SIAM J. SCI. COMPUT.
Vol. 40, No. 6, pp. C827–C850

A NOVEL PARTITIONING METHOD FOR ACCELERATING THE BLOCK CIMMINO ALGORITHM*

F. SUKRU TORUN†, MURAT MANGUOGLU‡, AND CEVDET AYKANAT§

Abstract. We propose a novel block-row partitioning method in order to improve the convergence rate of the block Cimmino algorithm for solving general sparse linear systems of equations. The convergence rate of the block Cimmino algorithm depends on the orthogonality among the block rows obtained by the partitioning method. The proposed method takes numerical orthogonality among block rows into account by proposing a row inner-product graph model of the coefficient matrix. In the graph partitioning formulation defined on this graph model, the partitioning objective of minimizing the cutsize directly corresponds to minimizing the sum of interblock inner products between block rows thus leading to an improvement in the eigenvalue spectrum of the iteration matrix. This in turn leads to a significant reduction in the number of iterations required for convergence. Extensive experiments conducted on a large set of matrices confirm the validity of the proposed method against a state-of-the-art method.

Key words. row projection methods, block Cimmino algorithm, Krylov subspace methods, row inner-product graph, graph partitioning

AMS subject classifications. 65F10, 65F50, 05C50, 05C70

DOI. 10.1137/18M1166407

1. Introduction.

Row projection methods are a class of iterative linear system solvers [13, 14, 36] that are used for solving a linear system of equations of the form

$$Ax = f, \tag{1.1}$$

where A is an $n \times n$ sparse nonsymmetric nonsingular matrix, x and f are column vectors of size n. In these methods, the solution is computed through successive projections onto rows of A. There are mainly two major variations known as Kacmarz [35] and Cimmino [18]. Kaczmarz obtains the solution through a product of orthogonal projections whereas Cimmino reaches the solution through a sum of orthogonal projections. Cimmino is known to be more amenable to parallelism than Kaczmarz [14]. However, Kaczmarz can be still parallelized via block Kaczmarz [36], CARP [32], or multicoloring [27]. The required number of iterations for the Cimmino algorithm, however, could be quite large. One alternative variation is the block Cimmino [7] which is a block-row projection method. Iterative block Cimmino has been studied extensively in [7, 9, 14, 17, 23, 25, 26]. A pseudodirect version of block Cimmino based on the augmented block-row projections is proposed in [24]. However as in any

*Submitted to the journal's Software and High-Performance Computing section January 22, 2018; accepted for publication (in revised form) September 24, 2018; published electronically December 13, 2018.

http://www.siam.org/journals/sisc/40-6/M116640.html

Funding: The work of the first author was supported by the Scientific and Technological Research Council of Turkey (TUBITAK), under the program BIDEB-2211. The second author's work was supported by the Alexander von Humboldt Foundation for support of a research stay at TU-Berlin.

†Department of Computer Engineering, Ankara Yıldırım Beyazıt University, 06010, Ankara, Turkey (ftorun@ybu.edu.tr).

‡Institut für Mathematik, Technische Universität Berlin, 10623 Berlin, Germany and Department of Computer Engineering, Middle East Technical University, 06800 Ankara, Turkey (manguoglu@ceng.metu.edu.tr).

§Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey (aykanat@cs.bilkent.edu.tr).

other direct solver [3, 12, 40, 44], this approach also suffers from an extensive memory requirement due to fill-in.

In the block Cimmino scheme, the linear system (1.1) is partitioned into K block rows, where $K \leq n$, as follows:

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_K \end{pmatrix} x = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_K \end{pmatrix}. \tag{1.2}$$

In (1.2), the coefficient matrix and right-hand side vector are partitioned conformably. Here A_i is a submatrix of size $m_i \times n$ and f_i is a subvector of size m_i for $i = 1, 2, \ldots, K$, where

$$n = \sum_{i=1}^{K} m_i. \tag{1.3}$$

The block Cimmino scheme is given in Algorithm 1.1, where A_i^+ is the Moore–Penrose pseudoinverse of A_i and it is defined as

$$A_i^+ = A_i^T (A_i A_i^T)^{-1}. \tag{1.4}$$

In Algorithm 1.1, A_i^+ is used for the sake of the clarity of the notation and it is never computed explicitly. In fact, at line 4 of Algorithm 1.1, the minimum norm solution of an underdetermined linear least squares problem is obtained via the augmented system approach as discussed later. In Algorithm 1.1, δ_i vectors, which are of size n, can be computed independently in parallel without any communication, hence, the block Cimmino algorithm is quite suitable for parallel computing platforms. At line 6, the solution is updated by the sum of projections which is scaled by the relaxation parameter (ω). In the parallel Cimmino algorithm, communication is required only for summing up the δ_is.

Algorithm 1.1 Block Cimmino method.

1: Choose $x^{(0)}$
2: **while** $t = 0, 1, 2, \ldots,$ until convergence **do**
3: **for** $i = 1, \ldots, K$ **do**
4: $\delta_i = A_i^+ (f_i - A_i x^{(t)})$
5: **end for**
6: $x^{(t+1)} = x^{(t)} + \omega \sum_{i=1}^{K} \delta_i$
7: **end while**

An iteration of the block Cimmino method can be reformulated as follows:

$$\begin{aligned} x^{(t+1)} &= x^{(t)} + \omega \sum_{i=1}^{K} A_i^+ \left(f_i - A_i x^{(t)} \right) \\ &= \left(I - \omega \sum_{i=1}^{K} A_i^+ A_i \right) x^{(t)} + \omega \sum_{i=1}^{K} A_i^+ f_i \\ &= (I - \omega H) x^{(t)} + \omega \sum_{i=1}^{K} A_i^+ f_i \\ &= Q x^{(t)} + \omega \sum_{i=1}^{K} A_i^+ f_i, \end{aligned} \tag{1.5}$$

where Q is the iteration matrix for the block Cimmino algorithm. $\omega H = I - Q$ is the sum of $\mathcal{P}_{\mathcal{R}}(A_i^T)$s (projections onto A_i^T) and it is defined by

$$\omega H = \omega \sum_{i=1}^{K} \mathcal{P}_{\mathcal{R}}(A_i^T) = \omega \sum_{i=1}^{K} A_i^+ A_i. \tag{1.6}$$

The projections in block Cimmino iterations can be calculated using several approaches, such as normal equations [31], seminormal equations [30, 31], QR factorization [31], and augmented system [6, 31]. The normal and seminormal equation approaches are not considered since they have the potential of introducing numerical difficulties that can be disastrous [21, 30] in some cases when the problem is ill-conditioned. Although the QR factorization is numerically more stable, it is computationally expensive. Therefore we use the augmented system approach, which requires the solution of sparse linear systems that can be done effectively by using a sparse direct solver [7]. Note that if submatrix A_i is in a column overlapped block diagonal form, one could also use the algorithm in [45]. However, this approach is not considered since we assume no structure for the coefficient matrix.

In the augmented system approach, we obtain the solution of (1.7) by solving the linear system (1.8),

$$A_i \delta_i = r_i \qquad (r_i = f_i - A_i x^{(t)}), \tag{1.7}$$

$$\begin{pmatrix} I & A_i^T \\ A_i & 0 \end{pmatrix} \begin{pmatrix} \delta_i \\ \varsigma_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix}. \tag{1.8}$$

Hence, the solution of the augmented system gives δ_i.

1.1. The conjugate gradient acceleration of the block Cimmino method. The convergence rate of the block Cimmino algorithm is known to be slow [14]. In [14], the conjugate gradient (CG) method is proposed to accelerate the row projection method. It is also reported that the CG accelerated Cimmino method competes favorably compared to classical preconditioned generalized minimum residual and CG on normal equations for the solution of nonsymmetric linear systems that arise in an elliptic partial differential equation. On the other hand, it should be noted that the main motivation of the block Cimmino algorithm is its amenability to parallelism [23].

The iteration scheme of the block Cimmino (1.5) gives

$$x^{(t+1)} = (I - \omega H) x^{(t)} + \omega \sum_{i=1}^{K} A_i^+ f_i, \tag{1.9}$$

where the H matrix is symmetric and positive definite according to (1.6) if A is square and full rank. Hence, one can solve the following system using CG,

$$\omega H x = \omega \xi, \tag{1.10}$$

where $\xi = \sum_{i=1}^{K} A_i^+ f_i$ and x is the solution vector of the system (1.1). Note that since ω appears on both sides of (1.10), it does not affect the convergence of CG. Algorithm 1.2 is the pseudocode for the CG accelerated block Cimmino method [47], which is in fact the classical CG applied to the system given in (1.10). In the second line of the algorithm, the initial residual is computed in the same way as the first iteration of Algorithm 1.1. The matrix vector multiplications in the CG are expressed

as the solution of K independent underdetermined systems at line 5 which can be done in parallel and need to be summed to obtain $\psi^{(t)}$ by using an all-reduce operation.

Algorithm 1.2 CG acceleration of block Cimmino method.

1: Choose $x^{(0)}$
2: $r^{(0)} = \xi - \sum_{i=1}^{K} A_i^+ A_i \, x^{(0)}$
3: $p^{(0)} = r^{(0)}$
4: **while** $t = 0, 1, 2, \ldots,$ until convergence **do**
5: $\quad \psi^{(t)} = \sum_{i=1}^{K} A_i^+ A_i \, p^{(t)}$
6: $\quad \alpha^{(t)} = ({r^{(t)}}^T r^{(t)})/({p^{(t)}}^T \psi^{(t)})$
7: $\quad x^{(t+1)} = x^{(t)} + \alpha^{(t)} p^{(t)}$
8: $\quad r^{(t+1)} = r^{(t)} - \alpha^{(t)} \psi^{(t)}$
9: $\quad \beta^{(t)} = ({r^{(t+1)}}^T r^{(t+1)})/({r^{(t)}}^T r^{(t)})$
10: $\quad p^{(t+1)} = r^{(t+1)} + \beta^{(t)} p^{(t)}$
11: **end while**

1.2. The effect of partitioning. The convergence rate of the CG accelerated block Cimmino algorithm is essentially the convergence rate of CG applied to (1.10). A well-known upper bound on the convergence rate can be given in terms of the extreme eigenvalues ($\lambda_{\min}$ and $\lambda_{\max}$) of the coefficient matrix. Let

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{1.11}$$

be the 2-norm condition number of H. Then, as in [31], an upper bound on the convergence rate of the CG accelerated block Cimmino can be given by

$$\frac{\|x^{(t)} - x^*\|_H}{\|x^{(0)} - x^*\|_H} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^t, \tag{1.12}$$

where x^* is the exact solution and $\|y\|_H = y^T H y$. Furthermore, it was shown that the convergence rate of CG not only depends on the extreme eigenvalues but also on the separation between those extreme eigenvalues and interior eigenvalues [46] as well the clustering of the internal eigenvalues [34]. In summary, the convergence rate of the CG accelerated block Cimmino depends on the extreme eigenvalues and the number of clusters as well as the quality of the clustering.

Therefore, the partitioning of the coefficient matrix A into block rows is crucial for improving the convergence rate of the CG accelerated block Cimmino algorithm, since it can improve the eigenvalue distribution of H. Note that the eigenvalues of H are only affected by the block-row partitioning of the coefficient matrix A and are independent of any column ordering [23].

Let the QR factorization of A_i^T be defined as

$$Q_i R_i = A_i^T, \tag{1.13}$$

where the matrices Q_i and R_i have dimensions of $n \times m_i$ and $m_i \times m_i$, respectively.

Then, the H matrix can be written as follows [7];

$$\begin{aligned} H &= \sum_{i=1}^{K} A_i^T (A_i A_i^T)^{-1} A_i \\ &= \sum_{i=1}^{K} Q_i Q_i^T \\ &= (Q_1, \ldots, Q_K)(Q_1, \ldots, Q_K)^T. \end{aligned} \tag{1.14}$$

Since the eigenvalue spectrum of matrix $(Q_1, \ldots, Q_K)(Q_1, \ldots, Q_K)^T$ is the same as the eigenvalue spectrum of matrix $(Q_1, \ldots, Q_K)^T(Q_1, \ldots, Q_K)$ [29], H is similar to

$$\begin{pmatrix} I_{m_1 \times m_1} & {Q_1}^T Q_2 & \ldots & {Q_1}^T Q_K \\ {Q_2}^T Q_1 & I_{m_2 \times m_2} & \ldots & {Q_2}^T Q_K \\ \vdots & \ldots & \ddots & \vdots \\ {Q_K}^T Q_1 & {Q_K}^T Q_2 & \ldots & I_{m_K \times m_K} \end{pmatrix}, \tag{1.15}$$

where the singular values of matrix ${Q_i}^T Q_j$ represent the principal angles between the subspaces spanned by the rows of A_i and A_j [11]. Hence, the smaller the off-diagonals of the matrix (1.15), the more eigenvalues of H will be clustered around one by the Gershgorin theorem [28]. Therefore, the convergence rate of the block Cimmino method highly depends on the orthogonality among A_i blocks. If A_i blocks are more orthogonal to each other, row inner products between blocks would be small and hence the eigenvalues will be clustered around one. Consequently, CG is expected to converge in a fewer number of iterations.

In the literature, Cuthill–Mckee- (CM-) [19] based partitioning strategies [23, 43, 47] are utilized to define block rows using CM level set information on the normal equations of the original matrix. These strategies benefit from the level sets of CM for creating the desired number of block rows. In CM, nodes in a level set have the same distance from the starting node and these nodes have neighbors only in the previous and the next level sets. Therefore a permuted matrix based on the ordering of the level sets constitutes a block tridiagonal structure. These strategies may suffer from not reaching the desired number of block rows due to a smaller number of level sets. They also suffer from obtaining quite unbalanced partitions due to relatively larger sizes of some level sets. Numerical values are only considered when a dropping-based filtering strategy is used. Although filtering small elements on normal equations before applying CM allows more freedom in partitioning by increasing the number of level sets, it does not, however, hold the properties of the strict two-block partitioning [9, 43, 47]. In addition, it is difficult to determine the best filtering threshold value a priori and find a common threshold which is good for all matrices.

In recent studies [23, 47], a hypergraph partitioning method is used to find a good block-row partitioning for the CG accelerated block Cimmino algorithm. It is reported that it performs better than CM-based methods (with or without filtering small elements). In hypergraph partitioning, the partitioning objective of minimizing the cutsize corresponds to minimizing the number of linking columns among row

blocks, where a linking column refers to a column that contains nonzeros in more than one block row. This in turn loosely relates to increasing the structural orthogonality [39] among row blocks. Here, two rows are considered to be structurally more orthogonal if they have fewer nonzeros in the same columns. This measure depends on only nonzero counts and ignores the numerical values of nonzeros.

In this work, we propose a novel graph theoretical block-row partitioning method for the CG accelerated block Cimmino algorithm. For this purpose, we introduce a row inner-product graph model of a given matrix A and then the problem of finding a good block-row partitioning is formulated as a graph partitioning problem on this graph model. The proposed method takes the numerical orthogonality between block rows of A into account. In the proposed method, the partitioning objective of minimizing the cutsize directly corresponds to minimizing the sum of interblock inner products between block rows thus leading to an improvement in the eigenvalue spectrum of H.

The validity of the proposed method is confirmed against two baseline methods by conducting experiments on a large set of matrices that arise in a variety of real life applications. One of the two baseline methods is the state-of-the-art hypergraph partitioning method introduced in [23]. We conduct experiments to study the effect of the partitioning on the eigenvalue spectrum of H. We also conduct experiments to compare the performance of three methods in terms of the number of CG iterations and parallel CG time to solution. The results of these experiments show that the proposed partitioning method is significantly better than both baseline methods in terms of all of these performance metrics. Finally, we compare the preprocessing overheads of the methods and show that the proposed method incurs much less overhead than the hypergraph partitioning method, thus allowing a better amortization.

The rest of the paper is organized as follows. Section 2 presents the proposed partitioning method and its implementation. In section 3, we present and discuss the experimental results. Finally, the paper concludes with conclusions and directions for future research in section 4.

2. The proposed partitioning method. In this section, we first describe the row inner-product graph model and then show that finding a good block-row partitioning can be formulated as a graph partitioning problem on this graph model. Finally, we give the implementation details for the construction and partitioning of the graph model. We refer the reader to Appendix A for a short background on graph partitioning.

2.1. Row inner-product graph model. In the row inner-product graph $\mathcal{G}_{\mathrm{RIP}}(A) = (\mathcal{V}, \mathcal{E})$ of matrix A, vertices represent the rows of matrix A and edges represent nonzero inner products between rows. That is, $\mathcal{V}$ contains vertex v_i for each row r_i of matrix A. $\mathcal{E}$ contains an edge (v_i, v_j) only if the inner product of rows r_i and r_j is nonzero. That is,

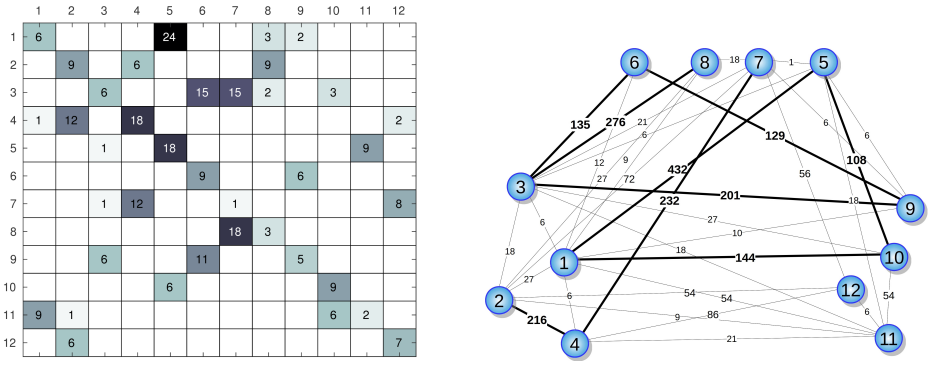
$$\mathcal{E} = \{(v_i, v_j) \mid \mathrm{r}_i \mathrm{r}_j^T \neq 0\}. \tag{2.1}$$

Each vertex v_i can be associated with a unit weight or a weight that is equal to the number of nonzeros in row r_i, that is,

$$w(v_i) = 1 \ \text{ or } \ w(v_i) = nnz(\mathrm{r}_i), \tag{2.2}$$

respectively. Each edge (v_i, v_j) is associated with a cost equal to the absolute value of the respective inner product. That is,

$$cost(v_i, v_j) = \left|\mathrm{r}_i \mathrm{r}_j^T\right| \ \text{ for all } \ (v_i, v_j) \in \mathcal{E}. \tag{2.3}$$



(a) Sample matrix A (b) Row inner-product graph $\mathcal{G}_{\text{RIP}}$ of A

FIG. 1. *Row inner-product graph model.*

If we prescale the rows of coefficient matrix A such that each row has a unit 2-norm, then the cost of edge (v_i, v_j) will correspond to the cosine of the angle between the pair of rows r_i and r_j. Therefore we prescale the matrix and the right-hand side vector in order to improve the effectiveness of the proposed graph model. We note that the convergence of the block Cimmino algorithm is independent of row scaling [25, 43, 47].

This graph is topologically equivalent to the standard graph representation of the symmetric matrix resulting from the sparse matrix-matrix multiplication operation $C = AA^T$. That is, the sparsity pattern of C corresponds to the adjacency matrix representation of $\mathcal{G}_{\text{RIP}}$. Each nonzero c_{ij} of the resulting matrix C incurs an edge (v_i, v_j). Since each nonzero entry c_{ij} of C is computed as the inner product of row r_i and row r_j, the absolute value of the nonzero c_{ij} determines the cost of the respective edge (v_i, v_j). That is, since $c_{ij} = \mathrm{r}_i \mathrm{r}_j^T$, we have $cost(v_i, v_j) = |c_{ij}|$.

Figure 1(a) shows a 12×12 sample sparse matrix that contains 38 nonzeros. Note that for the sake of clarity of presentation, rows of the sample matrix A are not prescaled. Figure 1(b) depicts the proposed row inner-product graph $\mathcal{G}_{\text{RIP}}$ for this sample matrix. As seen in Figure 1(b), $\mathcal{G}_{\text{RIP}}$ contains 12 vertices each of which corresponds to a row and 35 edges each of which corresponds to a nonzero row inner product. For example, the inner product of rows r_2 and r_4 is nonzero, where $\mathrm{r}_2 \mathrm{r}_4^T = (9\times12) + (6\times18) = 216$ so that $\mathcal{G}_{\text{RIP}}$ contains the edge (v_2, v_4) with $cost(v_2, v_4) = 216$. In Figure 1(b), edges with cost larger than 100 are shown with thick lines in order to make such high inner-product values more visible.

Figure 4(a) shows the resulting matrix C of the sparse matrix-matrix multiplication $C = AA^T$. Only the off-diagonal nonzero entries together with their values are shown since the values of the diagonal entries do not affect $\mathcal{G}_{\text{RIP}}$. The cells that contain nonzeros larger than 100 are shown with black background in order to make such high values more visible. Comparison of Figures 1(b) and 4(a) shows that the topology of the standard graph model $\mathcal{G}(C)$ of matrix $C = AA^T$ is equivalent to the topology of $\mathcal{G}_{\text{RIP}}$. As also seen in Figure 4(a), the values of the nonzero entries of matrix C are equal to the costs of respective edges of $\mathcal{G}_{\text{RIP}}$. For example, nonzero c_{24} with a value 216 incurs an edge (v_2, v_4) with $cost(v_2, v_4) = 216$.

2.2. Block-row partitioning via partitioning $\mathcal{G}_{\text{RIP}}$. A K-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{G}_{\text{RIP}}$ can be decoded as a partial permutation on the rows of A

to induce a permuted matrix A^{Π}, where

$$A^{\Pi} = PA = \begin{bmatrix} A_1^{\Pi} \\ \vdots \\ A_k^{\Pi} \\ \vdots \\ A_K^{\Pi} \end{bmatrix} = \begin{bmatrix} \mathcal{R}_1 \\ \vdots \\ \mathcal{R}_k \\ \vdots \\ \mathcal{R}_K \end{bmatrix}. \tag{2.4}$$

Here, P denotes the row permutation matrix which is defined by the K-way partition Π as follows: the rows associated with the vertices in $\mathcal{V}_{k+1}$ are ordered after the rows associated with the vertices in $\mathcal{V}_k$ for $k = 1, 2, \ldots, K-1$. That is, the block row $\mathcal{R}_k$ contains the set of rows corresponding to the set of vertices in part $\mathcal{V}_k$ of partition Π, where ordering of the rows within block row $\mathcal{R}_k$ is arbitrary for each $k = 1, 2, \ldots, K$. Note that we use the notation $\mathcal{R}_k$ to denote both kth block row A_k^{Π} and the set of rows in A_k^{Π}. Since the column permutation does not affect the convergence of the block Cimmino algorithm [23], the original column ordering of A is maintained.

Consider a partition Π of $\mathcal{G}_{\mathrm{RIP}}$. The weight W_k of a part $\mathcal{V}_k$ is either equal to the number of rows or number of nonzeros in block row $\mathcal{R}_k$ depending on the vertex weighting scheme used according to (2.2). That is,

$$W_k = |\mathcal{R}_k| \;\text{ or }\; W_k = \sum_{r_i \in \mathcal{R}_k} nnz(\mathrm{r}_i). \tag{2.5}$$

Therefore in partitioning $\mathcal{G}_{\mathrm{RIP}}$, the partitioning constraint of maintaining balance among part weights according to (A.2) corresponds to maintaining balance on either the number of rows or the number of nonzeros among the block rows.

Consider a partition Π of $\mathcal{G}_{\mathrm{RIP}}$. A cut edge (v_i, v_j) between parts $\mathcal{V}_k$ and $\mathcal{V}_m$ represents a nonzero interblock inner product $\mathrm{r}_i \mathrm{r}_j^T$ between block rows $\mathcal{R}_k$ and $\mathcal{R}_m$. Therefore the cutsize of Π (given in (A.3)) is equal to

$$\begin{aligned} \mathrm{cutsize}(\Pi) \triangleq \sum_{(v_i,v_j)\in\mathcal{E}_{cut}} cost(v_i, v_j) &= \sum_{1\le k<m\le K} \sum_{\substack{v_i\in\mathcal{V}_k \\ v_j\in\mathcal{V}_m}} cost(v_i, v_j) \\ &= \sum_{1\le k<m\le K} \sum_{\substack{\mathrm{r}_i\in\mathcal{R}_k \\ \mathrm{r}_j\in\mathcal{R}_m}} \left|\mathrm{r}_i \mathrm{r}_j^T\right|, \end{aligned} \tag{2.6}$$

which corresponds to the total sum of the interblock inner products (interIP(Π)). So, in partitioning $\mathcal{G}_{\mathrm{RIP}}$, the partitioning objective of minimizing the cutsize corresponds to minimizing the sum of interblock inner products between block rows. Therefore this partitioning objective corresponds to making the block rows numerically more orthogonal to each other. This way, we expect this method to yield a faster convergence in the CG accelerated block Cimmino algorithm.

We introduce Figures 2 and 3 in order to clarify the proposed graph partitioning method for block-row partitioning. Figure 2(a) shows a straightforward 3-way block-row partition $\{\mathcal{R}_1^s, \mathcal{R}_2^s, R_3^s\}$ of the sample matrix A given in Figure 1(a), where the first four, the second four, and the third four consecutive rows in the original order constitute the block rows $\mathcal{R}_1^s, \mathcal{R}_2^s$, and $\mathcal{R}_3^s$, respectively. Figure 2(b) shows the 3-way vertex partition $\Pi^s(\mathcal{V}) = \{\mathcal{V}_1^s, \mathcal{V}_2^s, \mathcal{V}_3^s\}$ of $\mathcal{G}_{\mathrm{RIP}}$ that corresponds to this straightforward 3-way block-row partition. Figure 3(a) shows a good 3-way vertex partition $\Pi^g(\mathcal{V}) = \{\mathcal{V}_1^g, \mathcal{V}_2^g, \mathcal{V}_3^g\}$ of $\mathcal{G}_{\mathrm{RIP}}$ obtained by using the graph partitioning tool

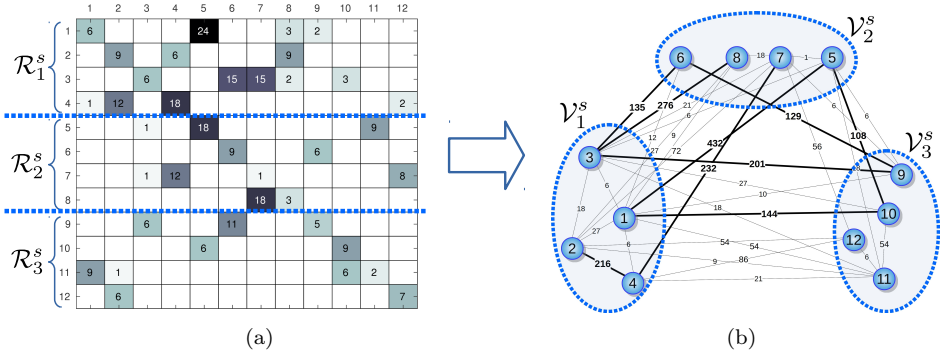


FIG. 2. (a) *Straightforward* 3*-way row partition of* A *and* (b) 3*-way partition* $\Pi^s(\mathcal{V})$ *of* $\mathcal{G}_{\text{RIP}}(A)$ *induced by Figure* 2(a).

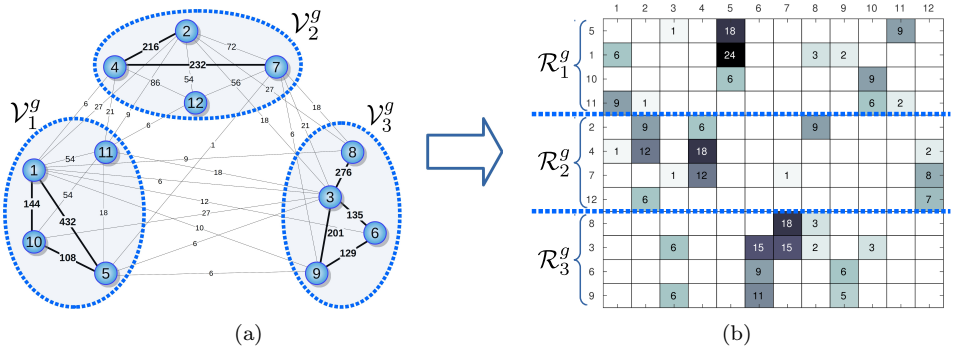


FIG. 3. (a) *Good* 3*-way partition* $\Pi^g(\mathcal{V})$ *of* $\mathcal{G}_{\text{RIP}}(A)$ *and* (b) 3*-way row partition of* A *induced by Figure* 3(a).

METIS [38]. Figure 3(b) shows the permuted A^Π matrix and block-row partition $\{\mathcal{R}_1^g, \mathcal{R}_2^g, R_3^g\}$ induced by the 3-way vertex partition $\Pi^g(\mathcal{V})$.

As seen in Figures 2 and 3, both straightforward and good block-row partitions achieve perfect balance on the row counts of blocks by having exactly four rows per block. The quality difference between straightforward and good block-row partitions can be easily seen by comparing the 3-way partitions of $\mathcal{G}_{\text{RIP}}$ in Figures 2(b) and 3(a), respectively. As seen in Figure 2(b), eight out of nine thick edges remain on the cut of $\Pi^s(\mathcal{V})$, whereas all of the nine thick edges remain internal in $\Pi^g(\mathcal{V})$ as seen in Figure 3(a).

Figure 4 shows the 3×3 block-checkerboard partitioning of the resulting matrix $C = AA^T$ induced by straightforward and good block-row partitioning of the sample matrix A in Figures 2(a) and 3(b), respectively. Note that both rows and columns of the C matrix are partitioned conformably with the row partitions of the A matrix. The comparison of Figures 4(a) and 4(b) shows that large nonzeros (dark cells) are scattered across the off-diagonal blocks of matrix C for the straightforward partitioning, whereas large nonzeros (dark cells) are clustered to the diagonal blocks of C for the good partitioning.

We introduce Figure 5 to compare the quality of the straightforward and good block-row partitions in terms of interblock row-inner-product sums. In the figure,

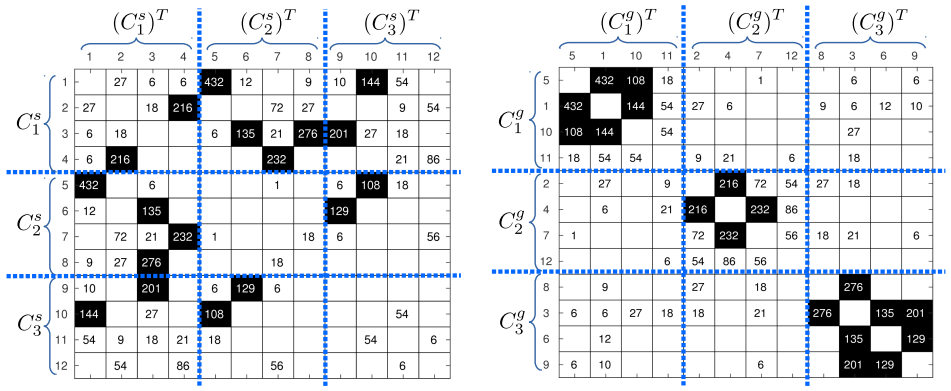


FIG. 4. 3×3 *block-checkerboard partition of matrix* $C = AA^T$ *induced by* 3*-way* (a) *straightforward (Figure* 2(a)*) and* (b) *good (Figure* 3(b)*) block-row partitions of matrix* A.

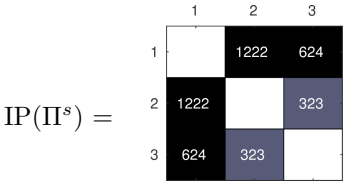


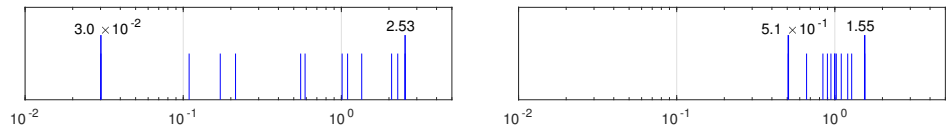
FIG. 5. *Interblock row inner-product matrix* IP(Π) *for* (a) *straightforward and* (b) *good block-row partitions.*

each off-diagonal entry ip_{km} of the 3×3 IP matrix shows the sum of the interblock row inner products between the respective block rows $\mathcal{R}_k$ and $\mathcal{R}_m$. That is,

$$\text{ip}_{km} \triangleq \sum_{\substack{\text{r}_i \in \mathcal{R}_k \\ \text{r}_j \in \mathcal{R}_m}} \left| \text{r}_i \text{r}_j^T \right| \quad \text{for } k \neq m. \tag{2.7}$$

As seen in Figure 5, $\text{ip}^s_{12} = 1{,}222$ for the straightforward partition, whereas $\text{ip}^g_{12} = 70$ for the good partition. Note that ip_{km} is also equal to the sum of the absolute values of the nonzeros of the off-diagonal block C_{km} at the kth row block and mth column block of the C matrix, i.e.,

$$\text{ip}_{km} = \sum_{\substack{\text{r}_i \in R_k \\ \text{r}_j \in R_m}} |c_{ij}|. \tag{2.8}$$



(a) Straightforward partitioning (Figure 2(a)) (b) Good partitioning (Figure 3(b))

FIG. 6. *Eigenvalue spectrum of H for the block-row partitionings of the sample matrix given in Figure* 1(a).

Therefore the total sum of interblock inner products is

$$\begin{aligned} \text{interIP}(\Pi^s) &= \text{ip}_{12}^s + \text{ip}_{13}^s + \text{ip}_{23}^s \\ &= 1{,}222 + 624 + 323 = 2{,}169 \end{aligned} \tag{2.9}$$

for the straightforward partition, whereas for the good partition it is

$$\text{interIP}(\Pi^g) = 70 + 94 + 90 = 254. \tag{2.10}$$

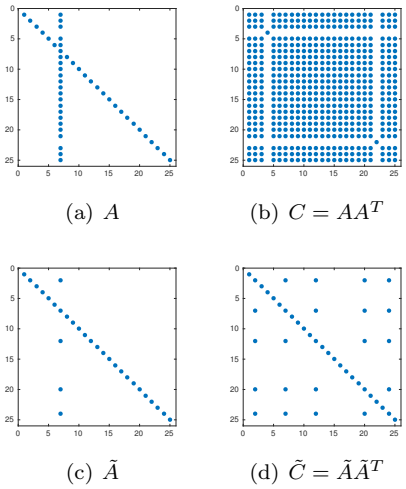
Figures 6(a) and 6(b), respectively, show the eigenvalue spectrum of H for the straightforward and good partitionings. As seen in the figures, for the straightforward partitioning the eigenvalues reside in the interval $[3.0 \times 10^{-2}, 2.53]$, whereas for good partitioning the eigenvalues reside in the interval $[5.1 \times 10^{-1}, 1.55]$. As seen in Figure 6(b), after using $\mathcal{G}_{\text{RIP}}$ partitioning the eigenvalues are much better clustered around 1 and the smallest eigenvalue is much larger than that of the straightforward partitioning method.

2.3. Implementation. Implementation of the proposed partitioning method consists of two stages which are constructing $\mathcal{G}_{\text{RIP}}$ and partitioning $\mathcal{G}_{\text{RIP}}$.

Constructing $\mathcal{G}_{\text{RIP}}$. For constructing $\mathcal{G}_{\text{RIP}}$, we choose to use basic sparse matrix-matrix multiplication (SpGEMM) [33] kernel due to existing efficient implementations. The edges of the $\mathcal{G}_{\text{RIP}}$ are obtained from the nonzeros of the $C = AA^T$ matrix, whereas their weights are obtained from the absolute values of those nonzeros.

Note that when matrix A has dense column(s), the corresponding matrix $C = AA^T$ will be quite dense. In other words, when a column has nz nonzeros, the corresponding C matrix will have at least nz^2 nonzeros. For example, Figure 7(a) shows a 25×25 sparse matrix A which has a dense column having 23 nonzero entries. As seen in Figure 7(b), matrix AA^T is dense as it has 531 nonzero entries. Clearly, a large number of nonzeros in C (i.e., a large number of edges in $\mathcal{G}_{\text{RIP}}$) increases the memory requirement and computation cost of SpGEMM as well as the time requirement for partitioning $\mathcal{G}_{\text{RIP}}$.

In order to alleviate the aforementioned problem, we propose the following methodology for sparsifying C. We identify a column $A(:, i)$ (in MATLAB [41] notation) of an $n \times n$ matrix A as a dense column if it contains more than $\sqrt{n}$ nonzeros ($nnz(A(:, i)) > \sqrt{n}$). Given A, we extract a sparse matrix $\tilde{A}$ by keeping the largest (in absolute value) $\sqrt{n}$ nonzeros of dense columns of A. That is, the smallest $nnz(A(:, i)) - \sqrt{n}$ entries of a dense A-matrix column $A(:, i)$ is ignored during constructing column $\tilde{A}(:, i)$ of $\tilde{A}$. Hence, the SpGEMM operation is performed on $\tilde{A}$ to obtain sparsified resulting matrix $\tilde{C} = \tilde{A}\tilde{A}^T$. This will lead to a sparsified $\tilde{\mathcal{G}}_{\text{RIP}}$ graph. For example, Figure 7(c) shows this sparsity pattern of a sparse matrix $\tilde{A}$ which is extracted from A by keeping the 5 largest nonzeros in the dense column of A. As seen



(a) A (b) $C = AA^T$

(c) $\tilde{A}$ (d) $\tilde{C} = \tilde{A}\tilde{A}^T$

FIG. 7. *Nonzero patterns of A, AA^T, $\tilde{A}$, and $\tilde{A}\tilde{A}^T$.*

in Figure 7(d), matrix $\tilde{C} = \tilde{A}\tilde{A}^T$ is very sparse with respect to Figure 7(b). Note that both $\tilde{A}$ and $\tilde{C}$ are used only for constructing $\tilde{\mathcal{G}}_{\text{RIP}}$ of A. After the partitioning stage, both matrices are discarded. In the rest of the paper, $\tilde{\mathcal{G}}_{\text{RIP}}$ will be referred to as $\mathcal{G}_{\text{RIP}}$ for the sake of the simplicity of presentation.

Partitioning $\mathcal{G}_{\text{RIP}}$. We use multilevel graph partitioning tool METIS [38] for partitioning $\mathcal{G}_{\text{RIP}}$. In order to compute integer edge weights required by METIS, we multiply the floating-point edge cost values with α and round them up to the nearest integer value; $wgt(v_i, v_j) = \lceil \alpha \times cost(v_i, v_j) \rceil$, where α is a sufficiently large integer. Here, $cost(v_i, v_j)$ is the edge cost computed according to (2.3) and $wgt(v_i, v_j)$ is the weight of the respective edge provided to METIS. Since the rows of matrix A are prescaled to have the 2-norm equal to one in the preprocessing phase, each edge cost $cost(v_i, v_j)$ should be in the range $(0, 1]$ and the resulting edge weight $wgt(v_i, v_j)$ will be an integer in the range $[1, \alpha]$.

3. Experimental results.

3.1. Experimental framework. In the experiments, we used the CG accelerated block Cimmino implementation available in ABCD Solver v1.0 [48]. In ABCD Solver, we used the MUMPS 5.1.2 [3] sparse direct solver to factorize the systems in (1.8) once and solve the system iteratively. We note that the proposed scheme is designed for the classical block Cimmino by improving the numerical orthogonality between blocks and it does not intend to improve the structural orthogonality. Hence, it is not applicable to the augmented block Cimmino algorithm that is also available in the ABCD Solver where the number of augmented columns depends only on structural orthogonality.

We adopted the same matrix scaling procedure [4] as in ABCD Solver. This is a parallel iterative procedure which scales the columns and rows of A so that the absolute value of the largest entry in each column and row is one. We first perform row and column scaling in order to avoid problems due to poor scaling of the input matrix. Then, we also perform row scaling on A to have 2-norm equal to exactly one, so that the actual values in AA^T would then correspond to cosines of the angles

between pairs of rows in A [23, 47]. We note that H is numerically independent of row scaling. However the column scaling affects H and it can be considered as preconditioner [43, 47].

ABCD Solver includes a stabilized block-CG accelerated block Cimmino algorithm [9] especially for solving systems with multiple right-hand side vectors. Since the classical CG is guaranteed to converge [42] for systems where the coefficient matrix is symmetric and positive definite and its convergence theory is well-established, in this work we utilized the classical CG accelerated block Cimmino in Algorithm 1.2 for solving a sparse linear system of equations with single right-hand side vector rather than the block-CG acceleration.

In parallel CG accelerated block Cimmino algorithm, the work distribution among processors is performed in exactly the same way as in ABCD Solver. That is, if the number of row blocks is larger than the number of processors, row blocks are distributed among processors so that each processor has equal workload in terms of number of rows. If the number of row blocks is smaller than the number of processors, a master-slave computational approach [5, 23, 24] is adopted. Each master processor owns a distinct row block and is responsible for inner-product and matrix-vector computations. Each slave processor is a supplementary processor which helps the specific master processor in the factorization and solution steps of MUMPS. After the analysis phase of MUMPS, slave processors are mapped to some master processors according to the information of FLOP estimation in the analysis phase.

In all experiments with the CG accelerated block Cimmino, we use the normwise backward error [8] at iteration t,

$$\gamma^{(t)} = \frac{\|Ax^{(t)} - f\|_\infty}{\|A\|_\infty \|x^{(t)}\|_1 + \|f\|_\infty} < 10^{-10}, \tag{3.1}$$

as the stopping criterion and we use 10,000 as the maximum number of iterations. The right-hand side vectors of the systems are obtained by multiplying the coefficient matrices with a vector whose elements are all one. In all instances, the CG iterations are started from the zero vector.

In the experiments, we have compared the performance of the proposed graph partitioning method (GP) against two baseline partitioning methods already available in ABCD Solver. The first baseline method, which is referred to as the uniform partitioning (UP) method in ABCD Solver, partitions the rows of the coefficient matrix into a given number of block rows with almost an equal number of rows without any permutation on the rows of A. Note that the UP method is the same as the straightforward partitioning method mentioned in section 2.

The second baseline method is the hypergraph partitioning (HP) method [23]. This method uses the column-net model [15] of sparse matrix A in which rows and columns are, respectively, represented by vertices and hyperedges both with unit weights [23]. Each hyperedge connects the set of vertices corresponding to the rows that have a nonzero on the respective column. In the HP method, a K-way partition of the vertices of the column-net model is used to find K block rows. The partitioning constraint of maintaining balance on part weights corresponds to finding block rows with an equal number of rows.

CM-based partitioning strategies [23, 43, 47] are also considered as another baseline approach. However, our experiments showed that CM-based strategies fail in producing the desired number of balanced partitions and achieving convergence for

most of the test instances. Due to a significantly larger number of failures of CM-based strategies compared to UP and HP, UP and HP are selected as much better baseline algorithms.

The HP method in ABCD Solver uses the multilevel hypergraph partitioning tool PaToH [16] for partitioning the column-net model of matrix A. Here, we use the same parameters for PaToH specified in ABCD Solver. That is, the final imbalance (i.e., ϵ in (A.2)) and initial imbalance (imbalance ratio of the coarsest hypergraph) parameters in PaToH are set to 50% and 100%, respectively. The other parameters are left as a default of PaToH as in ABCD Solver.

Parallel block Cimmino solution times on some real problems are experimented upon by using PaToH with different imbalance ratios in [23]. It is reported that although partitioning with weak balancing greatly reduces the number of interconnections which lead to a decrease in the number of iterations; however, it increases the parallel solution time because of highly unbalanced computational workload among processors. Therefore, finding good partition imbalance ratios can be important for the parallel performance of block Cimmino. Due to the space limitation, the impact of different imbalance ratios on the parallel performance is left as future work.

In the proposed GP method, as mentioned in subsection 2.3, the multilevel graph partitioning tool METIS is used to partition the row inner-product graph model $\mathcal{G}_{\text{RIP}}$ of A. The imbalance parameter of METIS is set to 10% and the k-way option is used. For the sake of a fair comparison between HP and GP methods, unit vertex weights are used in $\mathcal{G}_{\text{RIP}}$. The other parameters are left as a default of METIS. Since both PaToH and METIS use randomized algorithms, we report the results of the geometric mean of 5 experiments with different seeds for each instance.

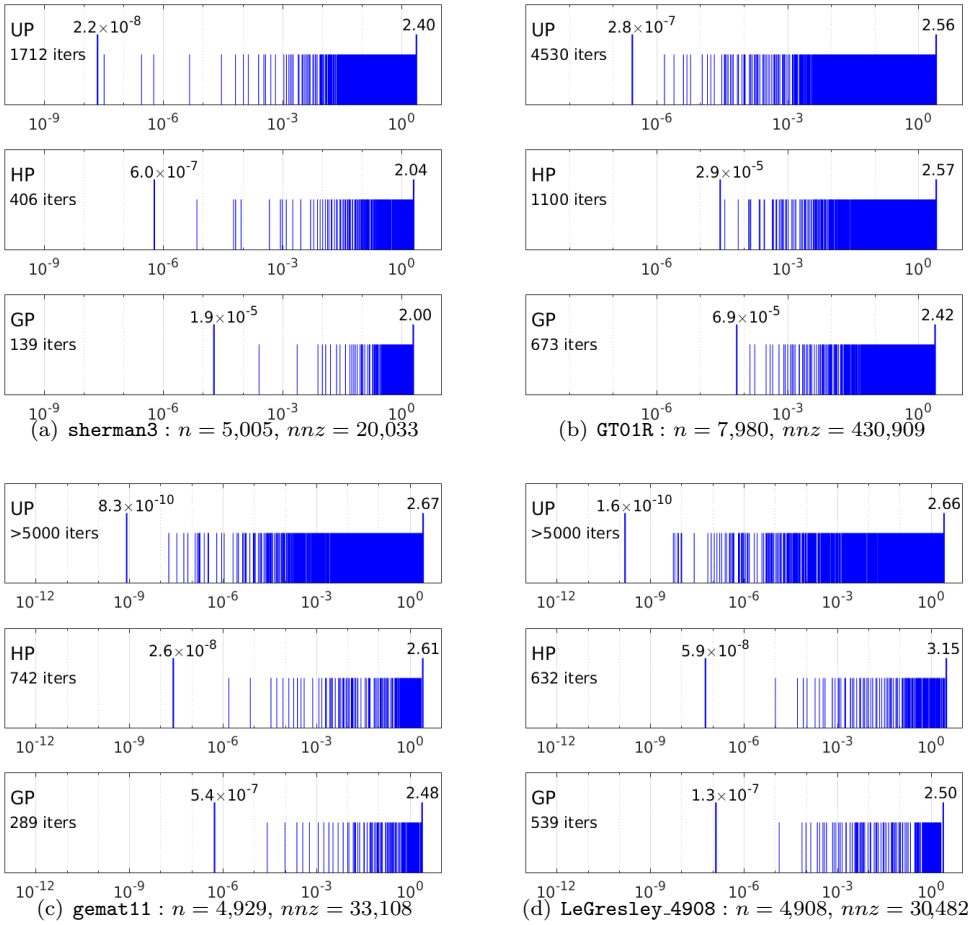
Here and hereafter, we use GP, HP, and UP to refer to the respective block-row partitioning methods as well as ABCD Solver that utilizes the regular block Cimmino algorithm for solving the systems partitioned by the respective method.

The extensive numerical experiments were conducted on a shared memory system. The shared memory system is a four socket 64-core computer that contains four AMD Opteron 6376 processors, where each processor has 16 cores running at 2.3 GHz and a total of 64 GB of DDR3 memory. Due to memory bandwidth limitations of the platform, experiments are performed with 32 cores.

ABCD Solver [48] is implemented in C/C++ programming language with MPI- and OpenMP-based hybrid parallelism. Furthermore, an additional parallelism level can be incorporated with multithreaded BLAS/LAPACK libraries. However, in the experiments, we used pure MPI-based parallelism which gives the best performance for our computing system.

3.2. Effect of block-row partitioning on the eigenvalue spectrum of H. The convergence rate of the CG accelerated block Cimmino algorithm is related to the eigenvalue spectrum of H. By the nature of the block Cimmino algorithm, most of the eigenvalues of H are clustered around 1, but there can be some eigenvalues at extremes of the spectrum.

In this subsection, we conduct experiments to study the effect of the partitioning on the eigenvalue spectrum of H by comparing the proposed GP method against UP and HP. In these experiments, in order to be able to compute the eigenvalue spectrum requiring a reasonable amount of time and memory, we use four small nonsymmetric sparse matrices: `sherman3`, `GT01R`, `gemat11`, and `LeGresley_4908` from the SuiteSparse matrix Collection [20]. The first and the second matrices arise in computational fluid dynamics problems, whereas the third and fourth matrices arise in



(a) `sherman3` : $n = 5{,}005$, $nnz = 20{,}033$ (b) `GT01R` : $n = 7{,}980$, $nnz = 430{,}909$

(c) `gemat11` : $n = 4{,}929$, $nnz = 33{,}108$ (d) `LeGresley_4908` : $n = 4{,}908$, $nnz = 30{,}482$

FIG. 8. *Eigenvalue spectrum of H (with the smallest and largest eigenvalues) and the number of CG iterations (iters) required for convergence.*

power network problems. We partition the matrices into 8 block rows for all of the three partitioning methods UP, HP, and GP.

Figure 8 shows the eigenvalue spectrum of H obtained by UP, HP, and GP methods for each test instance. The figure also reports the number of CG iterations (iters) required for convergence as well as the smallest and largest eigenvalues of H. As seen in the figure, both HP and GP methods achieve significantly better clustering of the eigenvalues around 1 compared to UP. This experimental finding is reflected in a remarkable decrease in the number of CG iterations attained by both HP and GP over UP.

In the comparison of HP and GP, GP achieves better eigenvalue clustering and hence a better convergence rate than HP for all instances. For `sherman3`, `GT01R`, `gemat11`, and `LeGresley_4908` instances, the better clustering quality attained by GP over HP leads to significant improvement in the convergence rate by 66%, 39%, 61%, and 15%, respectively.

3.3. Dataset for performance analysis. For the following experiments, we selected all nonsingular nonsymmetric square matrices whose dimensions are between

50,000 and 5,000,000 rows and columns from the SuiteSparse Matrix Collection [20]. The number of matrices based on this criteria turns out to be 112. Only the `HV15R` and `cage14` matrices are excluded due to memory limitations. We have observed that at least one of the three partitioning methods was able to converge in 76 instances out of these 110 in less than 10,000 CG iterations. Table 1 shows the properties of those 76 matrices that are used in the experiments. We note that the main advantage of the block Cimmino algorithm is its amenability to parallelism and requirement of less storage compared to direct methods. It would not be competitive for the smallest problems in the dataset against direct solvers or classical preconditioned iterative solvers.

In Table 1, the matrices are displayed in increasing sorted order according to their sizes. The matrices are partitioned into a number of partitions where each row block has approximately 20,000 rows if $n > 100{,}000$ or 10,000 rows if $n < 100{,}000$. Thus in our dataset, the smallest matrix is partitioned into 6 row blocks whereas the largest matrix is partitioned into 235 row blocks.

3.4. Convergence and parallel performance. In this subsection, we study the performance of the proposed GP method against UP and HP in terms of the number of CG iterations and parallel CG time to solution.

Table 2 shows the number of CG iterations and parallel CG time for each matrix. In the table, "`F`" denotes that an algorithm fails to reach the desired backward error in 10,000 iterations for the respective matrix instance. As seen in Table 2, UP and HP fail to converge in 26 and 18 test instances, respectively, whereas GP does not fail in any test instance.

In Table 2, the best result for each test instance is shown in bold. As seen in the table, out of 76 instances, the proposed GP method achieves the fastest convergence in 58 instances, whereas HP and UP achieve the fastest convergence in only 8 and 11 instances, respectively. As also seen in Table 2, GP achieves the fastest iterative solution time in 56 instances, whereas HP and UP achieve the fastest solution time in 11 and 9 instances, respectively.

Figure 9 shows the performance profiles [22] of 76 matrix instances which compare multiple methods over a set of test instances. A performance profile is used to compare multiple methods with respect to the best performing method for each test instance and report the fraction of the test instances in which performance is within a factor of that of the best method. For example, in Figure 9(a) a point (abscissa = 2, ordinate = 0.40) on the performance curve of a method refers to the fact that the performance of the respective method is no worse than that of the best method by a factor of 2 in approximately 40% of the instances. If a method is closer to the top-left corner, then it achieves a better performance.

In Figures 9(a) and 9(b), we show the performance profiles in terms of the number of CG iterations and parallel CG times, respectively. As seen in Figure 9(a), the number of CG iterations required by GP for convergence does not exceed that of the best method by a factor of 2 in approximately 95% of the instances, whereas HP and UP achieve the same relative performance compared to the best method in approximately 42% and 30% of the instances, respectively. As seen in Figure 9(b), the CG time using GP is not slower than that of the best method by a factor of 2 in approximately 95% of the instances. Whereas HP and UP achieve the same relative performance compared to the best method in approximately 45% and 23% of the instances, respectively. Figures 9(a) and 9(b) show that the CG time is directly proportional to the number of CG iterations as expected.

TABLE 1
Matrix properties (n: number of rows/columns, nnz: number of nonzeros).

Matrix name	n	nnz	Matrix name	n	nnz
rajat26	51,032	247,528	dc3	116,835	766,396
ecl32	51,993	380,415	trans4	116,835	749,800
2D_54019_highK	54,019	486,129	trans5	116,835	749,800
bayer01	57,735	275,094	matrix-new_3	125,329	893,984
TSOPF_RS_b39_c30	60,098	1,079,986	cage12	130,228	2,032,536
venkat01	62,424	1,717,792	FEM_3D_thermal2	147,900	3,489,300
venkat25	62,424	1,717,763	para-4	153,226	2,930,882
venkat50	62,424	1,717,777	para-10	155,924	2,094,873
laminar_duct3D	67,173	3,788,857	para-5	155,924	2,094,873
lhr71c	70,304	1,528,092	para-6	155,924	2,094,873
shyy161	76,480	329,762	para-7	155,924	2,094,873
circuit_4	80,209	307,604	para-8	155,924	2,094,873
epb3	84,617	463,625	para-9	155,924	2,094,873
poisson3Db	85,623	2,374,949	crashbasis	160,000	1,750,416
rajat20	86,916	604,299	majorbasis	160,000	1,750,416
rajat25	87,190	606,489	ohne2	181,343	6,869,939
rajat28	87,190	606,489	hvdc2	189,860	1,339,638
LeGresley_87936	87,936	593,276	shar_te2-b3	200,200	800,800
rajat16	94,294	476,766	stomach	213,360	3,021,648
ASIC_100ks	99,190	578,890	torso3	259,156	4,429,042
ASIC_100k	99,340	940,621	ASIC_320ks	321,671	1,316,085
matrix_9	103,430	1,205,518	ASIC_320k	321,821	1,931,828
hcircuit	105,676	513,072	ML_Laplace	377,002	27,582,698
lung2	109,460	492,564	RM07R	381,689	37,464,962
rajat23	110,355	555,441	language	399,130	1,216,334
Baumann	112,211	748,331	CoupCons3D	416,800	17,277,420
barrier2-1	113,076	2,129,496	largebasis	440,020	5,240,084
barrier2-2	113,076	2,129,496	cage13	445,315	7,479,343
barrier2-3	113,076	2,129,496	rajat30	643,994	6,175,244
barrier2-4	113,076	2,129,496	ASIC_680k	682,862	2,638,997
barrier2-10	115,625	2,158,759	atmosmodd	1,270,432	8,814,880
barrier2-11	115,625	2,158,759	atmosmodj	1,270,432	8,814,880
barrier2-12	115,625	2,158,759	Hamrle3	1,447,360	5,514,242
barrier2-9	115,625	2,158,759	atmosmodl	1,489,752	10,319,760
torso2	115,967	1,033,473	atmosmodm	1,489,752	10,319,760
torso1	116,158	8,516,500	memchip	2,707,524	13,343,948
dc1	116,835	766,396	circuit5M_dc	3,523,317	14,865,409
dc2	116,835	766,396	rajat31	4,690,002	20,316,253

It is clear that GP mainly aims at improving the convergence rate, whereas HP mainly aims at reducing total communication volume. We made additional measurements in order to discuss this trade-off between these two methods. Due to the lack of space, here we only summarize the average results using 32 processors. Although GP incurs 44% more total communication volume than HP per iteration, this results in only a 6% increase in the per-iteration execution time. Here, the per-iteration execution time for a given instance shows the overall parallelization efficiency attained by the respective partitioning method irregardless of its convergence performance. This experimental finding can be attributed to several other factors affecting the communication overhead in addition to the total communication volume as well as the fact that per-iteration execution time is dominated by the computational cost of local solution of the linear systems via a direct solver in block Cimmino. On average, although GP incurs only 6% more per-iteration time than HP, GP requires 59% fewer iterations for

TABLE 2
Number of CG iterations and parallel CG times in seconds.

Matrix name	# of CG iterations			Parallel CG time to soln.		
	UP	HP	GP	UP	HP	GP
rajat26	F	3303	**245**	F	62.4	**8.4**
ecl32	5307	1253	**314**	246.0	36.8	**10.1**
2D_54019_highK	42	**4**	9	0.9	**0.1**	0.2
bayer01	2408	382	**131**	63.8	8.7	**3.4**
TSOPF_RS_b39_c30	676	**262**	473	24.9	**4.6**	8.0
venkat01	54	37	**34**	1.8	**0.9**	0.9
venkat25	915	625	**599**	30.8	**14.3**	14.4
venkat50	1609	975	**970**	56.2	**23.5**	23.7
laminar_duct3D	466	630	**394**	29.6	39.0	**23.0**
lhr71c	F	6164	**4166**	F	193.3	**136.5**
shyy161	**13**	20	15	**0.4**	0.6	0.5
circuit_4	F	256	**183**	F	15.5	**12.3**
epb3	2583	3089	**2318**	87.7	100.9	**77.9**
poisson3Db	4797	983	**715**	715.0	51.9	**41.0**
rajat20	629	641	**322**	81.5	40.9	**34.6**
rajat25	1172	937	**448**	121.0	62.6	**48.9**
rajat28	556	369	**207**	86.3	22.3	**21.2**
LeGresley_87936	F	7625	**3102**	F	266.5	**122.2**
rajat16	6834	1022	**180**	835.0	68.5	**20.0**
ASIC_100ks	F	**23**	45	F	**1.0**	2.1
ASIC_100k	76	324	**49**	22.7	93.5	**13.4**
matrix_9	**3944**	F	9276	**272.0**	F	704.0
hcircuit	2061	2477	**460**	591.0	136.5	**32.7**
lung2	**12**	**12**	13	0.9	**0.8**	0.9
rajat23	F	7770	**501**	F	465.6	**33.4**
Baumann	**732**	1551	1340	**47.4**	86.5	89.8
barrier2-1	F	F	**1219**	F	F	**145.1**
barrier2-2	F	F	**1024**	F	F	**99.8**
barrier2-3	F	F	**1013**	F	F	**106.1**
barrier2-4	F	F	**1360**	F	F	**116.0**
barrier2-10	F	F	**1206**	F	F	**135.8**
barrier2-11	F	F	**1169**	F	F	**125.7**
barrier2-12	F	F	**1139**	F	F	**116.1**
barrier2-9	F	F	**1306**	F	F	**136.2**
torso2	16	**14**	15	0.7	**0.6**	0.7
torso1	F	**4376**	9200	F	**322.1**	833.7
dc1	629	2059	**83**	255.0	745.4	**32.9**
dc2	478	1313	**68**	170.3	504.0	**22.4**
dc3	2172	3329	**90**	793.0	1220.1	**31.9**
trans4	292	1416	**23**	105.9	452.1	**5.8**
trans5	1006	4533	**33**	368.7	1693.1	**8.5**
matrix-new_3	F	9739	**6707**	F	709.7	**519.4**
cage12	**9**	12	10	**1.7**	3.1	2.2
FEM_3D_thermal2	67	54	**45**	4.9	4.3	**3.8**
para-4	7675	F	**3546**	1600.0	F	**423.4**
para-10	F	F	**5565**	F	F	**588.6**
para-5	F	F	**5054**	F	F	**610.5**
para-6	F	F	**5019**	F	F	**574.1**
para-7	F	F	**4576**	F	F	**508.6**
para-8	F	F	**4973**	F	F	**535.7**
para-9	F	F	**5667**	F	F	**682.0**
crashbasis	68	**17**	21	10.0	**1.1**	1.4
majorbasis	48	**16**	18	6.6	**1.0**	1.2
ohne2	**2623**	F	3881	F	2103.1	**820.8**
hvdc2	F	5622	**3042**	F	446.7	**262.1**
shar_te2-b3	**23**	27	26	9.3	8.9	**8.8**
stomach	**8**	11	9	**1.1**	1.4	1.3
torso3	22	30	**11**	5.2	7.0	**3.1**
ASIC_320ks	F	20	**2**	F	4.5	**0.6**
ASIC_320k	114	37	**11**	56.9	40.4	**8.0**
ML_Laplace	8615	9136	**8438**	**3890.3**	4220.8	4215.4
RM07R	F	F	**3944**	F	F	**10200.0**
language	974	661	**453**	335.0	196.4	**159.0**
CoupCons3D	277	132	**107**	100.0	57.8	**48.3**
largebasis	1155	701	**348**	549.1	174.8	**88.3**
cage13	**10**	13	12	**6.7**	14.7	10.6
rajat30	157	200	**61**	229.0	274.5	**86.3**
ASIC_680k	10	19	**2**	11.3	38.2	**3.5**
atmosmodd	**744**	2055	1183	**869.2**	2115.2	1279.6
atmosmodj	**787**	2235	1253	**994.0**	2323.2	1293.1
Hamrle3	F	2394	**2010**	F	2375.8	**2012.9**
atmosmodl	1206	805	**379**	1800.6	1005.3	**484.0**
atmosmodm	1164	697	**202**	1730.0	871.4	**257.6**
memchip	3278	1002	**379**	7450.3	1589.9	**862.3**
circuit5M_dc	173	58	**10**	512.0	143.6	**23.4**
rajat31	2840	2767	**1500**	9590.1	9456.2	**5166.6**
Number of bests	**11**	**8**	**58**	**9**	**11**	**56**

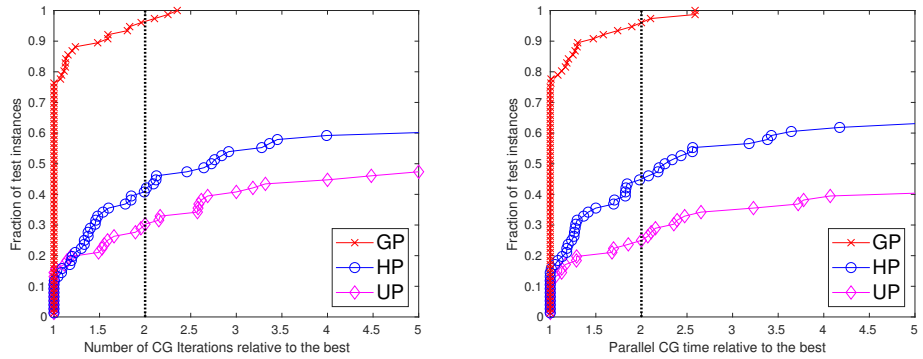


FIG. 9. *Performance profiles for* (a) *convergence rate and* (b) *parallel CG time.*

convergence than HP. This explains the significant overall performance improvement achieved by GP against HP.

3.5. Preprocessing overhead and amortization. In this subsection, we analyze the relative preprocessing overhead of the three methods UP, HP, and GP in order to find out whether the intelligent partitioning methods HP and GP are amortized. For all of the three block-row partitioning methods, the preprocessing overhead includes matrix scaling, block-row partitioning, creating the submatrices corresponding to these block rows and distributing these submatrices among processors. Recall that the partitioning for UP is straightforward and hence it incurs only negligible additional cost to the preprocessing overhead. On the other hand, intelligent partitioning algorithms utilized in HP and GP incur considerable amount of cost to the overall preprocessing overhead. Furthermore, for GP, construction of the row inner-product graph also incurs a significant amount of cost.

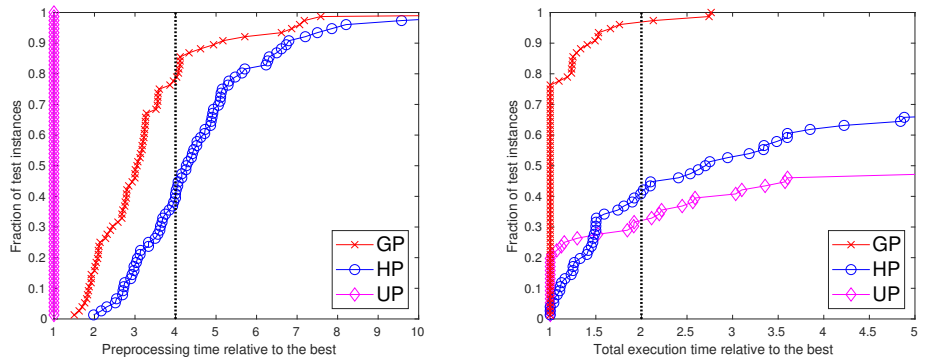
In Table 3, we display total preprocessing time and total execution time for all three methods for each one of the 76 test instances. In the table, total execution time is the sum of the total preprocessing and the total solution times. Here the total solution time includes parallel factorization and parallel CG solution times. Note that the factorization in block Cimmino algorithm in ABCD Solver is performed by parallel sparse direct solver MUMPS [3]. The factorization, which needs to be performed only once, involves symbolic and numerical factorizations of the coefficient matrices of the augmented systems that arise in the block Cimmino algorithm. Note that this factorization process is embarrassingly parallel since the factorization of the coefficient matrices of the augmented systems are done independently. Recall that MUMPS is also used during the iterative solution stage, where at each iteration a linear system is solved using the factors of the augmented systems that were computed during the factorization stage.

As seen in Table 3, in terms of the preprocessing time, UP is the clear winner in all of the 76 instances as expected, whereas GP incurs much less preprocessing time than HP in all except 7 instances. As also seen in Table 3, comparing all three methods, GP achieves the smallest total execution time in 57 instances, whereas UP and HP, respectively, achieve the smallest total execution time in only 15 and 4 instances.

In the relative comparison of GP and UP, GP incurs only 209% more preprocessing time than UP, on average, which leads GP to achieve less total execution time than UP in 61 out of 76 instances. In other words, compared to UP, the sequential

TABLE 3
Preprocessing time and total execution time (including preprocessing) in seconds.

Matrix name	Preprocessing time			Total execution time		
	UP	HP	GP	UP	HP	GP
rajat26	0.12	0.53	0.52	F	63.7	**9.3**
ecl32	0.20	0.98	0.54	246.7	39.3	**11.8**
2D_54019_highK	0.22	0.69	0.50	1.5	1.4	**1.3**
bayer01	0.14	0.41	0.39	64.4	9.7	**4.0**
TSOPF_RS_b39_c30	0.47	0.93	0.87	26.1	**6.2**	9.4
venkat01	0.75	2.02	1.44	3.0	3.4	**2.8**
venkat25	0.67	2.05	1.55	32.0	16.9	**16.7**
venkat50	0.76	2.07	1.46	57.3	26.0	**25.8**
laminar_duct3D	1.30	5.89	4.63	34.3	53.8	**33.8**
lhr71c	0.60	1.82	1.68	F	197.0	**140.3**
shyy161	0.16	0.50	0.32	**1.2**	1.7	1.5
circuit_4	0.16	0.79	1.13	F	16.6	**13.1**
epb3	0.21	0.70	0.44	88.3	102.1	**78.7**
poisson3Db	1.10	6.02	5.08	723.0	60.2	**48.6**
rajat20	0.31	2.05	2.05	83.0	44.1	**37.8**
rajat25	0.28	2.10	2.01	122.1	66.5	**49.9**
rajat28	0.30	2.02	2.06	87.9	25.8	**24.4**
LeGresley_87936	0.35	0.88	0.64	F	294.7	**123.5**
rajat16	0.24	1.97	1.82	836.5	71.8	**22.9**
ASIC_100ks	0.37	1.50	1.32	F	**3.1**	4.1
ASIC_100k	0.51	2.24	1.62	27.7	99.5	**18.2**
matrix_9	0.57	2.19	1.81	**275.0**	F	753.9
hcircuit	0.27	0.73	0.81	593.4	137.8	**34.2**
lung2	0.24	0.52	0.46	**1.6**	1.8	1.9
rajat23	0.31	1.12	1.77	F	467.9	**37.9**
Baumann	0.39	1.39	0.81	**49.4**	89.5	92.5
barrier2-1	1.10	4.40	3.58	F	F	**150.2**
barrier2-2	1.10	4.40	3.56	F	F	**109.0**
barrier2-3	1.10	4.42	3.45	F	F	**121.0**
barrier2-4	1.10	4.35	3.58	F	F	**125.8**
barrier2-10	1.20	4.42	3.61	F	F	**147.2**
barrier2-11	1.20	4.40	3.66	F	F	**136.3**
barrier2-12	1.10	4.47	3.61	F	F	**128.0**
barrier2-9	1.10	4.32	3.54	F	F	**145.7**
torso2	0.46	1.17	0.86	**1.8**	2.6	2.4
torso1	3.10	35.19	7.59	F	**421.5**	983.1
dc1	0.37	1.82	1.49	287.7	752.9	**51.3**
dc2	0.35	1.77	1.44	204.0	514.1	**30.3**
dc3	0.37	1.80	1.51	828.1	1233.9	**48.5**
trans4	0.39	2.01	1.54	129.5	462.0	**9.9**
trans5	0.40	2.05	1.42	377.3	1705.9	**12.4**
matrix-new_3	0.42	1.90	1.37	F	715.0	**524.1**
cage12	0.97	5.47	3.99	**23.9**	64.2	33.6
FEM_3D_thermal2	1.40	4.90	3.75	**8.3**	11.6	10.3
para-4	1.40	6.62	4.99	1605.4	F	**478.3**
para-10	1.00	4.45	3.07	F	F	**673.6**
para-5	0.91	4.82	3.20	F	F	**619.7**
para-6	1.10	4.57	3.13	F	F	**610.2**
para-7	1.00	4.62	3.20	F	F	**552.0**
para-8	1.10	4.57	3.07	F	F	**608.8**
para-9	1.00	4.72	3.23	F	F	**691.2**
crashbasis	0.84	2.50	2.01	13.2	5.2	**5.1**
majorbasis	0.72	2.40	2.11	9.6	5.1	**5.0**
ohne2	2.40	12.24	9.27	F	2128.6	**843.9**
hvdc2	0.79	1.82	1.34	F	449.4	**264.2**
shar_te2-b3	0.16	2.97	3.10	**24.2**	35.2	35.8
stomach	1.20	5.12	3.21	**5.1**	9.5	7.8
torso3	1.90	9.27	4.90	**14.1**	25.5	17.4
ASIC_320ks	1.10	3.02	3.02	F	8.7	**5.2**
ASIC_320k	1.30	12.45	5.34	61.9	64.9	**20.4**
ML_Laplace	9.30	58.70	29.05	**3913.4**	4269.0	4288.5
RM07R	14.00	110.00	69.14	F	F	**10727.5**
language	1.20	4.50	4.32	338.5	204.8	**160.3**
CoupCons3D	6.40	32.72	17.66	109.5	103.0	**84.5**
largebasis	2.20	6.37	4.61	553.1	182.9	**98.8**
cage13	4.20	30.25	17.10	**78.6**	301.1	217.3
rajat30	2.90	19.71	14.99	326.9	350.0	**127.5**
ASIC_680k	2.60	16.21	7.79	**19.3**	228.2	23.8
atmosmodd	5.60	36.38	11.97	**881.9**	2165.8	1464.6
atmosmodj	6.80	36.17	12.29	**1006.7**	2372.9	1535.1
Hamrle3	4.70	20.00	9.92	F	**2406.8**	2469.9
atmosmodl	7.20	45.18	14.71	1815.8	1067.0	**585.5**
atmosmodm	8.00	45.68	14.07	1747.3	934.9	**333.2**
memchip	10.00	43.12	20.15	7476.3	1640.0	**893.4**
circuit5M_dc	16.00	58.06	25.95	540.3	209.7	**64.5**
rajat31	25.00	74.36	37.53	9634.0	9539.1	**5220.2**
	Geometric means			Number of bests		
	0.9	**4.2**	**2.9**	**15**	**4**	**57**



(a) (b)

FIG. 10. *Performance profiles for* (a) *preprocessing time and* (b) *total execution time for GP, HP and UP methods.*

implementation of GP amortizes its preprocessing cost for those 61 instances by reducing the number of CG iterations sufficiently. That is, GP amortizes its preprocessing cost for the solution of those 61 instances even if we solve only one linear system with a single right-hand side vector. Note that in many applications in which sparse linear systems arise, the solution of consecutive linear systems are required where the coefficient matrix remains the same but the right-hand side vectors change. In such applications, the amortization performance of the proposed GP method will further improve. For example, the amortization performance of GP will improve from 61 to 64 instances for solving only two consecutive linear systems.

In Figures 10(a) and 10(b), we show the performance profiles in terms of the preprocessing time and the total execution time, respectively. As seen in Figure 10(a), the preprocessing overhead incurred by GP remains no worse than that of the best method UP by a factor of 4 in approximately 77% of instances. As seen in Figure 10(b), the total execution time attained by GP does not exceed that of the best method by a factor of 2 in 96% of the instances. On the other hand, HP and UP achieve the same relative performance compared to the best method only in approximately 44% and 33% of the instances, respectively.

4. Conclusion and future work. In this paper, we proposed a novel partitioning method in order to improve the CG accelerated block Cimmino algorithm. The proposed partitioning method takes the numerical orthogonality between block rows of the coefficient matrix into account. The experiments on a large set of real world systems show that the proposed method improves the convergence rate of the CG accelerated block Cimmino compared to the state-of-the-art hypergraph partitioning method. Moreover, it requires not only less preprocessing time and a fewer number of CG iterations, but also much less total execution time than the hypergraph partitioning method.

As a future work, we consider two issues: further reducing the number of iterations through preconditioning and reducing the preprocessing overhead through parallelization.

Even though the H matrix is not available explicitly, it could still be possible to obtain a preconditioner, further reducing the required number of CG iterations. One viable option could be using a sparse approximate inverse-type preconditioner where

the coefficient matrix does not need to be available explicitly. This approach could be viable especially when consecutive linear systems are needed to be solved with the same coefficient matrix.

The proposed method involves two computational stages, namely, constructing a row inner-product graph via computing the SpGEMM operation and partitioning this graph. For parallelizing the first stage, parallel SpGEMM [1, 2, 10] operation could be used to construct local subgraphs on each processor. For parallelizing the second stage, a parallel graph partitioning tool ParMETIS [37] could be used. In each processor, the local subgraphs generated in parallel in the first stage could be used as input for ParMETIS.

Appendix A. Graph and graph partitioning. An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Each edge $(v_i, v_j) \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. Each vertex $v_i \in \mathcal{V}$ can be assigned a weight shown as $w(v_i)$ and each edge $(v_i, v_j) \in \mathcal{E}$ can be assigned a cost shown as $cost(v_i, v_j)$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is defined as a K-way vertex partition of $\mathcal{G}$ if parts are mutually disjoint and exhaustive. An edge (v_i, v_j) is said to be cut if the vertices v_i and v_j belong to different vertex parts and uncut otherwise. The set of cut edges of a partition Π is denoted as $\mathcal{E}_{\text{cut}}$. In a given partition Π of $\mathcal{G}$, the weight W_k of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, i.e.,

$$W_k \triangleq \sum_{v_i \in \mathcal{V}_k} w(v_i). \tag{A.1}$$

In the graph partitioning problem, the partitioning constraint is to maintain a given balance condition on the part weights, i.e.,

$$W_k \leq W_{avg}(1+\epsilon) \text{ for } k = 1, 2, \ldots, K, \text{ where } W_{avg} = \sum_{v_i \in \mathcal{V}} w(v_i)/K. \tag{A.2}$$

Here ϵ is the predefined maximum imbalance ratio. The partitioning objective is to minimize the cutsize defined as the sum of the costs of the cut edges, i.e.,

$$\text{cutsize}(\Pi) \triangleq \sum_{(v_i, v_j) \in \mathcal{E}_{cut}} cost(v_i, v_j). \tag{A.3}$$

Acknowledgments. This work was done when the first author was at Bilkent University, Ankara, Turkey and was revised when the first author was at IRIT-CNRS, Toulouse, France.

REFERENCES

[1] K. Akbudak and C. Aykanat, *Simultaneous input and output matrix partitioning for outer-product–parallel sparse matrix-matrix multiplication*, SIAM J. Sci. Comput., 36 (2014), pp. C568–C590.

[2] K. Akbudak and C. Aykanat, *Exploiting locality in sparse matrix-matrix multiplication on many-core architectures*, IEEE Transactions on Parallel and Distributed Systems, 28 (2017), pp. 2258–2271.

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

[4] P. R. Amestoy, I. Duff, D. Ruiz, and B. Uçar, *A parallel matrix scaling algorithm*, in International Conference on High Performance Computing for Computational Science, Springer, Berlin, 2008, pp. 301–313.
[5] M. Arioli, A. Drummond, I. Duff, and D. Ruiz, *Parallel block iterative solvers for heterogeneous computing environments*, in Algorithms and Parallel VLSI Architectures III, Elsevier, Amsterdam, 1995, pp. 97–108.
[6] M. Arioli, I. Duff, and P. P. de Rijk, *On the augmented system approach to sparse least-squares problems*, Numer. Math., 55 (1989), pp. 667–684.
[7] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for sparse matrices*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 47–70.
[8] M. Arioli, I. Duff, and D. Ruiz, *Stopping criteria for iterative solvers*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 138–144.
[9] M. Arioli, I. S. Duff, D. Ruiz, and M. Sadkane, *Block Lanczos techniques for accelerating the block Cimmino method*, SIAM J. Sci. Comput., 16 (1995), pp. 1478–1511.
[10] A. Azad, G. Ballard, A. Buluç, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, and S. Williams, *Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication*, SIAM J. Sci. Comput., 38 (2016), pp. C624–C651.
[11] Å. Björck and G. H. Golub, *Numerical methods for computing angles between linear subspaces*, Math. Comp., 27 (1973), pp. 579–594.
[12] E. S. Bolukbasi and M. Manguoglu, *A multithreaded recursive and nonrecursive parallel sparse direct solver*, in Advances in Computational Fluid-Structure Interaction and Flow Simulation, Springer, Cham, Switzerland, 2016, pp. 283–292.
[13] R. B. Bramley, *Row Projection Methods for Linear Systems*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Champaign, IL, 1989.
[14] R. Bramley and A. Sameh, *Row projection methods for large nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 168–193.
[15] U. V. Catalyurek and C. Aykanat, *Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Syst., 10 (1999), pp. 673–693.
[16] U. V. Catalyurek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool*, Technical report BU-CE-9915, Boston University, Boston, 1999.
[17] Y. Censor, *Parallel application of block-iterative methods in medical imaging and radiation therapy*, Math. Program., 42 (1988), pp. 307–325.
[18] G. Cimmino and C. N. delle Ricerche, *Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari*, Ric Sci. Ser. II, 9 (1938), pp. 326–333.
[19] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th National Conference, ACM, New York, 1969, pp. 157–172.
[20] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1.
[21] J. W. Demmel and N. J. Higham, *Improved error bounds for underdetermined system solvers*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 1–14.
[22] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Math. Program., 91 (2002), pp. 201–213.
[23] L. Drummond, I. Duff, R. Guivarch, D. Ruiz, and M. Zenadi, *Partitioning strategies for the block Cimmino algorithm*, J. Engrg. Math., 93 (2015), pp. 21–39.
[24] I. S. Duff, R. Guivarch, D. Ruiz, and M. Zenadi, *The augmented block Cimmino distributed method*, SIAM J. Sci. Comput., 37 (2015), pp. A1248–A1269.
[25] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numer. Math., 35 (1980), pp. 1–12.
[26] T. Elfving and T. Nikazad, *Properties of a class of block-iterative methods*, Inverse Problems, 25 (2009), 115011.
[27] M. Galgon, L. Kraemer, J. Thies, A. Basermann, and B. Lang, *On the parallel iterative solution of linear systems arising in the feast algorithm for computing inner eigenvalues*, Parallel Comput., 49 (2015), pp. 153–163.
[28] S. A. Gerschgorin, *Uber die abgrenzung der eigenwerte einer matrix*, Izv. Akad. Nauk. USSR Otd. Fiz-Mat. Nauk. 6 (1931), pp. 749–754 (in German).
[29] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, J. SIAM Ser. B Numer. Anal., 2 (1965), pp. 205–224.
[30] G. H. Golub and M. A. Saunders, *Linear Least Squares and Quadratic Programming*, Technical report, Stanford University, Stanford, CA, 1969.
[31] G. H. Golub and C. F. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, MD, 1983.

[32] D. Gordon and R. Gordon, *Component-averaged row projections: A robust, block-parallel scheme for sparse linear systems*, SIAM J. Sci. Comput., 27 (2005), pp. 1092–1117.
[33] F. G. Gustavson, *Two fast algorithms for sparse matrices: Multiplication and permuted transposition*, ACM Trans. Math. Software, 4 (1978), pp. 250–269.
[34] A. Jennings, *Influence of the eigenvalue spectrum on the convergence rate of the conjugate gradient method*, IMA J. Appl. Math., 20 (1977), pp. 61–72.
[35] S. Kaczmarz, *Angenäherte auflösung von systemen linearer gleichungen*, Bull. Internat. Acad. Polonaise Sci. Lett., 35 (1937), pp. 355–357.
[36] C. Kamath and A. Sameh, *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Comput., 9 (1989), pp. 291–312.
[37] G. Karypis and V. Kumar, *A parallel algorithm for multilevel graph partitioning and sparse matrix ordering*, J. Parallel Distrib. Comput., 48 (1998), pp. 71–95.
[38] G. Karypis and V. Kumar, *METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering*, http://www.cs.umn.edu/~metis (2013).
[39] N. Li and Y. Saad, *MIGR: A multilevel incomplete QR preconditioner for large sparse least-squares problems*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 524–550.
[40] X. S. Li, *An overview of SuperLU: Algorithms, implementation, and user interface*, ACM Trans. Math. Software, 31 (2005), pp. 302–325.
[41] MATLAB, *Version* 8.6.0 *(R*2015*b)*, The MathWorks Inc., Natick, MA, 2015.
[42] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.
[43] D. Ruiz, *Solution of Large Sparse Unsymmetric Linear Systems with a Block Iterative Method in a Multiprocessor Environment*, Ph.D. thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 1992.
[44] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with pardiso*, Future Gen. Comput. Syst., 20 (2004), pp. 475–487.
[45] F. S. Torun, M. Manguoglu, and C. Aykanat, *Parallel minimum norm solution of sparse block diagonal column overlapped underdetermined systems*, ACM Trans. Math. Software, 43 (2017), 31.
[46] A. van der Sluis and H. A. van der Vorst, *The rate of convergence of conjugate gradients*, Numer. Math., 48 (1986), pp. 543–560.
[47] M. Zenadi, *Méthodes Hybrides pour la Résolution de Grands Systèmes Linéaires Creux sur Calculateurs Parallèles*, Ph.D. thesis, École Doctorale Mathématiques, Informatique et Télécommunications, Toulouse, France, 2013.
[48] M. Zenadi, D. Ruiz, and R. Guivarch, *The Augmented Block Cimmino Distributed Solver*. http://abcd.enseeiht.fr/ (2015).