}s^3 + 208s^2 + \frac{172}{3}s$
3D 27-pt	Memory	$(s+28)\|\alpha\| + 3(s^2+55s-54)\|\alpha\|^{\frac{2}{3}} + [4s^3+330s^2-646s+324]\|\alpha\|^{\frac{1}{3}} + 2s^4 + 220s^3 - 646s^2 + 648s - 216$	$(s+57)\|\alpha\| + (6s^2+678s-78)\|\alpha\|^{\frac{2}{3}} + 4[4s^3+678s^2-154s+45]\|\alpha\|^{\frac{1}{3}} + 16s^4 + 8s[452s^2-154s+90] - 88$
	Flops	$53s\|\alpha\| + 159(s^2-s)\|\alpha\|^{\frac{2}{3}} + 106[2s^3 - 3s^2+s]\|\alpha\|^{\frac{1}{3}} + 106s^4 + 106(-2s^3+s^2)$	$107s\|\alpha\| + 6s[107s+80]\|\alpha\|^{\frac{2}{3}} + 2s[856s^2+960s+266]\|\alpha\|^{\frac{1}{3}} + 1712s^4 + 1064s^2 + 2560s^3$

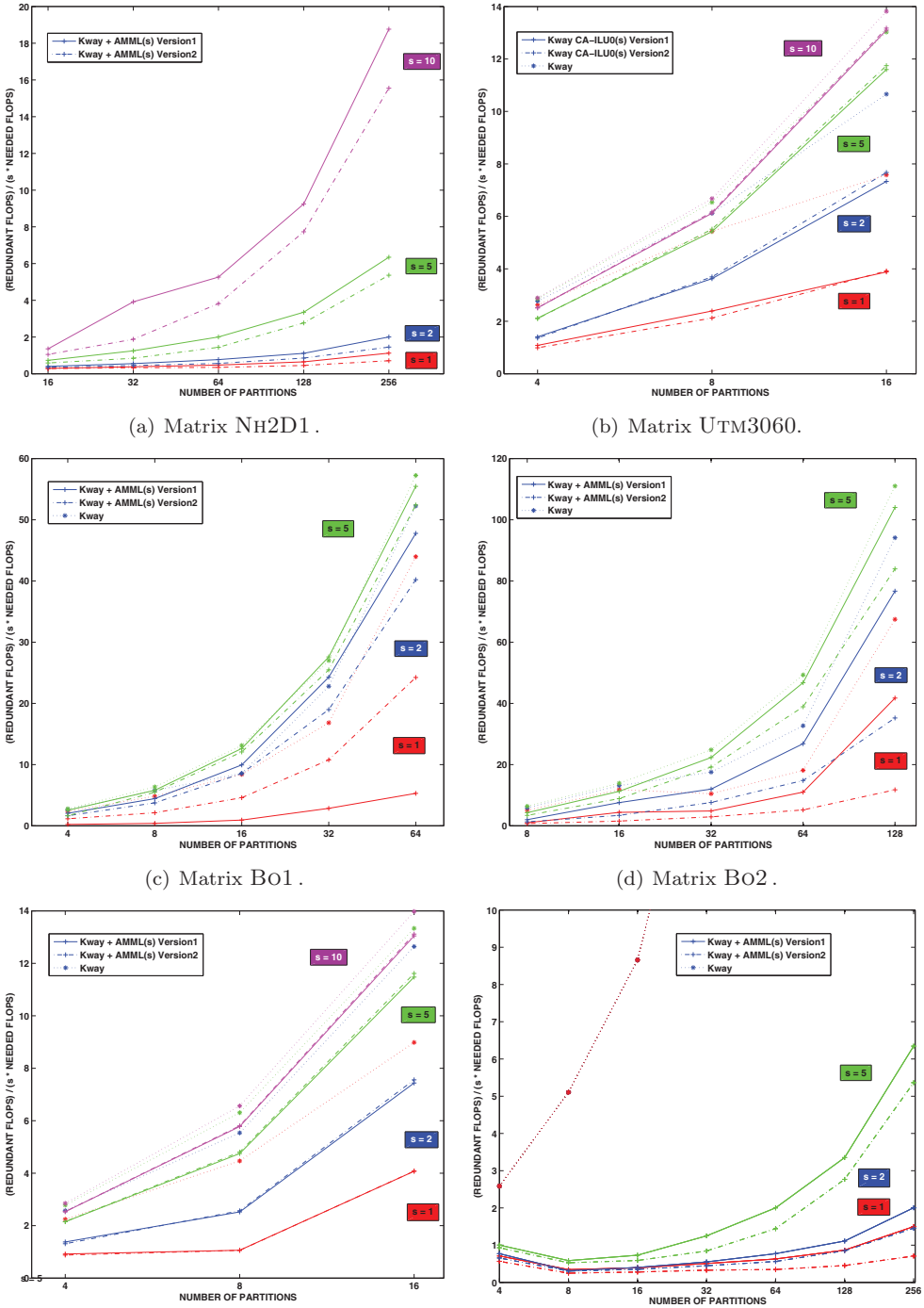
$|R(G(A),\alpha_0^{(j)},2s)|$, the corresponding $|R(G(A),\alpha_0^{(j)},2s)|$ rows of the matrices A and L, and the $|R(G(A),\alpha_0^{(j)},2s-1)|$ rows of the matrix U. Then it performs $\sum_{i=1}^{s}(2\times nnz-1)((w+4i)^d)$ flops to compute Ax. Solving the s lower triangular systems $(Lz=f)$ requires $\sum_{i=1}^{s}[1+2\times(nnz-1)/2]((w+4i)^d)$ flops. Similarly, solving the s upper triangular systems requires $\sum_{i=1}^{s}[1+2\times(nnz-1)/2]((w+4i-2)^d)$ flops. Note that the memory and flops of CA-GMRES and CA-ILU0 preconditioned CA-GMRES are governed by the same big O function.

Figure 6 plots the ratio of the total redundant flops in the ILU0 matrix powers kernel for $s=1,2,5,10$ with respect to the flops needed for computing s matrix vector multiplications in the sequential ILU0 preconditioned GMRES for six matrices in our set that are reordered using k-way, k-way+AMML(s) Version1, and k-way+AMML(s) Version2. Figure 7 plots the ratio of the ghost data that has to be saved in memory in the ILU0 matrix powers kernel for $s=1,2,5,10$ with respect to the needed memory in the matrix vector multiplication of the sequential ILU0 preconditioned GMRES for six matrices in our set that are reordered using k-way, k-way+AMML(s) Version1, and k-way+AMML(s) Version2. In Figures 6(a) and 7(a) we do not show the ratio of redundant flops to needed flops for the Kway reordered matrix NH2D1, since it is at least 10 times more than that of AMML(s) reordering. Hence the AMML(s) reordering leads to at least 90% fewer redundant flops and less ghost memory in the ILU0 matrix powers kernel than does METIS's k-way partitioning. This leads to a reduction of the volume of the communicated data at the end of the s steps.

In Figures 6(b) and 6(e), AMML(s) reordering performs from 10 to 50% fewer redundant flops than Kway partitioning in the ILU0 matrix powers kernel. On the other hand, in Figures 7(b) and 7(e), AMML(s) reordering needs 50% and 25% less ghost memory for $s=1$ and $s=2$, respectively. Whereas for $s=5$ and 10, AMML(s) reordering and Kway partitioning ratios are equal to $P-1$, where P is the number of processors or partitions. This means that each processor ends up needing all the matrices A, L, U and computing almost everything for a number of steps. Hence for matrices UTM3060 and BCSSTK18, s has to be less than 5.

We compare the ratio of redundant flops and ghost data of the two versions of AMML(s) reordering for the above three matrices. For matrix NH2D1 (Figures 6(a) and 7(a)), version2 performs fewer redundant flops. For matrix UTM3060 (Figures 6(b) and 7(b)), version1 has a slightly better performance for $s>1$. As for the matrix

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) Matrix NH2D1. (b) Matrix UTM3060.

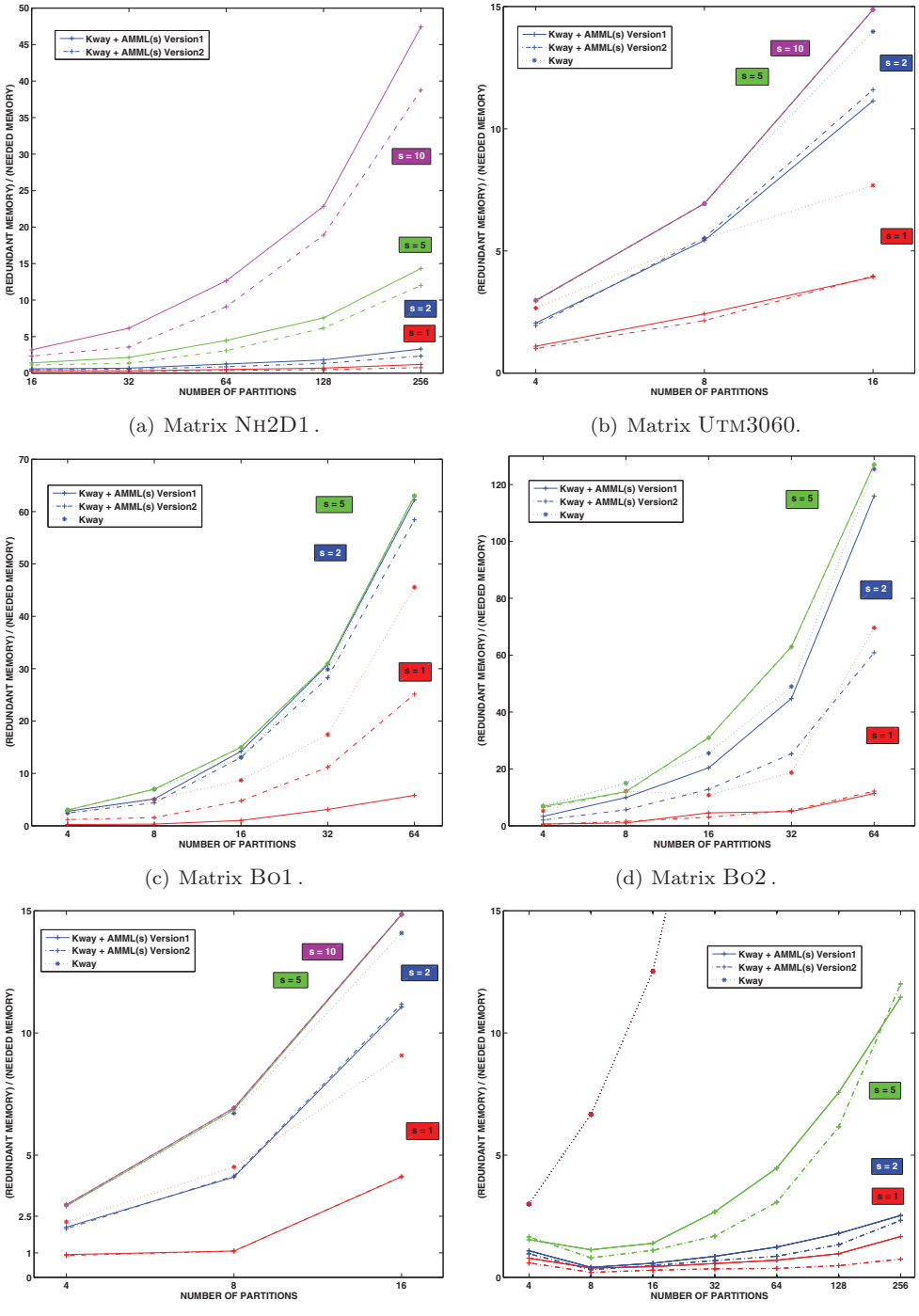
(c) Matrix BO1. (d) Matrix BO2.

(e) Matrix BCSSTK18. (f) Matrix SKY2D.

FIG. 6. *The ratio of redundant flops to needed flops in the ILU*0 *matrix powers kernel as a function of the number of partitions and steps s. The matrices are either reordered using k-way partitioning* (Kway)*, or k-way partitioning followed by* AMML*(s) based on Algorithm* 3 *(Version* 1*) or Algorithm* 4 *(Version* 2*). The number of partitions varies from* 4, 8, 16, 32, 64, 128 *to* 256 *depending on the size of the matrix. The number of steps s is either* 1 *(red),* 2 *(blue),* 5 *(green), or* 10 *(magenta).*

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.



(a) Matrix NH2D1. (b) Matrix UTM3060.

(c) Matrix BO1. (d) Matrix BO2.

(e) Matrix BCSSTK18. (f) Matrix SKY2D.

FIG. 7. *The ratio of redundant data to needed data in the ILU0 matrix powers kernel as a function of the number of partitions and steps s. The matrices are either reordered using k-way partitioning* (Kway)*, or k-way partitioning followed by* AMML(s) *based on Algorithm* 3 *(Version* 1*) or Algorithm* 4 *(Version* 2*). The number of partitions varies from* 4, 8, 16, 32, 64, 128 *to* 256 *depending on the size of the matrix. The number of steps s is either* 1 *(red),* 2 *(blue),* 5 *(green), or* 10 *(magenta).*

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Bcsstk18 (Figures 6(e) and 7(e)), the two versions have almost the same performance.

It is clear that as s or the number of partitions increase, the redundant flops and ghost memory increase. Thus, one has to choose the appropriate number of partitions and steps s with respect to the problem at hand, to obtain the best performance. In other words, one has to find a balance between the redundant flops and communication (number of messages) while taking into consideration the available memory. The choice of the number of partitions P is related to the concept of surface-to-volume ratio discussed in [24] which is an indicator of data dependencies. In other words, the ratio of a subdomain's vertices with edge-cuts to those without edge-cuts should be relatively small. On the other hand, values of s should be chosen so that the processor communicate at most with his neighbors and some factor of his neighbor's neighbors. And the smaller the subdomains are (large P), the smaller s should be and vice versa.

For example, for matrix Nh2D1 of size $40,000 \times 40,000$, which corresponds to a graph of size 200×200, for $p = 256$ with subdomain of size 12×13 the surface-to-volume ratio is around 0.3, which is not very small. Thus for $s = 1$ or $s = 2$ the redundant flops computed are 1 or 2 times the flops needed to perform the multiplication sequentially, which is reasonable (Figure 6(a)). But for $s = 5$ or $s = 10$ the ratio is prohibitive (6 and 18 times). Similarly, for $s = 10$ with $p = 16$ (50×50 subdomains), $p = 32$ (35×36 subdomains), and $p = 64$ (25×25 subdomains) the redundant flops are 1, 2, and 4 times the sequential version, which is reasonable. Note that an increase in the computed redundant flops is equivalent to an increase in the needed memory and the volume of communicated data after computing s basis vectors. Thus, small values of s might be used in practice.

Table 3

Messages and number of words received for performing $s = 1$ multiplication per iteration, on one subdomain α_j of a 2D five-*point stencil matrix, for GMRES and CA-ILU0 preconditioned GMRES.*

	GMRES $y = Ax$	**CA-ILU0 GMRES** $y = (LU)^{-1}Ax$
Each processor j	Receives one message from each of its four neighbors of size $w = (\frac{n}{P})^{\frac{1}{2}}$ words	Receives one message from each of its four neighbors of size $w = (\frac{n}{P})^{\frac{1}{2}}$ words
		Receives one message from four other processors each of size four words

In the case of $s = 1$, the CA-ILU0 preconditioner can be used with the classical preconditioned GMRES where the parallelized multiplication of the form $y_1 = (LU)^{-1}Ay_0$ is replaced by the $s = 1$ version of the ILU0 matrix powers kernel. At the beginning of the first iteration, each processor fetches its corresponding parts of A and y_0 and then factorizes its part of A and computes its part of y_1. Then, before every iteration of GMRES, one communication phase is needed when y_0 is fetched. Table 3 shows the messages and number of words received by processor j on domain α_j of a 2D five-point stencil, for computing $y = Ax$ in GMRES and $y = (LU)^{-1}Ax$ in CA-ILU0 preconditioned GMRES, where the communication pattern in both is similar. We did not compare CA-ILU0 preconditioned GMRES to ILU0 preconditioned GMRES since parallelizing the backward and forward substitution can be implemented by using different approaches. Consider, for example, that the implementation uses nested dissection. For each of the $\log(P)$ levels of nested dissection, there is need for one communication phase between processors, in both forward and backward substitution. Thus, at least $\log(P)$ messages are sent of different sizes. In summary, the communi-

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

cation cost of parallelizing the $y = (LU)^{-1}Ax$ in ILU0 preconditioned GMRES is at least $2\log(p)+4$ messages in $2\log(P)+1$ communication phases, whereas in CA-ILU0 GMRES, it is of the order of eight messages in one communication phase before the computations. Thus the communication is reduced by a factor of $O(2\log(P))$. In general, for $s > 1$, the CA-ILU0 preconditioner reduces communication by at least a factor of $O(2s\log(P) + s)$.

6.3. Comparison between the CA-ILU0 and block Jacobi preconditioners. The block Jacobi preconditioner is one of the simplest parallel preconditioners which avoids communication when performing one multiplication of the form $y = M^{-1}Ax$, where

$$M = \begin{pmatrix} A_{1,1} & 0 & \dots & 0 \\ 0 & A_{2,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & A_{P,P} \end{pmatrix} = \begin{pmatrix} L_{1,1}U_{1,1} & 0 & \dots & 0 \\ 0 & L_{2,2}U_{2,2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & \dots & 0 & L_{P,P}U_{P,P} \end{pmatrix} = LU$$

is constructed from the diagonal blocks of A. The block Jacobi preconditioner starts by partitioning the graph of A into P well-balanced partitions $\pi = \{\Omega_1, \Omega_2, \dots, \Omega_P\}$. Then each processor i is assigned the set of vertices $\alpha_0^{(i)} = V(\Omega_i)$ and has to compute $y(\alpha_0^{(i)})$. Processor i fetches $A(\alpha_0^{(i)}, :)$ and $x(\delta_1^{(i)})$, where $\delta_1^{(i)} = Adj(G(A), \alpha_0^{(i)})$. Then, processor i factorizes $A_{i,i} = A(\alpha_0^{(i)}, \alpha_0^{(i)})$ into $L_{i,i}$ and $U_{i,i}$ matrices by using complete or incomplete LU factorization. Since the diagonal blocks of M are independent, it is possible to perform the LU factorization, and the backward and forward solves in parallel without communication. Thus our CA-ILU0 preconditioner is very similar to block Jacobi in the communication pattern for $s = 1$ only. But for $s > 1$, the block Jacobi preconditioner can't be used with the ILU matrix powers kernel since the reachable sets $\beta_j^{(i)} = R(G(U), \alpha_{j-1}^{(i)})$, $\gamma_j^{(i)} = R(G(L), \beta_j^{(i)})$, and $\delta_j^{(i)} = Adj(G(A), \gamma_j^{(i)})$ can grow in size rapidly where $1 < j \leq s$.

We compare the convergence behavior of the block Jacobi preconditioner, with k-way reordering and LU or ILU0 factorization, to the CA-ILU0 preconditioner, where the input matrix A is reordered using k-way plus AMML(1) reordering. Table 4 shows the ratio of the norm of the error (Err) between the real solution and the approximate solution obtained by the different preconditioned GMRES versions for $tol = 10^{-8}$ ($\|x - x_{app}\|_2/\|x\|_2$), the number of iterations (Iter) needed till convergence, the correctness (LUErr) of the different factorizations ($\|A - LU\|_2/\|A\|_2$), and the introduced fill-in ratio (Fill) of the block Jacobi–LU ($(nnz(L)+nnz(U))/nnz(A)$; nnz is the number of nonzero entries). The input matrix is partitioned into 16, 32, 128, 256, 512, 1024, or 2048 parts (Pa) using k-way.

For all the matrices the CA-ILU0 preconditioner has better convergence than block Jacobi–ILU0. However, the block Jacobi–LU preconditioner has the best convergence when the number of partitions is relatively small, since it is then very similar to a complete LU preconditioner. But when the fill-in ratio decreases as the number of partitions increases, the convergence behavior of CA-ILU0 and block Jacobi–LU preconditioner become very similar. For example, the CA-ILU0 preconditioner converges faster for the BO1 matrix with 32 partitions, the BO2 matrix with 256 partitions, NH2D1 matrix with 1024 partitions, and SKY3D with 512 and 1024 partitions.

7. Conclusion and future work. In this paper, we have introduced CA-ILU0, a communication avoiding ILU0 left-preconditioner. First, we have adapted the matrix powers kernel to the ILU preconditioned system to obtain the ILU matrix powers

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 4
Comparison between the convergence of CA-ILU0 and a block Jacobi preconditioner.

		CA-ILU0			Block Jacobi–ILU0			Block Jacobi–LU			
	Pa	Iter	Err	LUErr	Iter	Err	LUErr	Iter	Err	LUErr	Fill
Bo1	16	51	2E-8	7E-10	62	7E-8	1E-9	49	5E-8	1E-9	6.0
	32	52	4E-8	7E-10	67	8E-8	1E-9	57	5E-8	1E-9	3.6
Bo2	32	144	6E-7	8E-10	151	4E-7	7E-10	110	13E-7	6E-10	15.1
	128	156	4E-7	1E-9	170	5E-7	4E-9	144	4E-7	4E-9	5.7
	256	154	8E-7	1E-9	197	1E-6	4E-9	176	6E-7	4E-9	3.4
Nh2D1	32	173	8E-7	9E-6	193	1E-6	1E-5	116	4E-7	1E-5	14.1
	128	179	1E-6	8E-6	196	1E-6	1E-5	139	6E-7	1E-5	7.0
	512	184	1E-6	1E-5	221	1E-6	1E-5	181	9E-7	1E-5	3.6
	1024	191	1E-6	1E-5	236	2E-6	1E-5	217	1E-6	1E-5	2.5
Nh2D2	32	301	5E-6	2E-6	322	3E-6	2E-6	154	1E-6	2E-6	29.3
	128	308	5E-6	2E-6	339	6E-6	2E-6	201	9E-7	2E-6	13.7
	1024	314	8E-6	2E-6	369	8E-6	2E-6	292	2E-6	2E-6	5.0
	2048	322	4E-6	2E-6	372	5E-6	2E-6	315	3E-6	2E-6	3.6
Sky3D	128	594	9E-5	1E-3	643	1E-4	1E-3	526	1E-4	1E-3	15.5
	256	576	1E-4	1E-3	674	2E-4	1E-3	569	1E-4	1E-3	9.5
	512	563	8E-5	1E-3	723	3E-4	1E-3	627	1E-4	1E-3	5.9
	1024	597	9E-5	1E-3	775	2E-4	1E-3	729	3E-4	1E-3	3.7

kernel. Then we have introduced AMML, a reordering of the matrix A which is applied once the input matrix is partitioned using k-way graph partitioning with edge separators or nested dissection with vertex separators. AMML reorders the matrix A such that s steps of a Krylov subspace solver based on multiplications of the form $y_i = (LU)^{-1}Ay_{i-1}$ can be performed with no communication. We have shown that the reordering does not much affect the convergence of the ILU0 preconditioned GMRES, once the matrix A is reordered using k-way partitioning. Then, we have shown that the complexity of the CA-ILU0(s) reordering is linear with respect to the number of vertices of the largest subdomain. We have also shown that the memory requirements and redundant flops are limited by the same big O function in both CA-GMRES and CA-ILU0 preconditioned GMRES. For all these reasons, we expect that our parallel CA-ILU0 preconditioner will be faster in practice than implementations of ILU0 preconditioners based on other reordering strategies. Yet to obtain good performance, the number of partitions or processors P and the number of steps s have to be chosen carefully to reduce the ghost memory requirements and redundant flops. Hence, there should be a balance between the avoided communication and the introduced redundant flops when choosing s and P.

The AMML(s) reordering allows us to both compute and apply the preconditioner in parallel with no communication, once some ghost data is stored redundantly on each processor. CA-ILU0 can be used with a classic Krylov subspace solver, in which case applying the left preconditioner at each iteration can be done in parallel with no communication. It can also be used with s-step methods, where the ILU0 matrix powers kernel allows the avoidance of communication during s iterations of the Krylov subspace solver. In addition, the CA-ILU0 preconditioner can be used for ILU0 right preconditioned and split preconditioned systems by slightly modifying the ILU matrix powers kernel and the AMML(s) reordering.

We have compared the convergence of CA-ILU0 preconditioned GMRES for $s = 1$ with respect to the block Jacobi preconditioned GMRES. The CA-ILU0 preconditioner has a slightly better convergence than block Jacobi. In practice, block Jacobi

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

might outperform the CA-ILU0 preconditioner, since it is naturally parallelizable without any need for performing redundant computations. However, our CA-ILU0 preconditioner is a parallel version of the ILU0 preconditioner that avoids communication. Thus the output L and U are identical to those obtained from the ILU0 factorization of the AMML(1) reordered matrix A. Hence all the numerical properties of the ILU0 preconditioner, which do not depend on the ordering of the input matrix, are true for the CA-ILU0 preconditioner. Moreover, for $s > 1$ block Jacobi requires communication when used with s-step methods. On the other hand, the CA-ILU0 preconditioner with AMML(s) reordering is one of the very few preconditioners that works with CA-GMRES, and this might render the communication avoiding and s-step methods more usable.

Our future work will focus on implementing the CA-ILU0 preconditioner in a parallel environment to evaluate the improvements with respect to existing implementations of ILU0. It will also focus on extending the method to more general and powerful incomplete LU factorizations such as ILU(k). To obtain a communication avoiding ILU(k) preconditioner (CA-ILUk), we will have to devise a new reordering of the matrix A that will reduce the data dependencies when solving the backward and forward substitution of the L and U matrices obtained from the ILU(k) factorization.

Appendix A. GMRES convergence for different reorderings. The matrices in Table 5 arise from the boundary value problem of the convection-diffusion equations $-\Delta u - 2P\frac{\partial u}{\partial x} + 2P\frac{\partial u}{\partial y} = g$ on $\Omega = (0,1) \times (0,1)$, used in [5, 11] for testing preconditioners, where $P > 0$ and the right-hand side g and the boundary conditions are determined by the solution $u(x,y) = \frac{e^{2P(1-x)}-1}{e^{2P}-1} + \frac{e^{2Py}-1}{e^{2P}-1}$.

TABLE 5
The test matrices.

Matrix	Size	Nonzeros	Symmetric	2D/3D	Problem
CD20P1	3243	22273	No	2D	Convection diffusion P1 FE
CD50P1	3638	25008	No	2D	Convection diffusion P1 FE
CD100P1	3946	27142	No	2D	Convection diffusion P1 FE
CD500P1	5993	41225	No	2D	Convection diffusion P1 FE
CD20P2	12905	146807	No	2D	Convection diffusion P2 FE
CD100P2	14606	166226	No	2D	Convection diffusion P2 FE
CD400P2	17599	200293	No	2D	Convection diffusion P2 FE

The matrices were generated by Pierre-Henri Tournier using FreeFem++ [22] with finite element P1 and P2 schemes with an adaptive mesh for $P = 20, 50, 100, 400, 500$. Note that the system is scaled.

In Table 6, we show the convergence (Iter) of the ILU0 preconditioned GMRES, where the matrices introduced in Table 5 are reordered differently for a different number of partitions (Pa). Note that NO, RCM, ND, AMML(s) V1, AMML(s) V2 denote natural ordering, reverse Cuthill–McKee reordering, nested dissection, k-way+AMML(s) reordering (Algorithm 3), and k-way+AMML(s) reordering (Algorithm 4), respectively.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

TABLE 6

GMRES convergence for different reorderings with respect to number of partitions for the initial guess $y_0 = 0$ and $tol = 10^{-8}$.

		NO		RCM		ND		Kway		AMML(1)V1		AMML(5)V1		AMML(1)V2		AMML(5)V2	
	Pa	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err
Watt2	2	50	3E-8	47	3E-8	64	2E-7	50	4E-8	51	3E-8	52	1E-7	50	3E-8	54	1E-7
	4					64	7E-8	50	3E-8	52	3E-8	53	3E-8	53	5E-8	58	6E-8
	8					65	4E-8	50	2E-8	51	8E-8	54	1E-7	56	2E-8	57	8E-8
	16					65	1E-7	51	3E-8	54	3E-8	54	1E-8	57	8E-8	58	8E-8
	32					65	1E-7	51	3E-8	52	1E-7	52	1E-7	56	1E-7	56	1E-7
	64					63	5E-8	54	1E-7	53	9E-8	52	3E-7	57	1E-7	57	1E-7
CD20P1	2	59	2E-8	62	4E-8	94	1E-7	59	3E-8	60	4E-8	83	4E-8	60	4E-8	83	8E-8
	4					95	9E-8	60	2E-8	65	4E-8	85	1E-7	67	3E-8	89	1E-7
	8					91	1E-7	59	3E-8	72	9E-8	97	1E-7	73	8E-8	98	1E-7
	16					94	1E-7	64	7E-8	76	9E-8	92	1E-7	76	4E-8	92	9E-8
	32					91	1E-7	63	6E-8	83	6E-8	89	1E-7	85	5E-8	93	1E-7
	64					89	5E-8	63	4E-8	87	5E-8	90	5E-8	90	5E-8	92	5E-8
CD50P1	2	64	4E-8	66	1E-7	96	1E-7	64	2E-8	66	5E-8	79	1E-7	67	3E-8	79	7E-8
	4					98	7E-8	64	5E-8	69	5E-8	79	1E-7	71	4E-8	83	1E-7
	8					96	1E-7	64	3E-8	71	7E-8	94	8E-8	77	1E-7	98	1E-7
	16					94	9E-8	71	2E-8	81	5E-8	98	1E-7	83	8E-8	97	7E-8
	32					96	1E-7	72	4E-8	87	7E-8	98	1E-7	92	5E-8	94	1E-7
	64					93	7E-8	71	5E-8	87	1E-7	95	1E-7	94	1E-7	95	1E-7
CD100P1	2	69	8E-8	71	3E-8	102	1E-7	70	1E-7	68	8E-8	78	1E-7	69	7E-8	80	1E-7
	4					97	2E-7	71	8E-8	75	1E-7	96	1E-7	74	1E-7	96	1E-7
	8					100	1E-7	71	8E-8	81	1E-7	107	2E-7	82	1E-7	101	1E-7
	16					105	1E-7	77	7E-8	89	6E-8	110	1E-7	89	8E-8	105	1E-7
	32					97	1E-7	73	4E-8	92	5E-8	106	1E-7	94	9E-8	105	1E-7
	64					96	7E-8	76	6E-8	96	7E-8	98	1E-7	97	7E-8	103	1E-7
CD500P1	2	92	1E-7	118	1E-7	186	4E-7	93	1E-7	94	1E-7	98	1E-7	94	1E-7	99	1E-7
	4					176	7E-7	93	1E-7	98	1E-7	113	2E-7	98	2E-7	119	2E-7
	8					174	7E-7	93	1E-7	102	1E-7	117	1E-7	105	1E-7	115	2E-7
	16					184	2E-7	94	1E-7	107	1E-7	167	1E-7	108	1E-7	180	1E-7
	32					190	6E-7	99	1E-7	139	1E-7	188	1E-7	144	1E-7	190	1E-7
	64					198	1E-7	102	1E-7	167	2E-7	177	1E-7	161	1E-7	178	1E-7
CD20P2	2	146	1E-7	155	2E-7	162	3E-7	147	2E-7	151	2E-7	158	2E-7	151	2E-7	154	3E-7
	4					163	3E-7	143	2E-7	147	2E-7	156	2E-7	147	2E-7	153	3E-7
	8					160	3E-7	142	2E-7	148	2E-7	167	3E-7	148	2E-7	161	2E-7
	16					158	3E-7	150	2E-7	158	2E-7	170	3E-7	154	2E-7	151	2E-7
	32					161	3E-7	150	3E-7	156	2E-7	164	3E-7	152	2E-7	154	3E-7
	64					162	3E-7	149	2E-7	163	2E-7	169	2E-7	156	3E-7	159	3E-7
CD100P2	2	157	2E-7	170	3E-7	177	2E-7	158	2E-7	159	2E-7	159	2E-7	159	3E-7	160	3E-7
	4					177	2E-7	152	2E-7	155	1E-7	155	1E-7	159	3E-7	154	3E-7
	8					183	2E-7	156	2E-7	164	4E-7	178	2E-7	163	2E-7	162	3E-7
	16					175	4E-7	153	3E-7	165	3E-7	171	2E-7	163	3E-7	163	3E-7
	32					167	3E-7	155	3E-7	164	2E-7	182	2E-7	158	3E-7	162	2E-7
	64					178	3E-7	162	3E-7	176	3E-7	184	3E-7	176	3E-7	169	2E-7
CD400P2	2	278	4E-7	244	2E-7	369	2E-7	277	4E-7	278	4E-7	276	4E-7	278	4E-7	280	4E-7
	4					339	3E-7	278	5E-7	279	5E-7	302	3E-7	280	4E-7	282	5E-7
	8					354	3E-7	275	4E-7	300	5E-7	305	2E-7	301	5E-7	304	5E-7
	16					391	2E-7	280	4E-7	308	4E-7	313	2E-7	313	3E-7	293	2E-7
	32					353	2E-7	287	4E-7	317	4E-7	327	3E-7	312	5E-7	363	4E-7
	64					335	3E-7	279	2E-7	316	3E-7	314	5E-7	289	4E-7	330	3E-7

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

REFERENCES

[1] Y. Achdou and F. Nataf, *Low frequency tangential filtering decomposition*, Numer. Linear Algebra Appl., 14 (2007), pp. 129–147.

[2] C. Aykanat, B. B. Cambazoglu, and B. Uçar, *Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices*, J. Parallel Distrib. Comput., 68 (2008), pp. 609–625.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Minimizing communication in numerical linear algebra*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 866–901.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[5] 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.

[6] E. Carson, N. Knight, and J. Demmel, *Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods*, SIAM J. Sci. Comput., 35 (2013), pp. S42–S61.

[7] E. Carson, N. Knight, and J. Demmel, *Hypergraph partitioning for computing matrix powers*, extended abstract, in 5th SIAM Workshop on Combinatorial Scientific Computing, Darmstadt, Germany, 2011, pp. 31–33.

[8] U. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Systems, 10 (1999), pp. 673–693.

[9] A. T. Chronopoulos and W. Gear, *s-step iterative methods for symmetric linear systems*, J. Comput. Appl. Math., 25 (1989), pp. 153–168.

[10] T. A. Davis, *The University of Florida Sparse Matrix Collection*, 1994, available online at http://www.cise.ufl.edu/research/sparse/matrices/.

[11] H. C. Elman, *Relaxed and stabilized incomplete factorizations for non-self-adjoint linear systems*, BIT, 29 (1989), pp. 890–915.

[12] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239.

[13] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Avoiding communication in sparse matrix computations*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), IEEE, Piscataway, NJ, 2008, pp. 1–12.

[14] S. Doi and T. Washio, *Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations*, Parallel Comput., 25 (1999), pp. 1995–2014.

[15] J. Erhel, *A parallel GMRES version for general sparse matrices*, Electron. Trans. Numer. Anal., 3 (1995), pp. 160–176.

[16] M. Garey, D. Johnson, and L. Stockmeyer, *Some simplified NP-complete graph problems*, Theoret. Comput. Sci., 1 (1976), pp. 237–267.

[17] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[18] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose, *Hiding global communication latency in the GMRES algorithm on massively parallel machines*, SIAM J. Sci. Comput., 35 (2013), pp. C48–C71.

[19] J. R. Gilbert and T. Peierls, *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 862–874.

[20] L. Grigori, J. W. Demmel, and H. Xiang, *CALU: A communication optimal LU factorization algorithm*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1317–1350.

[21] L. Grigori and S. Moufawad, *Communication Avoiding ILU*0 *Preconditioner*, Technical Report, ALPINES - INRIA Paris-Rocquencourt, Paris, France, 2013.

[22] F. Hecht, *New development in freefem++*, J. Numer. Math., 20 (2012), pp. 251–265.

[23] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Natl. Bur. Standards, 49 (1952), pp. 409–436.

[24] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, EECS Department, University of California, Berkeley, CA, 2010.

[25] G. Karypis and V. Kumar, *Metis* 4.0: *Unstructured Graph Partitioning and Sparse Matrix Ordering System*, technical report, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1998.

[26] B. W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Syst. Tech. J., 49 (1970), pp. 291–307.

[27] M. S. Khaira, G. L. Miller, and T. J. Sheffler, *Nested Dissection: A Survey and Comparison of Various Nested Dissection Algorithms*, Technical report, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1992.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[28] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comput., 31 (1977), pp. 148–162.
[29] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, *Minimizing communication in sparse matrix solvers*, in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 09, ACM, New York, 2009, pp. 1–12.
[30] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[31] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.
[32] J. Scott and M. Tuma, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. Sci. Comput., 36 (2014), pp. A609–A633.
[33] H. F. Walker, *Implementation of the GMRES method using Householder transformations*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 152–163.

Downloaded 05/04/15 to 61.129.42.15. Redistribution subject to SIAM license or copyright; see http://www.siam.org/journals/ojsa.php

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.