PARTITIONING AND REORDERING FOR SPIKE-BASED
DISTRIBUTED-MEMORY PARALLEL GAUSS-SEIDEL *
TUGBA TORUN†, F. SUKRU TORUN‡, MURAT MANGUOGLU§, AND CEVDET
AYKANAT †

Abstract. Gauss-Seidel (GS) is a widely-used iterative method for solving sparse linear system
equations and also known to be effective as a smoother in algebraic multigrid methods. Paralleliza-
tion of GS is a challenging task since solving the sparse lower triangular system in GS constitutes
a sequential bottleneck at each iteration. We propose a distributed-memory parallel GS (dmpGS)
by implementing a parallel sparse triangular solver (stSpike) based on the Spike algorithm. stSpike
decouples the global triangular system into smaller systems that can be solved concurrently and
requires the solution of a much smaller reduced sparse lower triangular system which constitutes a
sequential bottleneck. In order to alleviate this bottleneck and to reduce the communication over-
head of dmpGS, we propose a partitioning and reordering model consisting of two phases. The first
phase is a novel hypergraph partitioning model whose partitioning objective simultaneously encodes
minimizing the reduced system size and the communication volume. The second phase is an in-block
row reordering method for decreasing the nonzero count of the reduced system. Extensive experi-
ments on a dataset consisting of 359 sparse linear systems verify the effectiveness of the proposed
partitioning and reordering model in terms of reducing the communication and the sequential com-
putational overheads. Parallel experiments on 12 large systems using up to 320 cores demonstrate
that the proposed model significantly improves the scalability of dmpGS.

Key words. parallel Gauss-Seidel, distributed-memory, Spike algorithm, parallel sparse trian-
gular solve, linear system solution, hypergraph partitioning, sparse matrix reordering.

AMS subject classifications. 68W10, 05C65, 65F50, 65F10

1. Introduction. A wide range of applications in science and engineering re-
quire the solution of a sparse linear system of equations

(1.1) \[ Ax = f, \]

where \( A \in \mathbb{R}^{m \times m} \) is a general large sparse invertible matrix; and \( x \) and \( f \in \mathbb{R}^m \) are
the unknown and right hand side vectors, respectively. Depending on the numerical
and structural properties of the coefficient matrix, various solvers have been proposed.

Direct solvers require a sequence of operations: reordering and partitioning, sym-
bolic factorization, numerical factorization, and finally obtaining the solution, typ-
ically via forward and backward sweeps. The reordering and partitioning schemes
are used both to reduce the amount of fill-in and to enhance the parallel scalability.
Symbolic factorization is used to determine the sparsity structure of the factors, and
finally the numerical factorization (such as sparse LU [23], QR [34], SVD [13] and WZ
[17]) is computed. Direct solvers are robust and, in general, are known to be very
scalable during the factorization phase [5, 43], but not so much during the triangular
solution phase [45].

*Submitted to the editors April 11, 2021.

Funding: Computing resources used in this work were provided by the National Center for High
Performance Computing of Turkey (UHeM) under grant number 4007772020. The third author was
supported by the BAGEP Award of the Science Academy.

†Department of Computer Engineering, Bilkent University, 06800 Ankara, Turkey (tugba.uzuner@bilkent.edu.tr, aykanat@cs.bilkent.edu.tr).
‡Department of Computer Engineering, Ankara Yildirim Beyazit University, 06010, Ankara, Turkey (ftorun@ybu.edu.tr).
§Department of Computer Engineering, Middle East Technical University, 06800 Ankara, Turkey (manguoglu@ceng.metu.edu.tr).

This manuscript is for review purposes only.
Iterative solvers, on the other hand, are known to be more scalable but not as robust as direct solvers. Nevertheless, they are still preferred for large sparse systems due to their lower memory requirements. Starting with an initial guess for the solution vector, these methods improve the solution at each iteration. There are two main types of iterative solvers: stationary and non-stationary methods. Stationary methods have the general form $x^{(k+1)} = \phi(x^{(k)})$ where $x^{(k)}$ is the solution vector at the $k^{th}$ iteration and $\phi(\cdot)$ is a function which does not change during the iterations. Some examples are Jacobi, Gauss-Seidel, Successive Over Relaxation (SOR) and Symmetric SOR (SSOR) [34, 59]. Non-stationary methods have the form $x^{(k+1)} = \phi^{(k)}(x^{(k)})$ in which the function $\phi^{(k)}(\cdot)$ changes at each iteration. Some examples are projection methods, Krylov subspace methods and Chebyshev iterations [9, 35, 59].

In practice, linear systems are preconditioned to reduce the required number of iterations of the iterative solvers and to improve their robustness. There could be a variety of choices of preconditioners, some are problem specific and others are more general. General classical preconditioners include, incomplete factorization based preconditioners (such as incomplete LU [58, 59]), sparse approximate inverse [11], algebraic multigrid (AMG) [51, 57], and others. We refer the reader to [10] for a detailed survey of preconditioners. Among these preconditioners, AMG has been widely used recently in many applications [12, 30, 53] which is a generalization of Geometric Multigrid (GMG) [70]. GMG requires some knowledge of the physical problem and/or its geometry, while there is no such requirement for AMG. AMG can be also used as a direct solver [36, 71]. Furthermore, AMG typically uses another iterative method as a “smoother” which is required to reduce the error at each level and the smoother itself can also be preconditioned. More recently a preferred smoother for AMG is Gauss-Seidel [3, 16, 67], as in BoomerAMG [36] and Trilinos-ML [32].

Gauss-Seidel (GS) is a well-known stationary iterative method which solves the linear system (1.1) by splitting the coefficient matrix into its lower and strictly upper triangular parts, $A=L+U$. Then the solution is obtained iteratively by

$$x^{(k+1)} = L^{-1}(f - Ux^{(k)}).$$

At each iteration of GS, both a lower triangular system is required to be solved and an upper triangular SpMV (sparse matrix-vector multiplication) is performed. GS is guaranteed to converge if $A$ is strictly or irreducibly diagonal dominant [7] or symmetric positive definite [34]. It is known to be effective and preferred as a smoother for a wide variety of problems [3, 72]. However, a true distributed-memory parallelization of GS is considered to be a challenging task [3].

In the literature, parallel GS implementations are proposed either to solve the original problem (1.1) [6, 42, 62] or to use it as a smoother in multigrid schemes [38, 64, 73]. A commonly-used method to parallelize GS by finding independent sub-tasks is the red-black coloring strategy [2, 31, 41], which has been extended to multi-coloring [33, 52, 4] to attain more parallelism for complicated regular problems. However, multi-colored GS is not feasible for some cases such as unstructured finite element applications since the number of colors becomes too large [42]. Another approach is to use a processor-localized GS in which each processor performs GS as a subdomain solver, but its convergence rate is low and may diverge for a large number of processors [3].

The main difficulty in parallelizing GS inherits from the sequential nature of triangular solve included in GS [72]. Along with its importance in several applications, solving triangular systems often constitutes a sequential bottleneck because of the dependencies between unknowns in forward or backward substitutions. In [60], a
parallel banded triangular solver is proposed. This algorithm is extended for solving banded linear systems [21, 28] and further improved by implementing various alternatives in each step of the factorization including the solution of the reduced system in [55, 56, 63]. At this point, the algorithm is called Spike algorithm. For sparse linear systems, Spike is also proposed as a solver for a banded preconditioner that is sparse within the band [49, 61], and it is generalized for sparse linear systems [15, 47, 48].

In [69], a Spike-based parallel solver for general tridiagonal systems is implemented for GPU architectures. A recent study [22] proposes a multi-threaded parallel solver for sparse triangular systems by extending the Spike algorithm [60].

We propose a distributed-memory parallel GS (dmpGS) by implementing and using a distributed-memory version of the sparse triangular Spike (stSpike) algorithm. stSpike enables obtaining the solution of the system by solving independent sparse triangular subsystems and a smaller reduced triangular system. Solving this reduced system constitutes a sequential computational bottleneck in dmpGS. The size of this reduced system is equal to the number of nonzero columns in the lower off-diagonal blocks of the coefficient matrix. The computational cost of solving the reduced system is proportional to its nonzero count. The communication volume of dmpGS is equal to the number of nonzero column segments in the off-diagonal blocks plus the reduced system size. Both of these communication and computational overheads highly depend on the sparsity structure of the coefficient matrix.

We note that solving the reduced system is embarrassingly parallel if the coefficient matrix is banded and diagonally dominant [50, 54]. In case the coefficient matrix is not diagonally dominant, another way to alleviate the cost of solving the reduced system is to further parallelize the solution of the reduced system which has been done iteratively [55], or recursively [15, 56]. Instead, we propose to minimize the size and the nonzero count of the reduced system, together with the communication volume, and show that the resulting reduced system is so small that further parallelization of the solution of the reduced system is often no longer needed. For attaining these minimization objectives, we propose a partitioning and reordering model that exploits the sparsity of the coefficient matrix. The proposed model consists of two phases. The first phase is a row-wise partitioning of the coefficient matrix, whereas the second phase is a row reordering within the row blocks induced by the partition obtained in the first phase.

For the first phase, we propose a novel hypergraph model that extends and enhances the conventional column-net model for simultaneously decreasing the reduced system size and the communication volume. We introduce vertex fixing, net anchor- ing and net splitting schemes within the recursive bipartitioning framework to encode the minimization of the number of nonzero column segments in the lower triangular part of the resulting partition.

For the second phase, we propose an intelligent in-block row reordering method with the aim of decreasing the computational costs of both forming the coefficient matrix of the reduced system once and solving the reduced system at each iteration.

The rest of the paper is organized as follows. Section 2 provides the background information on hypergraph and sparse matrix partitioning, and stSpike. In section 3, we discuss the dmpGS algorithm along with its communication and computational costs. The proposed partitioning and reordering model for dmpGS is introduced in section 4. We provide the experimental results in section 5 and conclude in section 6.

2. Background.
2.1. Hypergraph Partitioning. A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ consists of a set of vertices $\mathcal{V} = \{v_i\}_{1 \leq i \leq n}$ and a set of nets $\mathcal{N} = \{n_j\}_{1 \leq j \leq m}$. Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which is referred to as the pins of $n_j$, and denoted by $Pins(n_j, \mathcal{H})$. Each vertex $v_i$ is assigned a weight $w(v_i)$ and each net $n_j$ is assigned a cost $\text{cost}(n_j)$. $\Pi = \{V_1, V_2, \ldots, V_k\}$ denotes a $K$-way partition of $\mathcal{H}$, where parts are mutually disjoint and exhaustive. The weight of a part is the sum of the weights of vertices in that part. For a given partition, if a net connects at least one vertex in a part, it is said to connect that part. Connectivity $\lambda(n_j)$ of net $n_j$ is the number of parts connected by $n_j$. If a net $n_j$ connects multiple parts (i.e. $\lambda(n_j) > 1$), it is called cut; and otherwise internal (i.e. $\lambda(n_j) = 1$). The set of cut nets is denoted by $\mathcal{N}_{\text{cut}}$. The cutsize of $\Pi$ is defined in various ways. Two most commonly used cutsize definitions are the cut-net and the connectivity metrics [18], which are respectively defined as

$$
\text{cs}_{\text{cutn}}(\Pi) = \sum_{n \in \mathcal{N}_{\text{cut}}} \text{cost}(n), \quad \text{and} \quad \text{cs}_{\text{conn}}(\Pi) = \sum_{n \in \mathcal{N}_{\text{cut}}} (\lambda(n) - 1)\text{cost}(n).
$$

Hypergraph partitioning (HP) is the problem of finding a $K$-way partition which minimizes the cutsize and satisfies the balance criterion $W_{\text{max}} \leq W_{\text{avg}}(1+\epsilon)$. Here, $\epsilon$ is the given maximum allowable imbalance ratio; and $W_{\text{max}}$ and $W_{\text{avg}}$ respectively denote the maximum and average part weights. HP with fixed vertices ensures to assign some preassigned vertices which are called fixed vertices to the respective parts.

The recursive bipartitioning (RB) is a widely used paradigm to obtain a $K$-way HP. It first partitions the hypergraph into two and then each part is further bipartitioned recursively until reaching the desired number of parts $K$. In order to encode the cut-net and connectivity metrics, cut-net removal and cut-net splitting methods are utilized in the RB-based HP, respectively [18].

2.2. Sparse Matrix Partitioning with HP. Several HP models and methods have been proposed and successfully utilized for obtaining matrix partitioning [8, 14, 19, 20, 25, 39, 65, 68]. Among these, the most relevant one is the column-net model [18] that represents a given sparse matrix $A$ as a hypergraph $\mathcal{H}_{CN}(A)$ in which nets and vertices respectively represent columns and rows. In this model, vertex $v_i$ is added to the pin list of net $n_j$ for each nonzero $A(i,j)$ in $A$. Throughout the paper, $r_i$ and $c_j$ respectively denote row $i$ and column $j$.

A $K$-way ordered partition $\Pi = \{V_1, V_2, \ldots, V_K\}$ of the column-net model $\mathcal{H}_{CN}(A)$ is decoded as a partial reordering of the rows of $A$ in such a way that the rows corresponding to vertices in $V_k$ are ordered before the rows corresponding to the vertices in $V_{k'}$ for $k < k'$. This is a partial reordering since the rows corresponding to the vertices in the same part can be ordered arbitrarily. Let $B^r_k$ denote the $k^{th}$ row block which contains the rows corresponding to the vertices in $V_k$. We consider a symmetric row-column reordering that yields a 2D grid structure of $A$. The submatrix consisting of the rows of $B^r_k$ and columns of $\ell^{th}$ column block $B^c_\ell$ is referred as block-$(k, \ell)$ of $A$. A column is said to connect a row block $B^r_k$ if it contains at least one nonzero in $B^r_k$. A column is called cut if it connects more than one row block. For a matrix with nonzero diagonal entries, each column connects a diagonal block and becomes a cut column if it connects at least one off-diagonal block.

In the column-net model with unit net cost, the partitioning objective using the connectivity and cut-net metrics (2.1) respectively encode the minimization of the number of nonzero column segments in off-diagonal blocks and the number of cut columns. The former partitioning objective is successfully utilized in encoding the
minimization of the row parallel SpMV operations [18].

### 2.3. Sparse Triangular Spike (stSpike) Algorithm

We describe stSpike for lower triangular systems since the algorithm for the upper triangular case is similar. Given a lower triangular linear system of equations

\[ Ly = b, \]

a DS factorization of sparse lower triangular matrix \( L \) is computed as \( L = DS \), where \( D \) is the lower block diagonal of \( L \) and \( S \) is the Spike matrix. These blocks are assumed to be obtained by matrix partitioning. Multiplying both sides of (2.2) from the left by \( D^{-1} \), we obtain a modified system

\[ Sy = g, \]

where \( g = D^{-1}b \) and \( S = D^{-1}L \). By splitting \( L = D + R \), we obtain \( S = I + G \) where \( G = D^{-1}R \), and \( R \) is the block off-diagonal part of \( L \). The sparse triangular system \( DG = R \) with multiple right hand side vectors can be solved for the block rows of \( G \) independently with perfect parallelism.

The nonzero column segments of \( R \) constitute dense column segments (called *spikes*) in the off-diagonal blocks of \( S \). The block diagonal of \( S \) is identity. Additional nonzeros (fill-in) are introduced within the off-diagonal blocks of \( S \) only in the locations below the top nonzero (having the smallest row index) for each nonzero column segment of \( R \). The submatrix consisting of rows and columns \( \mathcal{C} \) of \( S \), namely \( \hat{S} = S(\mathcal{C}, \mathcal{C}) \), constitutes an independent reduced system where \( \mathcal{C} \) is the set of nonzero columns of \( R \), i.e., cut columns of \( L \). Then the reduced system is of the form

\[ \hat{S}y = \hat{g}, \]

where \( \hat{g} = g(\mathcal{C}) \) and \( \hat{g} = y(\mathcal{C}) \), which can be solved independent from the rest of the unknowns in \( y \). After solving the reduced system, the only remaining computation for retrieving the solution of the original system is

\[ y = g - D^{-1}(\hat{R}\hat{y}), \]

which can be obtained in perfect parallelism where \( \hat{R} = R(:, \mathcal{C}) \) (in MATLAB notation).

We only partially compute \( S \) just to form \( \hat{S} \), since forming \( S \) explicitly is expensive and requires a large amount of memory. Partial computation of \( S \) constitutes the factorization phase, whereas computation of \( \hat{g} \), solving (2.4) and (2.5) constitutes the solution phase of stSpike.

An example \( L \) matrix and the corresponding \( S \) and \( \hat{S} \) matrices are shown in Figure 1. The reduced system indices \( \mathcal{C} = \{1, 3, 4, 6, 7, 9, 11\} \) are colored in red and circled. The nonzeros that constitute the reduced system are bold and colored in red. The background colors of the original nonzeros and possible fill-in are green and blue, respectively. Depending on the sparsity structure of the corresponding column and block diagonal, spikes may not fill the entire column segment. For example, nonzero \( L(4, 1) \) in block-(2,1) of \( L \) leads to the spike consisting of three nonzeros in the first column of block-(2,1) of \( S \).

### 3. Distributed-Memory Parallel GS (dmpGS) Algorithm

The pseudocode of dmpGS is given in Algorithm 3.1 for processor \( P_k \) in a \( K \)-processor system. Matrix \( A \) is assumed to be partitioned into \( K \) row blocks, where \( m_k \) denotes the
number of rows in the $k^{th}$ row block. In the algorithm, $R_k$, $D_k$ and $U_k$ respectively
denote the $k^{th}$ row block of the strictly block lower triangular, lower triangular part of
the block diagonal, and strictly upper triangular parts of $A$ as shown in Figure 2. The
number of columns in $R_k$, $D_k$ and $U_k$ are respectively $\sum_{i=1}^{k-1} m_i$, $m_k$ and $\sum_{i=k}^{N} m_i$.
$f_k, g_k, x_k, h_k, w_k$ and $z_k$ denote the local subvectors of size $m_k$ that are computed by
$P_k$. These subvectors are partitioned conformably with row-wise partitioning of $A$
as shown in Figure 2. $\hat{S}$, $\hat{x}$ and $\hat{g}$ respectively denote the $|C| \times |C|$ coefficient matrix,
$|C| \times 1$ unknown and $|C| \times 1$ right hand side vectors of the reduced system in stSpike.
$C_k$ denotes the subset of $C$ corresponding to the row indices in $R_k$.

In Algorithm 3.1, lines 2-7 denote the factorization phase of stSpike which computes $\hat{S}$. This phase is done only once after which we proceed with the GS iterations in
lines 8-22. Each dmpGS iteration involves two SpMVs at lines 11 and 20, two vector
subtraction operations at lines 12 and 22, an independent sparse triangular solve at
line 13, a reduced system solution at line 17, which enables independent sparse triangular
solves at line 21. The upper and lower triangular SpMV operations are incurred by
the GS and stSpike algorithms, respectively. These two SpMV operations incur
communication of $x$-vector entries depending on the sparsity structures of the upper
triangular $U$ and lower triangular $L$ matrices, respectively. Conformable partitioning
of the vectors avoids communication during vector subtraction operations.

At lines 9–10, communication operations are performed for local SpMV (line
11). After $P_k$ receives all necessary non-local $x$-vector entries, it forms its augmented
vector $\tilde{x}$. Each processor sends the selected entries of its $g_k$ vector to $P_1$ (line 14) to
form the right hand side vector $\tilde{g}$ (line 16) for the sequential solution of the reduced
system to obtain $\tilde{x}$ (line 17). Here $\tilde{x}$ corresponds to those unknowns in $x$ which are at
the interface of the partitioning of $L$ and obtaining them decouples the global lower
triangular system into independent much smaller systems. $P_1$ sends only those $x$-
vector entries that are required by other processors (line 18) so that each processor

![Fig. 1: Sparsity structure of $L$ and resulting $S$ and $\hat{S}$ matrices derived from stSpike.](image)

![Fig. 2: Four-way row-wise partition of matrix $A$ and vectors $x$ and $f$.](image)
Algorithm 3.1 Distributed-Memory Parallel Gauss Seidel (dmpGS) for processor \( P_k \)

**Input:** Submatrices \( R_k, D_k, U_k \), and right-hand side subvector \( f_k \)

**Output:** Subvector \( x_k \)

1: Choose an initial guess for \( x_k \)
2: if \( 2 \leq k \leq K - 1 \) then
3: \( G_k \leftarrow D_k^{-1} R_k \) \( \triangleright \) local partial sparse triangular solve with multiple RHS
4: Form and send \( \hat{G}_k \) to processor \( P_1 \)
5: if \( k = 1 \) then
6: Receive \( \hat{G}_\ell \) from \( P_\ell \) for \( 2 \leq \ell \leq K - 1 \) to form \( \hat{G} \)
7: \( \hat{S} \leftarrow \hat{G} + I \)
8: while not converged do
9: Send required local \( x_k \) entries to respective processors in \( \{P_1, \ldots, P_{k-1}\} \)
10: Receive non-local \( x_\ell \) entries from processors in \( \{P_{k+1}, \ldots, P_K\} \) to form \( \bar{x}_k \)
11: \( h_k \leftarrow U_k \bar{x}_k \) \( \triangleright \) local SpMV
12: \( g_k \leftarrow f_k - h_k \)
13: \( g_k \leftarrow D_k^{-1} h_k \) \( \triangleright \) local sparse triangular solve
14: if \( 2 \leq k \leq K - 1 \) then Send \( \{g_k(i)\}_{i \in C_k} \) to processor \( P_1 \)
15: if \( k = 1 \) then
16: Receive \( \{g_\ell(i)\}_{i \in C_k} \) from \( P_\ell \) for \( 2 \leq \ell \leq K - 1 \) to form \( \bar{g} \)
17: \( \tilde{x} \leftarrow \bar{S}^{-1} \bar{g} \) \( \triangleright \) solve reduced system
18: Send \( \tilde{x} \) entries to requiring processors
19: if \( k \neq 1 \) then Receive required \( \tilde{x} \)-entries to form \( \bar{x}_k \)
20: \( z_k \leftarrow R_k \bar{x}_k \) \( \triangleright \) local SpMV
21: \( w_k \leftarrow D_k^{-1} z_k \) \( \triangleright \) local sparse triangular solve
22: \( x_k \leftarrow g_k - w_k \)

\( P_k \) forms its \( \tilde{x} \) vector (line 19) to perform local SpMV (line 20).

The communication overhead in each iteration of dmpGS is as follows. The communication volume incurred by \( h = U \bar{x} \) (line 11) and \( z = R \bar{x} \) (line 20) are equal to the number of nonzero column segments in the off-diagonal blocks of \( U \) and \( L \), respectively. Thus the communication volume required by these two SpMV is equal to the total number of off-diagonal nonzero column segments in \( A \) (\text{offD.nzCol seg}(A))

This manuscript is for review purposes only.

The communication volume incurred at line 16 is equal to the size of the reduced system, \(|C|\). Therefore, the total communication volume of dmpGS is

\[
\text{commVol} = \text{offD.nzCol seg}(A) + |C|.
\]

Note that the different row blocks \( (R_k) \) seem to vary in the number of columns because of the triangular structure of the problem. On the other hand, the computational load imbalance is alleviated by the proposed partitioning model which also gathers most of the nonzeros to the diagonal blocks.

4. The Proposed Partitioning and Reordering Model. We propose a two-phase model for reducing the communication overhead of dmpGS while maintaining computational balance as well as reducing the sequential computational overhead incurred by solving the reduced system at each iteration. This computational overhead is proportional to the number of nonzeros in the off-diagonals of \( \hat{S} \). In subsection 4.1, we propose a novel HP model as the first phase which simultaneously encodes the minimization of the reduced system size \(|C|\) and the communication volume. Decreasing
\[ |C| \text{ is important not only because it directly contributes to reducing the communication volume, but it also relates to decreasing the computational overhead. In subsection 4.2, we propose an in-block reordering method as the second phase which refines the improvement further by decreasing the number of nonzeros in } \hat{S}. \text{ We provide the illustrations showing the effect of the proposed partitioning and reordering model on a sample matrix in subsection 4.3.} \]

### 4.1. Hypergraph Partitioning Model

The partitioning objective in this phase is minimizing the sum of communication volume overhead (3.1) and sequential overhead costs with proper scaling:

\[
\text{PartObj} = \text{commVol} + (\alpha - 1)|C| = (\text{offD}_{nzCol\_segs}(A) + |C|) + (\alpha - 1)|C|
\]

Here \( \alpha \) denotes the scaling factor between the effect of the reduced system size and the number of off-diagonal nonzero column segments on the overall overhead.

#### 4.1.1. Definitions and Layout

We define a column as \( L\text{-cut} \) if it connects at least one off-diagonal block in the lower triangular part. That is, a column \( c_i \) in \( k^{th} \) column block \( B_k^r \) is \( L\text{-cut} \) if it connects a row block \( B_r^k \) with \( \ell > k \). Since \( L\text{-cut} \) columns of \( A \) are the nonzero columns of \( \hat{R} \), the number of \( L\text{-cut} \) columns \( (L\text{-cut\_cols}(A)) \) is equal to the reduced system size, \( |C| \). Therefore, the partitioning objective (4.1) can be rewritten as

\[
\text{PartObj} = \text{offD}_{nzCol\_segs}(A) + \alpha(L\text{-cut\_cols}(A)).
\]

Let \( \mathcal{H}_{CN}(A) = (\mathcal{V}, \mathcal{N}) \) be the column-net hypergraph of an \( m \times m \) sparse matrix \( A \) with nonzero diagonal entries. An ordered partition \( \Pi_K = (\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K) \) of \( \mathcal{H}_{CN}(A) \) is decoded as a partial symmetric row and column reordering of \( A \) as explained in section 2.2. Each net \( n_i \) of \( \mathcal{H}_{CN}(A) \) connects vertex \( v_i \) since \( A(i, i) \neq 0 \) for each \( 1 \leq i \leq m \). A net \( n_i \) with \( v_i \in \mathcal{V}_k \) is called \( L\text{-cut} \) if it connects at least one vertex part \( \mathcal{V}_\ell \) such that \( \ell > k \). The set of \( L\text{-cut} \) nets is denoted as \( \mathcal{N}_{L\text{cut}} \). We define a new type of cutsize, which we call the \( L\text{-cut\_net metric}, \) as

\[
\text{cs}_{L\text{cut}}(\Pi_K) = \sum_{n \in \mathcal{N}_{L\text{cut}}} \text{cost}(n).
\]

Finally, the cost of partition \( \Pi_K \) is defined as the sum of connectivity metric with unit net cost and \( L\text{-cut\_net metric} \) with net cost \( \alpha \), i.e.,

\[
\text{cost}_{\text{conn} + L\text{cut}}(\Pi_K) = \sum_{n \in \mathcal{N}_{cut}} (\lambda(n) - 1) + \alpha|\mathcal{N}_{L\text{cut}}|.
\]

Here, each cut net \( n \) incurs \( \lambda(n) - 1 \), and each \( L\text{-cut} \) net incurs \( \alpha \) to the cutsize.

**Lemma 4.1.** A column \( c_i \) of \( A \) is \( L\text{-cut} \) iff net \( n_i \) of \( \mathcal{H}_{CN}(A) \) is \( L\text{-cut} \).

**Proof.** Due to symmetric row-column ordering, \( c_i \) is in \( B_k^r \) iff \( r_i \) is in \( B_r^k \), which corresponds to \( v_i \in \mathcal{V}_k \). Furthermore, \( c_i \) connects \( B_r^k \) iff \( n_i \) connects \( v_i \). Therefore, \( c_i \) in \( B_k^r \) connects \( B_r^k \) iff \( n_i \) with \( v_i \in \mathcal{V}_k \) connects \( v_i \), where \( \ell > k \).

**Proposition 4.2.** Minimizing cost\(_{\text{conn} + L\text{cut}}(\Pi_K) \) for a \( K \)-way partition \( \Pi_K \) of \( \mathcal{H}_{CN}(A) \) corresponds to minimizing the partitioning objective (4.2).
The partitioning objective at each RB step is to minimize (4.5) and is assigned to the conn-nets and cost \( \alpha \) of the \( L \)-cut-net metric. The motivation for net replication is the requirement of different net splitting and net removal procedures for encoding the connectivity and \( L \)-cut-net metrics at each RB step. In order to encapsulate the RB objective (4.5), we assign unit cost to the conn-nets and cost \( \alpha \) to the lcn-nets. We refer to the hypergraph formed by these replicated nets as \( \mathcal{H} \).

We extend \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \) into a hypergraph \( \mathcal{H}' = (\mathcal{V}', \mathcal{N}') \) so that minimizing the number of conventional cut nets in \( \mathcal{H} \) encodes minimizing (4.5). We introduce new fixed vertices \( v_U \in \mathcal{V}_U \) and \( v_L \in \mathcal{V}_L \) to form the extended vertex set \( \mathcal{V}' = \mathcal{V} \cup \{v_U, v_L\} \). We represent each lcn-net \( n_i^{\ell} \) in \( \mathcal{H} \) as a pair of nets \( \hat{n}_i^{\ell} \) and \( \tilde{n}_i^{\ell} \) in \( \mathcal{H}' \). \( \hat{n}_i^{\ell} \) is the same as \( n_i^{\ell} \) except it is \( U \)-anchored (connects \( v_U \)). \( \tilde{n}_i^{\ell} \) is a 2-pin \( L \)-anchored net which connects \( v_L \) and \( v_i \). That is, for each net \( n_i \) in \( \mathcal{H}_{CN}(A) \), \( \mathcal{H}' \) contains nets \( n_i^{\ell}, \hat{n}_i^{\ell}, \tilde{n}_i^{\ell} \), where

\[
\text{Pins}(n_i^{\ell}, \mathcal{H}') = \text{Pins}(n_i, \mathcal{H}_{CN}(A)),
\]

\[
\text{Pins}(\hat{n}_i^{\ell}, \mathcal{H}') = \text{Pins}(n_i, \mathcal{H}_{CN}(A)) \cup \{v_U\} \quad \text{and}
\]

\[
\text{Pins}(\tilde{n}_i^{\ell}, \mathcal{H}') = \{v_i, v_L\}.
\]

The nets in the extended hypergraph for a sample 3-pin net are shown in Figure 3.

We form \( \mathcal{H}' \) at the beginning and apply RB steps until reaching the desired part count, \( K \). The resulting \( K \)-way partition \( \Pi_K \) of \( \mathcal{H}' \) induces a \( K \)-way partition \( \Pi_K \) of \( \mathcal{H}_{CN}(A) \). \( \mathcal{H} \) is an in-between hypergraph introduced for the sake of clarity of presentation and is not constructed during implementation. We explain the proposed net splitting and removal methods on \( \mathcal{H} \) and show the correspondence on \( \mathcal{H}' \). We consider that each bipartition \( \Pi'_2 = (\mathcal{V}_U', \mathcal{V}_L') \) of \( \mathcal{H}' \) induces a bipartition \( \Pi_2 = (\mathcal{V}_U, \mathcal{V}_L) \)
Fig. 3: Net $n_i$ in $\mathcal{H}_{CN}(A)$ is replicated as conn-net $n_i^c$ and lcn-net $n_i^l$ to form $\mathcal{H}$. Net $n_i^l$ in $\mathcal{H}$ is represented by a pair of nets $\hat{n}_i^l$ and $\check{n}_i^l$ in $\mathcal{H}'$.
If \( n^i_L \) is not \( L \)-cut and \( v_i \in \mathcal{V}_L \) in \( \Pi_2 \), then \( \hat{n}^i_L \) is cut in \( \Pi'_2 \) because it connects \( v_U \in \mathcal{V}'_U \) and \( v_i \in \mathcal{V}'_L \); but \( \check{n}^i_L \) is not cut since both \( v_i \) and \( v_L \) are in \( \mathcal{V}'_L \).

Proposition 4.4. Minimizing the conventional cut-net metric for the bipartition \( \Pi'_2 \) of \( \mathcal{H}' \) encodes minimizing cost \( \text{cost}_{RB}(\Pi_2) \) defined in (4.5).

Proof. By Lemma 4.3, each \( L \)-cut net in \( \Pi_2 \) incurs 2 cut nets in \( \Pi'_2 \), whereas all remaining nets in \( \Pi_2 \) incur 1 cut net in \( \Pi'_2 \). Since the cost of lcn-nets is \( \alpha \), the cutsize incurred by lcn-nets in \( \Pi'_2 \) is \( \alpha(|N_{Lcut}| + |\mathcal{N}|) \). Since conn-nets are of unit cost, they incur \( |N_{cut}| \) to the cutsize of \( \Pi'_2 \). Hence the total cutsize of \( \Pi'_2 \) is \( |N_{cut}| + \alpha|N_{Lcut}| + \alpha|\mathcal{N}| \). Since \( \alpha|\mathcal{N}| \) is constant, minimizing the cutsize of \( \Pi'_2 \) is equivalent to minimizing \( |N_{cut}| + \alpha|N_{Lcut}| \), which is \( \text{cost}_{RB}(\Pi_2) \).

Figure 5 shows an example 2-level RB in terms of lcn-nets in \( \mathcal{H} \) and the corresponding 4-way matrix partitioning. The \( L \)-cut nets \( n^1_L \), \( n^2_L \) and \( n^6_L \) and the corresponding \( L \)-cut columns \( c_1 \), \( c_2 \), and \( c_6 \) of \( A \) are colored in red background. \( n^2_L \) is \( L \)-cut in the first level RB and discarded in the future bipartitions. This is because column \( c_2 \) is already counted as \( L \)-cut due to nonzero \( A(6,2) \) and should not be counted as \( L \)-cut again due to nonzero \( A(4,2) \) in further bipartitions.

Note that the \( L \)-cut net definition can be considered to be similar to the left-cut net defined in [1] for encapsulating the profile minimization, but the net splitting and

This manuscript is for review purposes only.
Theorem 4.5. Recursively bipartitioning $\mathcal{H}'$ by minimizing the cutsize according to the cut-net metric and applying the proposed net splitting and removal strategies until reaching $K'$ parts encode minimizing the partitioning objective (4.2).

Proof. By Proposition 4.4, recursively bipartitioning $\mathcal{H}'$ by minimizing the conventional cut-net metric encodes minimizing $\text{cost}_{RB}(\Pi_2)$ at each RB step. We show that this encodes minimizing $\text{cost}_{\text{conn}+\text{cut}}(\Pi_K)$. Proposed net splitting and removal strategies ensure that an $L$-cut net in $\Pi_K$ is also $L$-cut in $\Pi_2$ in exactly one RB step. Since an $L$-cut net contributes $\alpha$ to both $\text{cost}_{RB}(\Pi_2)$ and $\text{cost}_{\text{conn}+\text{cut}}(\Pi_K)$, minimizing $\alpha|\mathcal{N}_{\text{cut}}|$ in each partition $\Pi$ encodes minimizing $\alpha|\mathcal{N}_{\text{cut}}|$ in $\Pi_K$. Furthermore, minimizing the number of cut nets $|\mathcal{N}_{\text{cut}}|$ at each RB step and applying the cut-net splitting procedure encodes minimizing the connectivity metric $\sum_{n \in \mathcal{N}_{\text{cut}}}(\lambda(n) - 1)$ [18]. Therefore, minimizing the cutsize for each partition $\Pi'_k$ of $\mathcal{H}'$ hence by Proposition 4.2, this corresponds to the partitioning objective (4.2).  

4.2. Reordering within Row Blocks. Consider the $K$-way block structure (e.g. Figure 2) of $A$ induced by the partial symmetric row-column permutation obtained by the HP model (section 4.1). We perform row reordering within the $k$th row block of $A$ by considering nonzeros of the $k$th row block $R_k$ of $R$. The resulting row reordering within the $k$th row block of $A$ is symmetrically applied to the columns of the $k$th column block of $A$. $R_k$ is an $m_k \times z_k$ matrix where $z_k = \sum_{i=1}^{k-1} m_i$. For simplicity, we assume a local indexing for the rows of $R_k$ so that $R_k$ consists of rows $r_i$ with $1 \leq i \leq m_k$. Recall that in stSpike, fill-in may arise below the top nonzero of each spike in $R_k$. The top nonzero of a spike $c_j$ in $R_k$ is the nonzero with the minimum row index, i.e., $\text{top}(c_j, R_k) = \min\{i : R_k(i, j) \neq 0, 1 \leq i \leq m_k\}$. We define the height of a spike $c_j$ in $R_k$ as the number of reduced system row indices between $\text{top}(c_j, R_k)$ and $m_k$ inclusively, i.e.,

$$ (4.6) \quad \text{height}(c_j, R_k) = |\{i : \text{top}(c_j, R_k) \leq i \leq m_k, i \in \mathcal{C}_k\}|, $$

since only the rows with indices in $\mathcal{C}_k$ may contribute to the nonzero count of $\widehat{S}$. The height of a spike in $R_k$ constitutes an upper bound on the nonzero count (including the fill-in) of the corresponding column in $\widehat{S}$. In Figure 1b, the heights of the spikes are as follows: $\text{height}(c_1, R_2)=3$, $\text{height}(c_3, R_2)=2$, $\text{height}(c_1, R_3)=1$, $\text{height}(c_4, R_3)=2$, $\text{height}(c_5, R_3)=1$, and $\text{height}(c_7, R_3)=2$. The height of a non-spike column is assumed

This manuscript is for review purposes only.
The total height minimization problem (THMP) is NP-hard.

**Theorem 4.6.** The total height minimization problem (THMP) is NP-hard.

**Proof.** We reduce the profile minimization problem (PMP) \([1, 46]\), which is known to be NP-hard \([26, 44]\), to THMP as follows. Given a symmetric matrix \(V \in \mathbb{R}^{n \times n}\) with nonzero diagonal entries, the objective of PMP is finding a symmetric row/column reordering \(P\) that minimizes \(\sum_{j=1}^{n} |\{c_j \mid PVPT \}|\). This minimization objective is equivalent to maximizing \(\sum_{j=1}^{n} \text{top}(c_j, PVPT)\), since \(\sum_{j=1}^{n} j = \text{constant}\). Any instance \(PVPT\) of PMP can be mapped to an instance \(PV\) of THMP by simply removing the column reordering as \((PVPT)P = PV\). Note that the minimization objective of THMP, which is \(\sum_{j=1}^{n} \text{top}(c_j, PV)\), is equivalent to maximizing \(\sum_{j=1}^{n} \text{top}(c_j, PV)\), since \(\sum_{j=1}^{n} n = \text{constant}\). Thus, \(PVPT\) is a solution of PMP if \(PV\) is a solution of THMP since the column reordering itself has no effect on \(\sum_{j=1}^{n} \text{top}(c_j, PVPT) = \sum_{j=1}^{n} \text{top}(c_j, PV)\). If there had been a polynomial-time solution to THMP, then one could solve PMP in polynomial time by just applying the row reordering obtained by THMP on the columns as well. Therefore, PMP can be reduced to THMP in polynomial time; and since PMP is NP-hard, then so is THMP.

Algorithm 4.1 presents the pseudocode of the proposed heuristic for reordering the rows of \(R_k\). The efficient implementation of this algorithm requires accessing the nonzeros of both rows and columns of \(R_k\), so it is stored both in CSR and CSC formats. \(\text{Cols}(r_i)\) denotes the set of columns in row \(r_i\), whereas \(\text{Rows}(c_j)\) denotes the set of rows in column \(c_j\). Degree of a row or column is defined as the number of nonzeros in that row or column, i.e., \(\deg(r_i) = |\text{Cols}(r_i)|\) and \(\deg(c_j) = |\text{Rows}(c_j)|\). In lines 3-5, we define the load of each row \(r_i\) as the sum of degrees of columns \(c_j\) such that \(R_k(i, j) \neq 0\).

The greedy choice utilized in the proposed heuristic is to order the rows with smaller degrees to upper positions of \(R_k\) since placing denser rows to upper positions incurs more height in (4.7). We further improve our greedy approach by using dynamic row degrees during the row selection process. When a row is selected, the degree of each unselected row is decremented by the number of its nonzeros having the same column index with the nonzeros in the selected row. Since the nonzeros in a selected row already determine the heights of the respective columns, we do not need to consider the rest of the nonzeros of these columns in future row selections. When
selecting a row among rows with the same degree, load values of the rows are used as
a tie-breaking strategy. A row with a higher load is preferred to be selected since it
will lead to a larger amount of decrease on the degrees of unselected rows.

In Algorithm 4.1, $S(d)$ denotes the set of rows with degree $d$. Due to dynamic row
degrees, at each iteration we find the minimum degree $d^*$ (line 9). Then we choose the
row $r_{i^*} \in S(d^*)$ with the maximum load (line 10). After $r_{i^*}$ is selected, all remaining
nonzeros in each column $c_j$ with $R_k(i^*, j) \neq 0$ are deleted as in lines 15-18. For each
unselected row $r_{i'}$ with $R_k(i', j) \neq 0$, we dynamically update the load and degree of
$r_{i'}$, and the respective degree sets (lines 19-22).

Recall that forming $\hat{S}$ in dmpGS requires the computation of nonzeros up to the
largest reduced system row index and any entry beyond that is not required to be
computed for each row block. Hence the total height (4.7) also gives the computational
cost of forming $\hat{S}$ since we place $\overline{R}_k$ at the top of $R_k$ for each $1 < k < K$.

4.3. Illustration. Figure 6 illustrates the effect of applying the proposed part-
titioning and reordering model for $K=8$ on a sample matrix (msc23052) from the
SuiteSparse Matrix Collection [24]. The nonzero structure of the original matrix, the
structure obtained after applying the proposed HP model and the final structure after
the proposed in-block reordering are shown in order. Below each ordering of $A$, the
resulting Spike matrix ($\hat{S}$) is shown, including the nonzeros of the reduced system ($\hat{S}$)
which are highlighted with red circles. As seen in the figure, the proposed partitioning
and reordering model significantly reduces the nonzero count of the reduced system.
PARTITIONING AND REORDERING FOR PARALLEL GS

(a) A: original
(b) A: HP ordering
(c) A: HP&in-block ordering
(d) S: original
(e) S: HP ordering
(f) S: HP&final ordering

Fig. 6: Nonzero structure of \texttt{msc23052}: (a) before ordering, (b) after HP for $K = 8$, (c) after HP and in-block reordering; (d),(e),(f) the respective Spike (S) matrices (the reduced system ($\hat{S}$) nonzeros are circled in red color).

For example, the number of nonzeros in $\hat{S} - I$ in Figures 6d, 6e, and 6f are 277,113, 3,593, and 811, respectively. Note that these numbers may seem to be much larger than the ones appearing in the figures because of the overlapping red circles.

Notice that the proposed HP model gathers most of the nonzeros to the diagonal blocks so that the off-diagonal blocks become very sparse. Then, the proposed in-block reordering method gathers the reduced-system nonzeros to the upper left corner of the respective off-diagonal block (Figure 6f). This is because we agglomerate the reduced system row indices to the top within each block, and we apply the resulting row reordering to the columns symmetrically. Within each off-diagonal block, gathering the rows with reduced-system indices to the top corresponds to agglomerating the columns with these indices, which are actually all the columns having nonzeros, to the left. An exception is the first column block since no row reordering is performed for the first row block.

5. Experiments. We use the HSL software package MC64 [29] for scaling and permuting the coefficient matrices to avoid a singular $L$. We select the MC64 option that maximizes the product of the diagonal entries and then scales to make the absolute value of diagonal entries one and the off-diagonal entries less than or equal to one. For symmetric matrices, in order not to destroy the symmetry, we apply the symmetric MC64 if the main diagonal is already zero-free. Otherwise, we apply the nonsymmetric MC64 to obtain a zero-free main diagonal. For unsymmetric matrices, we just apply the nonsymmetric MC64.
Table 1: Number of instances among different matrix kinds in the dataset

<table>
<thead>
<tr>
<th>Kind ID</th>
<th>Kind Name</th>
<th>Sym</th>
<th>Unsym</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>structural</td>
<td>48</td>
<td>4</td>
<td>52</td>
</tr>
<tr>
<td>2</td>
<td>circuit simulation</td>
<td>2</td>
<td>46</td>
<td>48</td>
</tr>
<tr>
<td>3</td>
<td>economic</td>
<td>1</td>
<td>33</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>semiconductor device</td>
<td>0</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>5</td>
<td>computational fluid dynamics</td>
<td>6</td>
<td>27</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>2D/3D</td>
<td>19</td>
<td>9</td>
<td>28</td>
</tr>
<tr>
<td>7</td>
<td>power network</td>
<td>14</td>
<td>13</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>optimization</td>
<td>20</td>
<td>3</td>
<td>23</td>
</tr>
<tr>
<td>9</td>
<td>model reduction</td>
<td>13</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>chemical process simulation</td>
<td>0</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>theoretical/quantum chemistry</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>electromagnetics</td>
<td>6</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>thermal</td>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>materials</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>15</td>
<td>weighted graph</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>16</td>
<td>acoustics, oceanography, counter-ex., analytics</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>All</td>
<td></td>
<td>155</td>
<td>204</td>
<td>359</td>
</tr>
</tbody>
</table>

The experiments are conducted on an extensive dataset obtained from the SuiteSparse Matrix Collection [24]. For sufficiently coarse-grained parallel processing, we select real square matrices that have more than 20,000 rows and between 100,000 and 20,000,000 nonzeros. There are 199 symmetric and 208 unsymmetric matrices in SuiteSparse satisfying these properties at the time of experimentation. 44 symmetric and 4 unsymmetric matrices are eliminated because they are singular. The remaining are 155 symmetric and 204 unsymmetric, a total of 359 sparse matrices on which we conduct experiments. Table 1 shows the number of instances for each matrix kind. Kinds are sorted in decreasing order of instance count. The kinds having less than 5 instances in our dataset (acoustics, chemical oceanography, counter-example and data analytics) are grouped as one kind.

5.1. Partitioning Quality. We tested the performance of the proposed partitioning algorithm described in subsection 4.1 against the partitioning quality of the conventional column-net HP with connectivity metric (cnHP) and graph partitioning (GP) models. For both cnHP and GP, vertex weights are set as the number of nonzeros in the respective rows whereas nets and edges are assigned unit cost. In cnHP, the objective is to minimize the number of nonzero off-diagonal column segments. In GP, the objective is to minimize the number of nonzeros in the off-diagonal blocks. For unsymmetric matrices, GP is applied on $|A| + |A^T|$. The well-known partitioning tools METIS [40] and PaToH [19] are used for GP and cnHP models, respectively.

In the proposed model, we use PaToH as the HP tool in each bipartitioning step. Experiments are conducted with different scaling factors $\alpha = 1, 2, 5$ and 10 for len-net cost assignment. We set the maximum allowable imbalance ratio in each bipartitioning as $\epsilon = 0.05$. As both METIS and PaToH involve randomized algorithms in the coarsening phase, five partitioning runs are performed for each instance with different seeds and the averages are reported. We conduct experiments for $K = 8, 16, 32, 64, 128$ and 256 parts (processors).

Table 2 shows the results of the comparison experiments in terms of the communication volume and the reduced system size metrics for dmpGS utilizing the partitions generated by GP, cnHP and the proposed model. For each test instance, these metrics are normalized with respect to the number of rows and the average for all matrices are given for each $K$. Here and hereafter, all averages are given as geometric means.

As seen in Table 2, cnHP achieves considerably low communication volume and reduced system size than GP as expected. The average improvement of cnHP over
Table 2: Averages of total communication volume and the reduced system size in dmpGS, both normalized with respect to the number of rows.

<table>
<thead>
<tr>
<th>$K$</th>
<th>Comm. vol.</th>
<th>Red. sys. size</th>
<th>proposed HP model (Sec. 4.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\alpha = 1$</td>
<td>$\alpha = 2$</td>
<td>$\alpha = 5$</td>
</tr>
<tr>
<td>8</td>
<td>0.158</td>
<td>0.139</td>
<td>0.139</td>
</tr>
<tr>
<td>16</td>
<td>0.253</td>
<td>0.224</td>
<td>0.224</td>
</tr>
<tr>
<td>32</td>
<td>0.380</td>
<td>0.332</td>
<td>0.332</td>
</tr>
<tr>
<td>64</td>
<td>0.547</td>
<td>0.479</td>
<td>0.479</td>
</tr>
<tr>
<td>128</td>
<td>0.767</td>
<td>0.680</td>
<td>0.697</td>
</tr>
<tr>
<td>256</td>
<td>1.062</td>
<td>0.955</td>
<td>0.977</td>
</tr>
</tbody>
</table>

GP is approximately 10% for both metrics on $K = 256$. In fact, cnHP is equivalent to the proposed HP model for $\alpha = 0$. As seen in the table, there is a trade-off between the reduced system size and the communication volume for varying values of $\alpha$ for the proposed HP model. Yet the rate of increase in the communication volume is observed to be larger than the rate of decrease in the reduced system size with increasing $\alpha$. For example, for $K = 64$, compared to the cnHP model, the proposed model slightly increases the communication volume by 0.4%, 0.5%, 2.9% and 5.9% whereas it significantly decreases the reduced system size by 21.5%, 25.2%, 30.7% and 32.0% for $\alpha = 1, 2, 5$ and 10, respectively. Here, $\alpha = 2$ seems to be a balanced choice since it significantly decreases the reduced system size while it slightly increases the communication volume. This is reflected in the parallel scalability of the proposed algorithm as will be shown in subsection 5.3, thus we set $\alpha = 2$ in the upcoming results.

In Figure 7, we provide the performance profiles comparing GP, cnHP and the proposed model in terms of the reduced system size. We present the performance profiles only for $K = 16, 64$ and 256 due to lack of space. A performance profile [27] shows the comparison of different models relative to the best performing one for each data instance. On a profile, a point $(x, y)$ means that the respective model is within $x$ factor of the best result for a fraction $y$ of the instances. For example, the point (1.20, 0.60) on the curve of cnHP means that cnHP yields 20% more reduced system size than the smallest reduced system size achieved for 60% of the dataset. Therefore, the model closest to the top left corner is interpreted as the model with best performance.

As seen in Figure 7, the proposed model outperforms the baseline algorithms in terms of the reduced system size in the majority of the test instances. As $K$ increases, the performance gap between GP and cnHP decreases, whereas the performance gap
Table 3: Total height and nonzero count averages in the off-diagonal blocks of $\hat{S}$.

<table>
<thead>
<tr>
<th>kind ID</th>
<th>K = 8</th>
<th>K = 16</th>
<th>K = 32</th>
<th>K = 64</th>
<th>K = 128</th>
<th>K = 256</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>height</td>
<td>nnz</td>
<td>height</td>
<td>nnz</td>
<td>height</td>
<td>nnz</td>
</tr>
<tr>
<td>1</td>
<td>1,470.1</td>
<td>54.1</td>
<td>283.6</td>
<td>263.6</td>
<td>145.9</td>
<td>67.6</td>
</tr>
<tr>
<td>2</td>
<td>71.9</td>
<td>35.1</td>
<td>100.6</td>
<td>61.6</td>
<td>15.8</td>
<td>35.0</td>
</tr>
<tr>
<td>3</td>
<td>1,219.2</td>
<td>31.4</td>
<td>321.2</td>
<td>271.5</td>
<td>167.8</td>
<td>152.7</td>
</tr>
<tr>
<td>4</td>
<td>27.7</td>
<td>3.8</td>
<td>61.6</td>
<td>15.8</td>
<td>6.2</td>
<td>4.5</td>
</tr>
<tr>
<td>5</td>
<td>260.0</td>
<td>10.0</td>
<td>142.6</td>
<td>9.1</td>
<td>90.3</td>
<td>7.6</td>
</tr>
<tr>
<td>6</td>
<td>600.0</td>
<td>123.2</td>
<td>298.3</td>
<td>101.5</td>
<td>148.6</td>
<td>61.3</td>
</tr>
<tr>
<td>7</td>
<td>131.8</td>
<td>10.1</td>
<td>7.4</td>
<td>36.7</td>
<td>5.7</td>
<td>4.7</td>
</tr>
<tr>
<td>8</td>
<td>513.5</td>
<td>97.3</td>
<td>59.4</td>
<td>92.2</td>
<td>48.9</td>
<td>4.5</td>
</tr>
<tr>
<td>9</td>
<td>1,547.9</td>
<td>1,101.3</td>
<td>331.4</td>
<td>321.2</td>
<td>90.3</td>
<td>7.6</td>
</tr>
<tr>
<td>10</td>
<td>29.0</td>
<td>4.8</td>
<td>109.1</td>
<td>10.9</td>
<td>15.0</td>
<td>5.6</td>
</tr>
<tr>
<td>11</td>
<td>375.3</td>
<td>619.3</td>
<td>213.3</td>
<td>112.8</td>
<td>68.8</td>
<td>89.0</td>
</tr>
<tr>
<td>12</td>
<td>241.2</td>
<td>170.7</td>
<td>121.4</td>
<td>31.4</td>
<td>6.2</td>
<td>4.5</td>
</tr>
<tr>
<td>13</td>
<td>18.2</td>
<td>2.7</td>
<td>17.7</td>
<td>3.1</td>
<td>27.4</td>
<td>2.6</td>
</tr>
<tr>
<td>14</td>
<td>217.7</td>
<td>231.1</td>
<td>116.5</td>
<td>146.9</td>
<td>33.0</td>
<td>54.4</td>
</tr>
<tr>
<td>15</td>
<td>610.4</td>
<td>228.9</td>
<td>279.9</td>
<td>164.6</td>
<td>61.8</td>
<td>50.9</td>
</tr>
<tr>
<td>16</td>
<td>15.8</td>
<td>62.2</td>
<td>8.5</td>
<td>31.5</td>
<td>4.4</td>
<td>11.9</td>
</tr>
<tr>
<td>All</td>
<td>238.1</td>
<td>57.2</td>
<td>127.1</td>
<td>43.0</td>
<td>39.0</td>
<td>18.7</td>
</tr>
</tbody>
</table>

*The values are the ratios of the results attained by the baseline over the proposed in-block reordering.

The proposed model yields the best performance for 69%, 71%, 75%, 82%, 85% and 86% of the dataset for K = 8, 16, 32, 64, 128, and 256, respectively.

The proposed HP model yields very sparse off-diagonal blocks. The number of nonzeros in any lower off-diagonal block $R_k$ is at most 0.51%, 0.44%, 0.35%, 0.26%, 0.19%, and 0.13% of the total nonzero count of $A$ for K = 8, 16, 32, 64, 128, and 256 parts on the average, respectively. As the HP model maintains balance on the nonzero counts of the whole row blocks, these low nonzero counts in off-diagonal blocks do not disturb the computational load balance among processors considerably.

### 5.2. In-Block Reordering Quality

To our knowledge, no in-block reordering method has been proposed or tested for stSpike in the literature. Therefore, we compare the improvement gained by applying the proposed in-block ordering method against a baseline algorithm which does not apply an in-block reordering. In this comparison, both the proposed and the baseline reordering methods utilize the partitions obtained by the HP model (Section 4.1). Two quality metrics used in this comparison are total height and nonzero count in the off-diagonal blocks of $\hat{S}$.

Table 3 shows the ratios of these quality metrics of the in-block reorderings generated by the baseline to those of the proposed method. For each K value, the results are given as averages grouped by different matrix kinds, and the last row shows the average of all instances in the dataset.

As seen in Table 3, the proposed reordering method achieves significant improvement in terms of both quality metrics against the baseline reordering. For example for K = 64, on overall average, the proposed method achieves 39× and 18.7× improvement against the baseline ordering in terms of height and nonzero counts, respectively. The improvement rate attained in height does not always directly reflect to the improvement rate in the nonzero counts since height is an upper bound for fill-in and the fill-in also depends on the sparsity of the diagonal blocks.

Although the improvement of the proposed reordering against the baseline ordering tends to degrade with increasing K, this is expected since there are fewer rows per block and there is less room for improvement. For example on overall average, the proposed in-block reordering method achieves 57.2×, 43.0×, 27.8×, 18.7×, 12.1× and 8.3× decrease in the nonzero count for K = 8, 16, 32, 64, 128 and 256, respectively.

The proposed partitioning and reordering model yields very small reduced systems.
Table 4: The properties of matrices to run dmpGS.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Kind ID</th>
<th>Sym</th>
<th>Size</th>
<th>Nnz</th>
<th>Relative Residual*</th>
<th>mtGS* time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>msdoor</td>
<td>1</td>
<td>✓</td>
<td>415,863</td>
<td>19,173,163</td>
<td>1.9 × 10^{-4}</td>
<td>23.1</td>
</tr>
<tr>
<td>af_shell1</td>
<td>1</td>
<td>✓</td>
<td>504,855</td>
<td>17,562,051</td>
<td>8.2 × 10^{-4}</td>
<td>23.4</td>
</tr>
<tr>
<td>af_1_k101</td>
<td>1</td>
<td>✓</td>
<td>503,625</td>
<td>17,550,675</td>
<td>1.1 × 10^{-4}</td>
<td>23.4</td>
</tr>
<tr>
<td>CoupCons3D</td>
<td>1</td>
<td>✓</td>
<td>416,800</td>
<td>17,277,420</td>
<td>4.0 × 10^{-4}</td>
<td>23.1</td>
</tr>
<tr>
<td>circuit5M_dc</td>
<td>2</td>
<td>✓</td>
<td>3,523,317</td>
<td>14,865,409</td>
<td>1.9 × 10^{-12}</td>
<td>72.7</td>
</tr>
<tr>
<td>CurlCurl_3</td>
<td>9</td>
<td>✓</td>
<td>1,219,574</td>
<td>13,544,618</td>
<td>2.8 × 10^{-4}</td>
<td>35.4</td>
</tr>
<tr>
<td>memchip</td>
<td>2</td>
<td>✓</td>
<td>2,707,524</td>
<td>13,343,948</td>
<td>5.4 × 10^{-5}</td>
<td>57.5</td>
</tr>
<tr>
<td>BenElechi1</td>
<td>1</td>
<td>✓</td>
<td>245,874</td>
<td>13,150,496</td>
<td>6.5 × 10^{-4}</td>
<td>15.2</td>
</tr>
<tr>
<td>pwtk</td>
<td>1</td>
<td>✓</td>
<td>217,918</td>
<td>11,524,432</td>
<td>1.5 × 10^{-4}</td>
<td>13.6</td>
</tr>
<tr>
<td>bmw3_2</td>
<td>1</td>
<td>✓</td>
<td>227,362</td>
<td>11,288,630</td>
<td>1.9 × 10^{-4}</td>
<td>13.6</td>
</tr>
<tr>
<td>bmwera_1</td>
<td>1</td>
<td>✓</td>
<td>148,770</td>
<td>10,641,602</td>
<td>6.0 × 10^{-4}</td>
<td>11.9</td>
</tr>
</tbody>
</table>

*Relative residual and runtime results of mtGS on 40 cores for 500 iterations.

whose nonzero counts are significantly low relative to the original system. The average ratios of the nonzero count of the reduced system over the nonzero count of the original coefficient matrix, i.e. \( \text{nnz}(\hat{S})/\text{nnz}(A) \), are 0.05%, 0.12%, 0.26%, 0.49%, 0.87%, and 1.48% for \( K = 8, 16, 32, 64, 128, \) and 256 parts, respectively. These low nonzero counts of the reduced systems verify the effectiveness of the proposed partitioning and reordering model in terms of alleviating the sequential computational overhead of dmpGS.

5.3. Parallel Scalability. Parallel experiments are performed on the Sariyer cluster of UHEM [66] using up to 320 cores over 8 distributed nodes, each containing 40 cores (two Intel Xeon Gold 6148 CPUs) and 192GB memory. The nodes are connected by an InfiniBand EDR 100 Gbps network.

We implement an MPI+OpenMP hybrid parallel dmpGS to demonstrate the effectiveness of using stSpike and the proposed model. Throughout this section, the proposed model refers to the proposed partitioning and in-block reordering model (Section 4) applied to dmpGS. The number of MPI processes is the same as the number of parts (\( K \)) in a partition. For dmpGS, we experimented with different configurations of number of processes and threads. We found that the best configuration is 8 processes per node and 5 threads per process. Therefore, we conduct parallel experiments for dmpGS using 1, 2, 4 and 8 nodes corresponding to 40, 80, 160 and 320 cores and \( K = 8, 16, 32, 64, 128, \) and 256 parts, respectively.

To the best of our knowledge, there is no publicly available true distributed-memory parallel GS implementation. For comparing the performance of dmpGS, we also implemented a multi-threaded GS (\( \text{mtGS} \)) by using the multithreaded sparse triangular system solver (\( \text{mkl} \_\text{sparse} \_d \_\text{trsm} \)) and sparse matrix vector multiplicator (\( \text{mkl} \_\text{sparse} \_d \_\text{mv} \)) of Intel MKL [37]. As a baseline, we obtain the results of mtGS on 40 threads/cores (1 node) by using the GP reordering since it is shown in [22] that the triangular solution with MKL benefits most from GP.

We tested the parallel scalability of dmpGS for a subset of the dataset since we have limited core hours on the HPC platform. From the dataset, we considered the matrices with at least 100,000 rows and 10,000,000 nonzeros, for which GS converges with a relative residual of less than \( 10^{-3} \) in 500 iterations with initial guess \( x = [0, \ldots, 0]^T \) and right-hand side vector \( f = [1/m, 2/m, \ldots, 1]^T \). Then we select only those instances with different sparsity structures from each matrix group. There were exactly 12 such matrices in our dataset satisfying these criteria. The properties of those matrices are shown in Table 4, sorted in decreasing order of nonzero counts.

The sixth and the last column respectively show the relative residual and runtime of
mtGS after 500 iterations.

Table 5 shows the average speedup values obtained by dmpGS with GP, cnHP and the proposed model over mtGS. We run dmpGS with the proposed model for \(\alpha = 1, 2, 5\) and 10 to observe the effect of scaling factor (\(\alpha\)) on the parallel performance. As seen in the table, the proposed model achieves significantly higher speedup for dmpGS over the baseline models for all \(\alpha\). The speedup performance gap between the proposed and baseline models increase with increasing \(K\), thus confirming the effectiveness of the proposed model.

We also provide Figure 8 which depicts the performance profiles for comparing the dmpGS runtime using the proposed model for varying \(\alpha\) and \(K\) values. We choose \(\alpha = 2\) for better scalability of dmpGS since it yields the best performance for larger part counts (\(K = 32\) and 64) as seen in both Table 5 and Figure 8. As seen in Table 5, the proposed model with \(\alpha = 2\) yields average of 1.5\(\times\), 1.9\(\times\), 2.7\(\times\) and 3.2\(\times\) higher speedup relative to the best of the baseline models for \(K = 8, 16, 32\) and 64, respectively.

Figure 9 shows the results of the strong scaling experiments as speedup curves of dmpGS with GP, cnHP and the proposed model. The proposed model significantly enhances the scalability of dmpGS so that dmpGS scales up to 320 cores on all instances. As seen in the figure, the proposed model outperforms GP and cnHP models for all of the test instances, significantly so in 9 out of 12. In Figure 9 for memchip, dmpGS using the proposed model achieves up to 122.2 speedup on 320 cores over mtGS on 40 cores.

### Table 5: Average speedup obtained by dmpGS over mtGS on 40 cores.

<table>
<thead>
<tr>
<th>(K)</th>
<th>number of nodes</th>
<th>number of cores</th>
<th>GP</th>
<th>cnHP</th>
<th>proposed model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(\alpha = 1)</td>
<td>(\alpha = 2)</td>
<td>(\alpha = 5)</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>40</td>
<td>9.87</td>
<td>8.71</td>
<td><strong>14.85</strong></td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>80</td>
<td>14.65</td>
<td>13.58</td>
<td><strong>29.07</strong></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>160</td>
<td>17.41</td>
<td>16.11</td>
<td><strong>47.28</strong></td>
</tr>
<tr>
<td>64</td>
<td>8</td>
<td>320</td>
<td>15.79</td>
<td>17.60</td>
<td><strong>54.96</strong></td>
</tr>
</tbody>
</table>

*The best speedup value obtained for each \(K\) is shown in bold.*

### 6. Conclusion.

We proposed and implemented an stSpike-based distributed-memory parallel GS (dmpGS) algorithm. For improving the scalability of dmpGS, we hypothesize hypergraph partitioning (HP) based partitioning model and an in-block row reordering method. Extensive experiments show that the proposed HP model significantly decreases the reduced system size with respect to the baseline models while attaining comparable communication volume. The proposed in-block reordering method leads to a substantial decrease in the computational cost of both forming and solving the reduced system. Parallel experiments up to 320 cores demonstrate that using the proposed reordering model significantly improves the scalability of dmpGS.

![Fig. 8: Performance profiles in terms of the dmpGS runtime using the proposed model.](image-url)
As a future work, we will consider the parallel solution of the reduced system to further alleviate the sequential bottleneck. We will also consider an in-block row reordering which takes the nonzeros of the diagonal blocks into account for further reducing the nonzero count in the reduced system. Finally, the future work will include extending the dmpGS algorithm for multiple right-hand-side vectors as it is very common in modern applications. Using multiple right-hand-side vectors is expected to further enhance the performance of dmpGS since it enables using higher level BLAS subroutines compared to the single right-hand-side case. Moreover, the parallel solution time per right-hand-side vector will further decrease since the parallel factorization is done only once.
REFERENCES


This manuscript is for review purposes only.
PARTITIONING AND REORDERING FOR PARALLEL GS


[54] E. Polizzi and A. Sameh, A parallel hybrid banded system solver: the SPIKE algorithm,


This manuscript is for review purposes only.