Reordering sparse matrices into block-diagonal column-overlapped form

Seher Acer a, Cevdet Aykanat b,∗

a Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, USA
b Computer Engineering Department, Bilkent University, Ankara, Turkey

ABSTRACT

Many scientific and engineering applications necessitate computing the minimum norm solution of a sparse underdetermined system of equations. The minimum 2-norm solution of such systems can be obtained by a recent parallel algorithm, whose numerical effectiveness and parallel scalability are validated in both shared- and distributed-memory architectures. This parallel algorithm assumes the coefficient matrix in a block-diagonal column-overlapped (BDCO) form, which is a variant of the block-diagonal form where the successive diagonal blocks may overlap along their columns. The total overlap size of the BDCO form is an important metric in the performance of the subject parallel algorithm since it determines the size of the reduced system, solution of which is a bottleneck operation in the parallel algorithm. In this work, we propose a hypergraph partitioning model for reordering sparse matrices into BDCO form with the objective of minimizing the total overlap size and the constraint of maintaining balance on the number of nonzeros of the diagonal blocks. Our model makes use of existing partitioning tools that support fixed vertices in the recursive bipartitioning paradigm. Experimental results validate the use of our model as it achieves small overlap size and balanced diagonal blocks.

1. Introduction

Many scientific and engineering applications [6,12,19,22,25,27] require computing the minimum norm solution of an underdetermined system of equations of the form

Ax = f, (1)

where A is an m × n sparse matrix and m < n [18]. A common approach for obtaining the minimum 2-norm solution of an underdetermined linear least squares problem is the use of a QR factorization in a direct method. Various packages such as SuiteSparseQR [13,14], HSL MA49 [2], and qr.mumps [7] provide efficient parallel and sequential implementations for the general sparse QR algorithm.

One recent approach [24] for obtaining the minimum 2-norm solution of an underdetermined linear least squares problem is the use of the Balance scheme [17,21,23], which is an effective and efficient parallel algorithm originally proposed for an ill-conditioned banded linear system of equations. This parallel algorithm [24] can also be considered as an extension of the general sparse QR factorization to distributed-memory systems for obtaining the minimum 2-norm solution. In this parallel algorithm, the coefficient matrix is assumed to be in block-diagonal column-overlapped (BDCO) form with K diagonal blocks, which is shown in Fig. 1. As seen in the figure, the kth diagonal block E_k of the BDCO form is given by

E_k = [C_k A_k B_k].

for k = 2, 3, . . . , K − 1, whereas the first and last diagonal blocks E_1 and E_K are respectively given by

E_1 = [A_1 B_1] and E_K = [C_K A_K].

Here, successive diagonal blocks E_{k−1} and E_k overlap along the columns of their submatrices B_{k−1} and C_k. Columns of the overlapping submatrices are referred to as coupling columns.

The parallel algorithm [24] exploits the BDCO form so that each of K processors independently solves a small linear least squares problem of the form

E_k z_k = f_k, (2)
where

\[ z_1 = \begin{bmatrix} x_1 \\ e_1 \end{bmatrix}, \quad z_k = \begin{bmatrix} \hat{e}_{k-1} \\ x_k \end{bmatrix} \quad \text{for } k = 2, 3, \ldots, K - 1, \text{ and} \]

\[ z_K = \begin{bmatrix} \hat{e}_{K-1} \\ x_K \end{bmatrix}. \]

This step is performed in parallel without any communication. For \( x \) to be a solution, \( e_k \) and \( \hat{e}_k \) should be equal for \( k = 1, 2, \ldots, K - 1 \). This is ensured by a sequential step in which a small system \( PAQ \) is solved. The size of the reduced system is determined by the total overlap size, i.e., the total number of coupling columns. When compared to the state-of-the-art parallel QR implementations, this approach is reported to achieve better scalability on both shared- and distributed-memory architectures [24].

In this paper, our aim is to find two permutations \( P \) and \( Q \) such that \( PAQ \) is in a \( K \)-way BDCO form. We refer to this permutation problem as the BDCO problem. In the BDCO problem, the objective is to minimize the total overlap size in \( PAQ \) since the performance of the above-mentioned parallel algorithm considerably deteriorates with increasing total overlap size, as reported in [24]. The constraint of the BDCO problem is to maintain balance on the number of nonzeros in the diagonal blocks of \( PAQ \) to ensure balanced workload for processors.

To the best of our knowledge, literature lacks an algorithm for solving the BDCO problem. However, there are efforts for solving problems that resemble the BDCO problem. In [5] and [26], the problems of reordering sparse matrices into singly-bordered block diagonal (SBBD) form and separated block diagonal (SBD) form were addressed, respectively. In the SBBD form, all coupling rows are reordered to a border at the bottom or all coupling columns are reordered to a border on the right. In the SBD form, coupling rows are reordered in between two parts as they are encountered during the recursive bipartitioning steps. The SBBD and SBD forms differ from the BDCO form as they allow coupling rows/columns to span more than two diagonal blocks, which are not necessarily consecutive. In [1], the problem of reordering sparse square matrices into block diagonal form with overlap (BDO form) was addressed. The differences between the BDCO and BDO forms are two-fold: (i) the BDO form is obtained on structurally symmetric matrices via symmetric row/column permutation, whereas the BDCO form is obtained on non-square matrices via row and column permutations that are different from each other, (ii) consecutive diagonal blocks overlap along both rows and columns in the BDO form, whereas they overlap along only columns in the BDCO form.

In order to solve the BDCO problem, we propose the \( HP_{BDCO} \) algorithm, which utilizes the column-net hypergraph model [8] of the given coefficient matrix \( A \). First, we investigate the feasibility of the \( K \)-way BDCO permutation for matrix \( A \) by considering the vertex-to-vertex adjacency topology of the column-net hypergraph of \( A \). If the respective permutation is feasible, then the corresponding hypergraph is partitioned by the proposed \( HP_{BDCO} \) algorithm so that the partitioning objective corresponds to minimizing the total overlap size, whereas the partitioning constraint corresponds to maintaining balance on the number of nonzeros of the diagonal blocks. The \( HP_{BDCO} \) algorithm utilizes fixed vertices within the recursive bipartitioning paradigm in order to ensure that only the successive diagonal blocks may share columns. The proposed algorithm is flexible in the sense that any hypergraph partitioning tool that supports fixed vertices can be used in the implementation. The effectiveness of the proposed \( HP_{BDCO} \) algorithm is validated by reordering a wide range of matrices into BDO form with small overlap size and balanced diagonal blocks as well as by running the parallel code [24] on the reordered matrices.

The rest of the paper is organized as follows. Section 2 gives background on hypergraph partitioning. The proposed \( HP_{BDCO} \) algorithm is given in Section 3. Section 4 gives experimental results and Section 5 concludes.

2. Background

2.1. Hypergraphs

A hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \) is defined as a set \( \mathcal{V} \) of vertices and a set \( \mathcal{N} \) of nets. Each net \( n \in \mathcal{N} \) connects a subset of vertices in \( \mathcal{V} \), which is denoted by \( \text{Pins}(n) \). We use phrase “pins of \( n \)” to refer to the vertices connected by net \( n \). In a dual manner, each vertex \( v \in \mathcal{V} \) is connected by a subset of nets in \( \mathcal{N} \), which is denoted by \( \text{Nets}(v) \).

We adapt the following graph terminology to hypergraphs. In a hypergraph, two vertices are said to be adjacent if they are connected by at least one common net. Let \( \text{Adj}(v) \) denote the set of vertices adjacent to vertex \( v \). Then,

\[ \text{Adj}(v) = \{ u : \text{Nets}(v) \cap \text{Nets}(u) \neq \emptyset \} - \{ v \}. \]

In a hypergraph, a path is defined as a sequence \( (v_1, v_2, \ldots, v_p) \) of vertices such that each two consecutive vertices are adjacent. The distance between vertices \( v_i \) and \( v_j \), which is denoted by \( d(v_i, v_j) \), is defined as the number of vertex-to-vertex hops in a shortest path from \( v_i \) to \( v_j \). For example, if \( (v_i, v_j, v_k) \) is a shortest path, then the distance between \( v_i \) and \( v_j \) is equal to 2, i.e., \( d(v_i, v_j) = 2 \). Trivially, \( d(v, v) = 0 \). The distance between a vertex \( u \) and a set of vertices \( \mathcal{V} \) is defined as the minimum of distances between \( u \) and vertices in \( \mathcal{V} \), that is, \( d(u, \mathcal{V}) = \min_{v \in \mathcal{V}} d(u, v) \).

The diameter of hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \) is then defined as the maximum distance between two vertices, that is,

\[ \text{diameter}(\mathcal{H}) = \max_{v_i, v_j \in \mathcal{V}} d(v_i, v_j). \]

For a hypergraph with diameter \( D \), a vertex \( v \) is said to be a peripheral vertex if there exists a vertex \( u \) such that \( d(v, u) = D \).
2.2. Hypergraph partitioning (HP) problem

In a given K-way partition $\Pi = \{V_1, V_2, \ldots, V_k\}$ of hypergraph $H = (V, N)$, net $n$ is said to connect part $V_i$ if it connects at least one vertex in $V_i$, i.e., $\Pi(n) \cap V_i \neq \emptyset$. Let $A(n)$ denote the set of the parts connected by $n$, that is, $A(n) = \{V_i : \Pi(n) \cap V_i \neq \emptyset\}$. The number of parts connected by $n$ is denoted by $\lambda(n)$, hence, $\lambda(n) = |A(n)|$. A net $n$ is said to be cut if it connects more than one part, i.e., $\lambda(n) > 1$. Otherwise, $n$ is said to be uncut. If the part that is connected by uncut net $n$ is $V_i$, then $n$ is said to be internal to $V_i$. A vertex $v$ is said to be boundary if it is connected by at least one cut net.

In $H$, each vertex $v \in V$ is assigned a non-negative weight denoted by $w(v)$, whereas each net is assigned a non-negative cost denoted by $c(n)$. Then, the cutsize of $\Pi$ is defined as the sum of the costs of the cut nets in $\Pi$, that is,
\[
\text{cutsize}(\Pi) = \sum_{n, \lambda(n) > 1} c(n).
\] (3)

Part weight $W_i$ of $V_i$ is defined as the sum of the weights of the vertices in $V_i$, i.e., $W_i = \sum_{v \in V_i} w(v)$.

For given $K$ and $\epsilon$ values and a hypergraph $H$, the HP problem is defined as finding a partition $\Pi = \{V_1, V_2, \ldots, V_k\}$ of $H$ with the objective of minimizing the cutsize (3) and the constraint of maintaining the balance on the part weights, which is given by $W_k < W_{\text{avg}}(1 + \epsilon)$ for $k = 1, 2, \ldots, K$. (4)

Here, $W_{\text{avg}}$ and $\epsilon$ denote the average part weight in $\Pi$ and a predetermined threshold for maximum allowable imbalance ratio, respectively.

The HP problem with fixed vertices is a variant of the HP problem which has been successfully used for solving the repartitioning/renumbering problem in the context of parallelizing irregular applications [3,4,9,11]. In this problem, vertices in $F_1 \subseteq V$ are pre-assigned to part $V_1$ in a K-way partition so that $F_k \subseteq V_k$ for $k = 1, 2, \ldots, K$ after the partitioning. A vertex $v$ is referred to as “fixed to $V_1$” if $v \in F_1$. Vertices can be fixed to at most one part, that is, $F_k \cap F_1 = \emptyset$ holds for any fixed-vertex set pair $(F_k, F_1)$. A vertex $v$ is referred to as “free” if it is not fixed, i.e., $v \in V - \bigcup_{k=1}^{K} F_k$.

2.3. Recursive bipartitioning (RB) paradigm

RB is a commonly-used paradigm for obtaining K-way hypergraph partitioning. In RB, a given hypergraph is recursively bipartitioned, i.e., partitioned into two parts, until K parts are obtained. At each RB step, a bipartition $\Pi = \{V_1, V_2\}$ of the current hypergraph $H$ is obtained and two new hypergraphs $H_1$ and $H_2$ are formed using vertex parts $V_1$ and $V_2$. This procedure forms a hypothetical full binary tree in which each node represents a hypergraph formed/bipartitioned. The vertex set of each leaf hypergraph in the RB tree corresponds to a part in the resulting K-way partition $\Pi = \{V_1, V_2, \ldots, V_k\}$.

3. Proposed model

Suppose that we are given an $m \times n$ sparse matrix $A$ and an integer $K > 1$. For reordering $A$ into K-way BDCO form, we propose a hypergraph partitioning algorithm, HP\textsubscript{BDCO}, which utilizes the column-net hypergraph model [8]. Let $H = (V, N)$ denote the column-net hypergraph of $A$ with vertex and net sets $V = \{v_1, v_2, \ldots, v_m\}$ and $N = \{n_1, n_2, \ldots, n_n\}$.

Vertex $v_i$ represents row $i$ of $A$ for $i = 1, 2, \ldots, m$, whereas net $n_j$ represents column $j$ of $A$ for $j = 1, 2, \ldots, n$. Net $n_j$ connects vertices that represent the rows with a nonzero entry in column $j$, that is,
\[
\text{Pins}(n_j) = \{v_i : a_{ij} \neq 0\}.
\]

Each net $n_j$ is assigned a unit cost, i.e., $c(n_j) = 1$, whereas each vertex $v_i$ is assigned a weight equal to the number of nonzeros in row $i$ of $A$, i.e., $w(v_i) = |\{j : a_{ij} \neq 0\}|$.

Consider a K-way partition $\Pi = \{V_1, V_2, \ldots, V_k\}$ of $H$. In Section 3.1, we introduce a new partitioning constraint, which we refer to as the BDCO constraint, for $\Pi$ to be utilized for reordering $A$ into K-way BDCO form. We refer to $\Pi$ as a BDCO partition if it satisfies the BDCO constraint. In Section 3.2, we investigate the feasibility of a K-way BDCO partition for hypergraph $H$. Finally in Section 3.3, we propose the RB-based HP\textsubscript{BDCO} algorithm which obtains a K-way BDCO partition of a given hypergraph $H$, if it is feasible.

3.1. BDCO constraint

We define the BDCO constraint as restricting each cut net in $\Pi$ to connect two consecutive parts. That is, for each $n \in N$ with $\lambda(n) > 1$, $A(n) = \{V_k, V_{k+1}\}$ for some $k \in \{1, 2, \ldots, K - 1\}$.

A K-way partition that satisfies the BDCO constraint, a K-way BDCO partition, induces partial row and column orderings for permuting $A$ into K-way BDCO form as follows. The rows that are represented by the vertices in $V_k$ are ordered after the rows that are represented by the vertices in $V_{k-1}$. The columns that are represented by the internal nets of $V_k$ are ordered after the columns that are represented by the cut nets connecting $V_k$ and $V_{k-1}$ and before the columns that are represented by the cut nets connecting $V_k$ and $V_{k+1}$. In a dual manner, the columns that are represented by the cut nets connecting $V_{k+1}$ and $V_k$ are ordered after the columns that are represented by the internal nets of $V_{k+1}$ and before the columns that are represented by the internal nets of $V_k$.

Internal orderings of the above-mentioned row/column groups can be arbitrary. Consequently, the vertices and internal nets of $V_k$ respectively correspond to the rows and columns of $A_k$ in Fig. 1, whereas the cut nets connecting $V_k$ and $V_{k+1}$ correspond to the coupling columns shared by diagonal blocks $E_k$ and $E_{k+1}$ in Fig. 1.

Fig. 2 displays a sample $10 \times 12$ sparse matrix $A$, a 4-way BDCO partition $\Pi$ of the column-net hypergraph of $A$, and matrix $A_{\text{BDCO}}$ in a 4-way BDCO form which is obtained via decoding $\Pi$ as described above. For example, consider vertex part $V_1 = \{v_2, v_5\}$ in $\Pi$ and nets connecting $V_3$, namely, $n_1, n_2, n_3, n_{11},$ and $n_{12}$. Then, diagonal block $E_3 = [C_3 A_3 B_3]$ in $A_{\text{BDCO}}$ consists of rows 5 and 2 and columns 1, 6, 8, 11, and 12, where $C_3$ contains column 1, $A_3$ contains columns 6 and 8, and $B_3$ contains columns 11 and 12. Note that column 1 is a coupling column and it is shared by $E_2$ and $E_3$ in $A_{\text{BDCO}}$ since cut net $n_1$ connects both $V_2$ and $V_3$ in $\Pi$. Similarly, columns 11 and 12 are coupling columns and they are shared by $E_2$ and $E_4$ since cut nets $n_{11}$ and $n_{12}$ connect both $V_3$ and $V_4$.

The BDCO constraint ensures that each coupling column is shared by two consecutive diagonal blocks in the resulting BDCO form. Consider a column $j$ in $A$ and the corresponding net $n_j$ in $H$. Note that the rows that have at least one nonzero in column $j$ are those represented by the vertices in Pins($n_j$), by the construction of the column-net hypergraph model. First, assume that $n_j$ is internal to part $V_k$ for some $k$, that is, Pins($n_j$) $\subseteq V_k$. Since the vertices in $V_k$ correspond to the rows of $A_k$ (or equivalently, $E_k$), each row that contains at least one nonzero in column $j$ is a row of $A_k$. Hence, column $j$ only links diagonal block $E_k$. Now, assume that $n_j$ is a cut net, that is, $n_j$ connects parts $V_k$ and $V_{k+1}$ for some $k$. Then, the vertices in Pins($n_j$) are either in $V_k$ or in $V_{k+1}$. Then, each row that contains at least one nonzero in column $j$ is either a row of $A_k$ or a row of $A_{k+1}$. Hence, column $j$ links consecutive diagonal blocks $E_k$ and $E_{k+1}$.
that \( \Pi \) is a \( K \)-way BDCO partition of a given hypergraph \( \mathcal{H} \). Note that the BDCO constraint implies that the adjacent vertices are assigned either to the same part or to the consecutive parts. In other words, no two adjacent vertices can be assigned to parts that have one or more parts in between. This statement relates the maximum number of parts that one can have at a BDCO partition of \( \mathcal{H} \) to the length of the longest shortest path in \( \mathcal{H} \), i.e., the diameter of \( \mathcal{H} \). In this section, we investigate the relation between the existence of a \( K \)-way BDCO partition of \( \mathcal{H} \) and the diameter of \( \mathcal{H} \).

Assume that \( v_i \) and \( v_j \) are two arbitrary vertices in \( \mathcal{H} \) such that \( d(v_i, v_j) = \delta \). Consider a \( K \)-way BDCO partition \( \Pi \) of \( \mathcal{H} \). Assume that \( v_i \) and \( v_j \) are assigned to the \( l \)th part \( V_l \) and the \( J \)th part \( V_J \) in \( \Pi \), respectively. We first claim that \( |J - l| \leq \delta \), that is, the parts containing \( v_i \) and \( v_j \) can have at most \( \delta - 1 \) parts in between. Assume to the contrary that \( |J - l| = \delta' > \delta \) and consider the case \( l < J \). Then, there exists a path \( \{v_i, v_{i_1}, \ldots, v_{i_{\delta'}+1} \} \) with end vertices \( v_i = v_l \) and \( v_{i_{\delta'}} = v_l \) such that \( v_{i_k} \in V_{i+k} \) for \( k = 0, 1, \ldots, \delta' \). This path is a shortest path since no two vertices \( v_{i+k} \) and \( v_{i+k'} \) in this path can be adjacent if \( |k - k'| > 1 \) by the BDCO constraint. Then \( d(v_i, v_{i_{\delta'}}) > \delta \), which is a contradiction. This implies that for a hypergraph \( \mathcal{H} \) with diameter \( D \), at most \( D - 1 \) parts can exist between any two parts in a BDCO partition. Consequently, a \( K \)-way BDCO partition of \( \mathcal{H} \) is not feasible if the diameter of \( \mathcal{H} \) is less than \( K - 1 \).

### 3.3. HP\textsubscript{BDCO} algorithm

The HP\textsubscript{BDCO} algorithm utilizes the RB paradigm to obtain a \( K \)-way BDCO partition of a given hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \). Before invoking the HP\textsubscript{BDCO} algorithm, we first find a pseudo-peripheral vertex \( u \) in \( \mathcal{H} \) by utilizing the algorithm in [16], which is originally proposed for finding pseudo-peripheral vertices in graphs. This algorithm easily extends to hypergraphs thanks to the definition of adjacent vertices in hypergraphs, which is introduced in Section 2.1. After a pseudo-peripheral vertex \( u \) is obtained, a vertex \( v \) which is one of the furthest vertices from \( u \) is found. If the distance between \( u \) and \( v \) is less than \( K - 1 \), we conclude that a \( K \)-way BDCO partition of \( \mathcal{H} \) is not feasible by the discussion given in Section 3.2. Otherwise, the recursive HP\textsubscript{BDCO} algorithm is invoked to obtain a \( K \)-way BDCO partition of \( \mathcal{H} \).

The basic outline of the recursive step of the HP\textsubscript{BDCO} algorithm is given as follows:

1. Obtain a bipartition \( \{V_1, V_2\} \) of hypergraph \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \).
2. Construct two hypergraphs \( \mathcal{H}_1 \) and \( \mathcal{H}_2 \) respectively on the vertex sets \( V_1 \) and \( V_2 \).
3. Invoke the HP\textsubscript{BDCO} algorithm on \( \mathcal{H}_1 \), which returns a \( 2 \)-way BDCO partition \( \Pi_1 = \{V_1, V_2\} \) of \( \mathcal{H}_1 \).
4. Invoke the HP\textsubscript{BDCO} algorithm on \( \mathcal{H}_2 \), which returns a \( 2 \)-way BDCO partition \( \Pi_2 = \{V_1', V_2'\} \) of \( \mathcal{H}_2 \).
5. Concatenate \( \Pi_1 \) and \( \Pi_2 \) to obtain a \( K \)-way BDCO partition \( \Pi \) of \( \mathcal{H} \) and return \( \Pi \).

This basic outline does not involve the details that are essential to satisfying the constraint and feasibility of a BDCO partition. In the following paragraphs, we construct the proposed BDCO algorithm on top of this basic outline by stressing the points where this basic outline fails and describing techniques to resolve them.

One problem with the given outline is that \( \Pi_1 \) and \( \Pi_2 \) being BDCO partitions does not automatically guarantee that \( \Pi \) too is a BDCO partition. In order for \( \Pi \) to satisfy the BDCO requirement, the cross-partition adjacencies between \( \Pi_1 \) and \( \Pi_2 \) should be confined to be between the last part of \( \Pi_1 \) and the first part of \( \Pi_2 \).
In other words, the vertices in $V_1$ that are adjacent to vertices in $V_2$ should be assigned to $\mathcal{V}_d$ in $\Pi_1$. In a dual manner, the vertices in $V_2$ that are adjacent to vertices in $V_1$ should be assigned to $\mathcal{V}_d$ in $\Pi_2$. The HBDCC algorithm confines the cross-partition adjacencies to be between $\mathcal{V}_d$ in $\Pi_1$ and $\mathcal{V}_d$ in $\Pi_2$ by fixing the end vertices of these adjacencies to respective parts before each bipartitioning. How we achieve this will be clear after we introduce the vertex fixation scheme in the proposed algorithm.

Another problem with the given outline is that the feasibility of BDCO partitions $\Pi_1$ and $\Pi_2$ is not guaranteed. That is, depending on the bipartition $\{V_1, V_2\}$ of $H$, the diameters of hypergraphs $H_1$ or $H_2$ may be less than $K/2 - 1$ even though the diameter of $H$ is at least $K - 1$. Assume that two pseudo-peripheral vertices $u$ and $v$ in $H$ are given, where the distance between $u$ and $v$ is at least $K - 1$, i.e., $d(u, v) \geq K - 1$. The HPBDCC algorithm ensures that the diameters of both $H_1$ and $H_2$ are at least $K/2 - 1$ by

- fixing vertex $u'$ to $V_1$ for each vertex $u' \in V$ with $d(u', u) < K/2$ and
- fixing vertex $v'$ to $V_2$ for each vertex $v' \in V$ with $d(v', v) < K/2$.

Note that the vertices that are fixed to $V_1$ and the vertices that are fixed to $V_2$ are disjoint due to $d(u, v) \geq K - 1$. Having the above-mentioned fixed vertices during the bipartitioning, $H_1$ and $H_2$ are guaranteed to have vertex pairs $(u, u')$ and $(v, v')$ with $d(u, u') \geq K/2 - 1$ and $d(v, v') \geq K/2 - 1$, respectively. Therefore, by the described vertex fixation scheme, the existence of BDCO partitions $\Pi_1$ and $\Pi_2$ is guaranteed by the discussion given in Section 3.2.

Algorithm 1 displays the proposed recursive HPBDCC algorithm, which we build upon the basic outline by adding a vertex fixation scheme to resolve the two problems mentioned above. As input, it takes a hypergraph $H = (V, E)$, the desired number of parts $K$, and two vertex subsets $P_1 \subseteq V$ and $P_2 \subseteq V$ that are respectively referred to as the left and the right boundary vertex sets of $H$. At the initial invocation of this recursive algorithm, the input boundary vertex sets $P_1$ and $P_2$ are respectively set to $\{v\}$ and $\{u\}$, where $v$ and $u$ are two pseudo-peripheral vertices of $H$ with $d(v, u) = \text{diameter}(H)$.

In Algorithm 1, the first step is to perform the proposed vertex fixation scheme (lines 1–2). We obtain sets of vertices fixed to $V_1$ and $V_2$, i.e., $F_1$ and $F_2$, by invoking the FIX function on $H$ and boundary vertex sets $P_1$ and $P_2$, respectively. Algorithm 2 displays the basic steps of the FIX function. This algorithm performs multi-source BFS sourced at a given boundary vertex set $P$ in order to find vertices in $H$ whose distances to $P$ are smaller than $D = K/2$. In other words, for each vertex $v \in V$ with $d(v, P_1) < K/2$, the first invocation of FIX includes $v$ in $F_1$, whereas for each vertex $v \in V$ with $d(v, P_2) < K/2$, the second invocation of FIX includes $v$ in $F_2$. Note that $P_1 \subseteq F_1$ and $P_2 \subseteq F_2$, that is, the vertices in the left and right boundary vertex sets are always fixed to the left and right parts, respectively. This ensures that the cross-partition adjacencies between $\Pi_1$ and $\Pi_2$ are confined to be between the last part of $\Pi_1$ and the first part of $\Pi_2$, which is required for maintaining the BDCO structure throughout the overall RB process.

After fixing vertices, Algorithm 1 obtains a bipartition $\{V_1, V_2\}$ of hypergraph $H$ with fixed vertices assigned to their respective parts, i.e., $F_1 \subseteq V_1$ and $F_2 \subseteq V_2$ (line 3). Here, any hypergraph partitioning tool that supports fixed vertices can be utilized. If $K$ is equal to 2, which is the base case, then bipartition $\Pi = \{V_1, V_2\}$ is returned (lines 5 and 22). Note that any bipartition is trivially a 2-way BDCO partition. If $K$ is larger than 2, the recursive step of the HPBDCC algorithm is performed (lines 6–21).

In the recursive step of Algorithm 1, new hypergraphs $H_1 = (V_1, X_1)$ and $H_2 = (V_2, X_2)$ are formed on vertex sets $V_1$ and $V_2$ and the HPBDCC algorithm is invoked on $H_1$ and $H_2$ to obtain $K/2$-way BDCO partitions $\Pi_1$ and $\Pi_2$, respectively. Net sets $N_1$ and $N_2$ are obtained via cut-net removal scheme which is described as follows. They are both initialized as empty sets (lines 7–8) and each internal net of $V_1$ is included in $N_1$, whereas each internal net of $V_2$ is included in $N_2$ (lines 9–13). Note that none of the net sets include cut nets. Also note that the pin set of each net included in $N_1$ and $N_2$ remains unchanged.

Before performing the recursive invocations on newly formed hypergraphs $H_1$ and $H_2$, Algorithm 1 forms boundary vertex sets for $\Pi_1$ and $\Pi_2$, which are computed using cut-net removal technique (lines 14–19). Each boundary vertex set $\Pi_1$ that is connected by at least one cut net in $\Pi$ is included in $\Pi_1$. In a dual manner, each boundary vertex in $V_2$ that is connected by at least one cut net in $\Pi$ is included in $\Pi_2$. Note that except for the boundary vertex sets $P_1 = \{v\}$ and $P_2 = \{u\}$ of the initial invocation, all vertices in boundary vertex sets are indeed boundary vertices of some bipartition obtained during the overall partitioning procedure.

After obtaining the left and right boundary vertex sets for both $H_1$ and $H_2$, the recursive invocations of the HPBDCC algorithm on $H_1$ and $H_2$ are performed to obtain $K/2$-way BDCO partitions $\Pi_1 = (V_1, \ldots, V_{K/2})$ and $\Pi_2 = (V_1', \ldots, V_{K/2}')$ (lines 19–20). These BDCO partitions are then concatenated to obtain a $K$-way

---

**Algorithm 1 HPBDCC**

**Require:** Hypergraph $H = (V, E)$, integer $K$, boundary vertex sets $P_1$ and $P_2$

▷ Fix vertices

1. $F_1 \leftarrow \text{FIX}(H, P_1, K/2)$ \quad \triangleright fix some vertices to $V_1$

2. $F_2 \leftarrow \text{FIX}(H, P_2, K/2)$ \quad \triangleright fix some vertices to $V_2$

▷ Bipartition $H$ with fixed vertices

3. $(V_1, V_2) \leftarrow \text{PARTITION}(H, F_1, F_2, 2)$

4. if $K = 2$ then

5. $\Pi \leftarrow (V_1, V_2)$ \quad \triangleright base case

6. else

7. $\Pi_1 \leftarrow (V_1, N_1 = \emptyset)$

8. $\Pi_2 \leftarrow (V_2, N_2 = \emptyset)$

▷ Form $N_1$ and $N_2$ using cut-net removal technique

9. foreach $n \in N$ do

10. if $n$ is internal to $V_1$ then

11. $N_1 \leftarrow N_1 \cup \{n\}$

12. else if $n$ is internal to $V_2$ then

13. $N_2 \leftarrow N_2 \cup \{n\}$

▷ Form boundary vertex sets of $\Pi_1$ and $\Pi_2$

14. $\Pi_1 \leftarrow P_1, P_{22} \leftarrow P_2$ \quad \triangleright inherited boundary vertex sets

15. $\Pi_2 \leftarrow P_{21} \leftarrow \emptyset$ \quad \triangleright new boundary vertex sets

16. foreach cut-net $n \in N$ do

17. $P_{12} \leftarrow P_{12} \cup (\text{pins}(n) \cap V_1)$

18. $P_{21} \leftarrow P_{21} \cup (\text{pins}(n) \cap V_2)$

▷ Recursive invocations on $H_1$ and $H_2$

19. $\Pi_1 = (V_1^1, \ldots, V_{K/2}^1) \leftarrow \text{HPBDCC}(H_1, K/2, \Pi_{11}, \Pi_{12})$

20. $\Pi_2 = (V_1^2, \ldots, V_{K/2}^2) \leftarrow \text{HPBDCC}(H_2, K/2, \Pi_{21}, \Pi_{22})$

▷ Concatenate $\Pi_1$ and $\Pi_2$ to obtain $\Pi$

21. $\Pi \leftarrow (\Pi_1, \Pi_2)$

22. return $\Pi$
Algorithm 2 FIX

Require: \( \mathcal{H} = (\mathcal{V}, \mathcal{N}) \), boundary vertex set \( \mathcal{P} \), integer \( D \)

\( \triangleright \) Initialize fixed vertex set
1: \( \mathcal{F} \leftarrow \emptyset \)

\( \triangleright \) Initialize distance and mark arrays
2: for each vertex \( u \in \mathcal{V} \) do
3: \( \text{dist}[v] \leftarrow \infty \)
4: \( \text{mark}[v] \leftarrow \text{false} \)

\( \triangleright \) Initialize the queue for multi-source BFS
5: \( \mathcal{Q} \leftarrow \emptyset \)
6: for each vertex \( v \in \mathcal{P} \) do
7: \( \text{ENQUEUE}(\mathcal{Q}, v) \)
8: \( \text{dist}[v] \leftarrow 0 \)
9: \( \text{mark}[v] \leftarrow \text{true} \)

\( \triangleright \) Run BFS
10: while \( \mathcal{Q} \neq \emptyset \) do
11: \( v \leftarrow \text{DEQUEUE}(\mathcal{Q}) \)
12: \( \mathcal{F} \leftarrow \mathcal{F} \cup \{v\} \) \( \triangleright \) Add \( v \) to fixed vertex set
13: if \( \text{dist}[v] < D - 1 \) then
14: for each vertex \( u \in \text{Adj}(v) \) do
15: if \( \text{mark}[u] = \text{false} \) then
16: \( \text{ENQUEUE}(\mathcal{Q}, u) \)
17: \( \text{dist}[u] \leftarrow \text{dist}[v] + 1 \)
18: \( \text{mark}[u] \leftarrow \text{true} \)
19: return \( \mathcal{F} \)

\[ \text{Algorithm 2 FIX} \]

Recall that the HP\(_{\text{BDCO}}\) algorithm utilizes the cut-net removal technique while forming the net sets of the new hypergraphs at each RB step (lines 9–13 of Algorithm 1). Minimizing the cutsize at all RB steps together with utilizing the cut-net removal technique corresponds to minimizing the cutsize metric (3) of the \( K \)-way partition obtained at the end of the overall RB process [8].

Fig. 3. The proposed vertex fixation scheme illustrated on the first three levels of the RB tree.

Recall that the HP\(_{\text{BDCO}}\) algorithm is the first algorithm in the literature that addresses the BDCO problem. For evaluating the performance of HP\(_{\text{BDCO}}\), we consider a baseline method that first permutes the given matrix into a banded form and then applies block partitioning on it to obtain \( K \) diagonal blocks with column overlap. The details of this baseline algorithm, which we refer to as the RCM\(_{\text{BDCO}}\) algorithm, are described in Section 4.1.

We performed two different sets of experiments. The first set of experiments, which are described in Section 4.2, were performed on 45 semi-real sparse matrices, for which we already know nice BDCO forms exist. The second set of experiments, which are described in Section 4.3, were performed on 8 real sparse matrices, for which no prior BDCO-form knowledge exists. In both sets, we first compare RCM\(_{\text{BDCO}}\) and HP\(_{\text{BDCO}}\) in terms of total overlap size and the imbalance ratio on the number of nonzeros in diagonal blocks. Then we compare these two algorithms in terms of the runtime of the target parallel code [24] on a subset of matrices on which RCM\(_{\text{BDCO}}\) performs relatively close to HP\(_{\text{BDCO}}\) in terms of overlap size.

We used the parallel code [24] with default settings and measured the total time for solving the least squares problems in diagonal blocks, forming the reduced system and solving it, and
finally broadcasting the solution of the reduced system. We conducted the parallel experiments on up to 64 processors on a Linux workstation equipped with four 18-core CPUs (Intel Xeon Processor E7-8860 v4) and 256 GB of memory.

For the hypergraph bipartitioning in the proposed algorithm (line 3 in Algorithm 1), we used PaToH [10] with the quality setting. We used 10% as the maximum allowable imbalance ratio for the resulting K-way partition.

4.1. Baseline algorithm: RCM<sub>BDCO</sub>

In the baseline RCM<sub>BDCO</sub> algorithm, we first permute A into a banded form and then apply block partitioning on the banded matrix. Note that K-way block row partitioning on a banded matrix results in a K-way BDCO form, which is illustrated in Fig. 4.

For permuting A into a banded form, we use one of the bandwidth minimization algorithms proposed for nonsymmetric matrices in [20]. While it is cumbersome to define the bandwidth of a rectangular matrix, we expect this algorithm to place the nonzeros of A in a hopefully narrow band that starts from the top left corner and ends at the bottom right corner, as illustrated in Fig. 4.

In this algorithm, first a symmetric B matrix of size (m + n) × (m + n) is obtained using A as follows:

\[
B = \begin{bmatrix}
0 & A \\
A' & 0
\end{bmatrix}.
\]

Row i of B represents row i of A if i ≤ m, and it represents column i − m of A, otherwise. It is important to note that the standard graph representation of B corresponds to the bipartite graph representation of A. Then, the standard RCM algorithm (symrcm routine in MATLAB) is run on B for finding a row/column permutation that minimizes the bandwidth of B. This is effectively equivalent to running RCM on the bipartite graph of A.

To reorder the rows and columns of A, we consider rows/columns of B in the order provided by the above-mentioned permutation. Suppose that row/column j of B is being considered at the moment. If j represents a row of A (i.e., j ≤ m), then row j of A is ordered as the next row. If j represents a column of A (i.e., j > m), then column j − m of A is ordered as the next column. After obtaining the reordered matrix A, which is assumed to be in a banded form, we obtain K row blocks on it by simple block partitioning where we balance the number of nonzeros across the blocks.

Fig. 4. Obtaining a BDCO form on a banded matrix.

4.2. Reordering semi-real matrices into BDCO form

While forming the dataset in this section, we follow an approach similar to the one utilized for obtaining “Realistic Dataset” in [24]. In the SuiteSparse collection [15], there were a total of 38 non-square matrices in categories “least squares problem”, “computer graphics/vision problem”, and “power network problem” at the time of experimentation. To eliminate small matrices, we excluded those with the sum of the number of rows and number of columns smaller than 3000. Due to the memory limitation, we excluded the matrices with more than 1,000,000 nonzeros. For each of the remaining matrices in the dataset, we linked together 64 copies of it to obtain a semi-real matrix in 64-way BDCO form so that each diagonal block of the semi-real matrix corresponds to the original matrix. For tall and skinny matrices in the dataset, we utilized their transposes in order to have underdetermined systems of linear equations.

While obtaining a semi-real matrix by linking together 64 copies of an original matrix, we used a fixed size for the column overlap, denoted by o, between each pair of consecutive diagonal blocks. For each original matrix, we obtained five different semi-real matrices in 64-way BDCO forms obtained by utilizing overlap sizes o ∈ \{5, 10, 20, 50, 100\}. Among the obtained semi-real matrices, we excluded matrices with disconnected hypergraphs. In order to prevent the RCM<sub>BDCO</sub> and HP<sub>BDCO</sub> algorithms from being biased with the existing BDCO forms, we destroyed those BDCO forms by randomly reordering rows and columns of each semi-real matrix. The resulting dataset consists of 45 semi-real matrices derived from a total of 9 original matrices.

Table 1 displays the properties of the test matrices obtained by the above-mentioned procedure along its first four columns. In the table, matrices are displayed in increasing order of their number of nonzeros. Columns 1, 2, 3, and 4 display matrix name, number of rows, number of columns, and number of nonzeros, respectively. Note that for each original matrix M, five semi-real matrices obtained from M are displayed consecutively. The semi-real matrix obtained from M with column overlap o is denoted by M<sub>o</sub>. Note that for each matrix M, the number of rows in semi-real matrices M<sub>5</sub>, M<sub>10</sub>, M<sub>20</sub>, M<sub>50</sub>, and M<sub>100</sub> is the same as well as the number of nonzeros. On the other hand, the number of columns differs for each semi-real matrix as the overlap size also differs. Since larger overlap size implies smaller number of columns, the number of nonzeros decreases as overlap size o increases.

Table 1 displays the results obtained by the RCM<sub>BDCO</sub> and HP<sub>BDCO</sub> algorithms along columns 5–7 and 8–11, respectively. The RCM<sub>BDCO</sub> algorithm was run on each matrix M<sub>o</sub> once as described in Section 4.1. In the HP<sub>BDCO</sub> algorithm, the column-net hypergraph of matrix M<sub>o</sub> was partitioned into 64 parts and then the resulting partition was utilized to permute M<sub>o</sub> into a 64-way BDCO form. This was repeated for ten times and the average results of these ten runs are reported in columns 8–10. Columns 5 and 8 display the resulting imbalance values, which are computed as the ratio of the maximum to the average number of nonzeros in diagonal blocks minus one. Column 6 and 9 display the total overlap sizes, which are computed as the sum of the number of coupling columns in the resulting BDCO form. Column 7 and 10 display the ratio of the total overlap size to the total number of columns in the corresponding matrix. Column 11 displays the number of HP<sub>BDCO</sub> runs that resulted in an ideal BDCO permutation on the corresponding matrix in terms of total overlap size out of ten runs. For a matrix M<sub>o</sub>, we define an ideal BDCO permutation as a permutation having a total overlap size smaller than \(63 \times o \times 1.1\). That is, for \(o = 5, 10, 20, 50,\) and 100, an ideal 64-way BDCO permutation has a total overlap size smaller than 347, 693, 1386, 3465, and 6930, respectively. The table does not
have such a column for RCM_{BDCO}, because it is not able achieve any ideal BDCO permutations.

As seen in Table 1, compared to the \( \text{RCM}_{\text{BDCO}} \) algorithm, the \( \text{HP}_{\text{BDCO}} \) algorithm always obtains smaller total overlap size while usually obtaining slightly larger imbalance. On average, \( \text{RCM}_{\text{BDCO}} \) places 27.5% of the columns in overlaps, while this ratio is only 0.6% for \( \text{HP}_{\text{BDCO}} \). The overlap size obtained by \( \text{RCM}_{\text{BDCO}} \) gets closest to those obtained by \( \text{HP}_{\text{BDCO}} \) on matrices graphic_10, graphic_20, and graphic_50, still being 4.48x, 4.11x, and 2.28x worse, respectively. For \( \text{RCM}_{\text{BDCO}} \) and \( \text{HP}_{\text{BDCO}} \), imbalance ratios on the number of nonzeros in diagonal blocks are 0.2% and 1.0% on average, respectively.

As seen in Table 1, the \( \text{HP}_{\text{BDCO}} \) algorithm obtains an ideal BDCO permutation in each of the 10 runs on 18 test matrices, which correspond to the 40% of the dataset. It obtains 9, 8, 7, and 6 ideal BDCO permutations out of 10 runs on 3, 5, 2, and 1 test matrices, respectively. So, the \( \text{HP}_{\text{BDCO}} \) algorithm obtains an ideal permutation in at least 5 of the 10 runs on 29 out of 45 test matrices, which correspond to 64% of the dataset. When all runs of all test matrices are considered, the \( \text{HP}_{\text{BDCO}} \) algorithm obtains
Table 2

<table>
<thead>
<tr>
<th>Matrix</th>
<th>RCM$_{BDCO}$</th>
<th>HP$_{BDCO}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphics_10</td>
<td>1.70</td>
<td>0.87</td>
</tr>
<tr>
<td>graphics_20</td>
<td>2.54</td>
<td>0.88</td>
</tr>
<tr>
<td>graphics_50</td>
<td>3.00</td>
<td>1.16</td>
</tr>
</tbody>
</table>

Fig. 5. Matrix mri1.

an ideal permutation in 291 runs, which correspond to the 64% of the 450 runs.

Although the HP$_{BDCO}$ algorithm could not obtain any ideal permutations on some test matrices, the overlap size normalized with respect to the number of columns is still small on almost all test matrices. For example on each of matrices psa1_10, psa1_20, and Kemelmacher_100, the normalized overlap size is smaller than 0.01 although no ideal permutations are obtained on them. Note that the HP$_{BDCO}$ algorithm obtains a normalized overlap size smaller than 0.05 for each test matrix. For 38 out of 45 matrices, the normalized overlap size obtained by the HP$_{BDCO}$ algorithm is smaller than 0.05. It is smaller than 0.001 and 0.0001 for 21 and 4 matrices, respectively.

As seen in Table 1, the performance of the HP$_{BDCO}$ algorithm in total overlap size usually increases with increasing matrix size. This can be attributed to the increasing solution space of the respective partitioning problem. For example, consider the largest 15 matrices in the dataset. The HP$_{BDCO}$ algorithm almost always finds an ideal BDCO partition on these large matrices.

As seen in Table 1, the HP$_{BDCO}$ algorithm obtains an imbalance value smaller than 5% on each test matrix, except for photogrammetry_2_100 which has an imbalance value of 9.1%. Note that the HP$_{BDCO}$ algorithm obtains an imbalance value smaller than 1% on 32 matrices, which corresponds to the 71% of the 45 matrices.

Table 2 displays the runtimes of the parallel code [24] on 64 processors when the input coefficient matrix is reordered into 64-way BDCO form by RCM$_{BDCO}$ and HP$_{BDCO}$. In these experiments, we only consider the three matrices on which RCM$_{BDCO}$ performs relatively close to HP$_{BDCO}$ in terms of total overlap size. As seen in the table, the parallel runtimes obtained by RCM$_{BDCO}$ on graphics_10, graphics_20, and graphics_50 are 1.70, 2.54, and 3.00 seconds, whereas these runtimes are 0.87, 0.88, and 1.16 seconds for HP$_{BDCO}$, respectively. On these matrices, the 1.95x, 2.88x, and 2.58x speedups in parallel runtime are due to the 4.48x, 4.11x, and 2.28x reductions in total overlap size.

4.3. Reordering real matrices into BDCO form

In this section, we performed experiments on eight real matrices obtained from the SuiteSparse collection [15]. Unlike the case
in Section 4.2, we do not know if a K-way BDCO form exists on these matrices for a given K value, until we run the RCM_{BDCO} and HP_{BDCO} algorithms.

In the first part of these experiments, we used seven matrices on which the HP_{BDCO} algorithm is able to obtain a 16-way BDCO form. The properties of these matrices and the permutation results are given in Table 3. The RCM_{BDCO} algorithm, however, fails to obtain a 16-way BDCO form on most of these matrices. It fails when more than two diagonal blocks overlap along a column, which happens when the band obtained by RCM is not narrow enough. Hence, on each of these matrices, we ran the RCM_{BDCO} algorithm for K \in \{2, 4, 8, 16\} values and report the results for the largest K value where RCM does not fail to obtain a K-way BDCO form.

As seen in Table 3, the overlap size obtained by HP_{BDCO} is smaller than that obtained by HP_{BDCO} on each of the seven matrices. On watson_1 and watson_2, the total overlap sizes obtained by RCM_{BDCO} are 98,886 and 160,097 for K = 2, whereas they are only 26 and 16 for HP_{BDCO}, respectively. On lp_garges, the overlap sizes obtained by RCM_{BDCO} and HP_{BDCO} for K = 4 are 628 and 212, which correspond to 36.8% and 12.4% of all columns, respectively. On model18 and l30, RCM_{BDCO} places 53.5% and 31.0% of columns in overlaps for K = 8, while these ratios are 31.0% and 2.7% for HP_{BDCO}, respectively. Finally on lp_truss and progas, the respective ratios for K = 16 are 73.7% and 68.6% for RCM_{BDCO} and 37.7% and 38.8% for HP_{BDCO}, respectively. On these matrices, the imbalance values obtained by RCM_{BDCO} are generally smaller than those obtained by HP_{BDCO}.

In the second part of the experiments, we show how the performances of the RCM_{BDCO} and HP_{BDCO} algorithms vary on different K values. These experiments were performed for K \in \{4, 8, 16, 32, 64\} values on a real matrix, mri1, which is also obtained from the SuiteSparse collection [15]. This matrix was obtained from MRI reconstruction data and contains 65,536 rows, 147,456 columns, and 589,824 nonzeros. We removed 32,819 empty columns, so, the resulting matrix contains 114,637 columns. Fig. 5 displays the sparsity pattern of the resulting mri1 matrix. Table 4 displays the resulting imbalance values and total overlap sizes obtained by RCM_{BDCO} and HP_{BDCO} on mri1. Fig. 6 displays the 4-, 8-, 16-, and 32-way BDCO forms of matrix mri1 obtained by HP_{BDCO}.

As seen in Table 4, compared to the RCM_{BDCO} algorithm, the HP_{BDCO} algorithm obtains smaller total overlap size while obtaining slightly larger imbalance for each K value. For K being 4, 8, 16, 32, and 64, the total overlap sizes obtained by RCM_{BDCO} are 2381, 5342, 11,103, 22,485, and 45,247, whereas those obtained by HP_{BDCO} are 2174, 5066, 9648, 19,687, and 41,067, respectively. The performance degradation in the overlap size with increasing K values is expected for both algorithms as the problem gets harder. Note that the performance gap between RCM_{BDCO} and HP_{BDCO} on mri1 is small compared to the other matrices in this work. We attribute this small performance gap to the fact that the graph/hypergraph corresponding to mri1 has a more regular structure compared to the other matrices, so, RCM_{BDCO} is able to achieve a much narrower band on this matrix.

In the last part of the experiments, we ran the parallel code [24] on the real matrices. Table 5 displays the respective results. In these experiments, we considered all real matrices except for Watson_1 and Watson_2. The reason why we excluded Watson_1 and Watson_2 is because the reduced systems obtained by the RCM_{BDCO} algorithm for these matrices for K = 2 were too large to be solved by the parallel code. For each remaining matrix, we performed the parallel experiments for the largest K value for which both RCM_{BDCO} and HP_{BDCO} are able to find solutions. As seen in the table, the parallel runtimes obtained by RCM_{BDCO} on lp_ganges, model18, l30, lp_truss, progas, and mri1 are 0.11, 3.03, 9.52, 7.58, 0.10, and 100.83 seconds, respectively. The parallel runtimes obtained by HP_{BDCO} on lp_ganges, model18, l30, lp_truss, progas, and mri1 are 0.03, 1.69, 0.57, 3.78, 0.06, and 75.40, respectively. Note that the largest reduction rate obtained by HP_{BDCO} is observed on l30, where the parallel runtime reduces from 9.52 seconds to 0.57 seconds. The 16x speedup in the parallel runtime for this matrix is explained by the 12x reduction in total overlap size.

### 5. Conclusion

In this paper, we target the problem of reordering a given sparse matrix into block-diagonal column-overlapped (BDCO) form, which arises in a recent parallel algorithm proposed for obtaining the minimum norm solution of a given underdetermined system of equations. We first defined an equivalent hypergraph partitioning problem with an additional constraint tailored to the BDCO form and investigated the feasibility of its solutions.
Then we proposed a recursive-bipartitioning-based partitioning algorithm utilizing fixed vertices in order to solve this problem. The experimental results prove the effectiveness of the proposed algorithm in obtaining a BDCO form of a given sparse matrix.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References


Seher Acer received her B.S., M.S. and Ph.D. degrees in Computer Engineering from Bilkent University, Turkey. She is currently a postdoctoral researcher at Center for Computing Research, Sandia National Laboratories, Albuquerque, New Mexico, USA. Her research interests include parallel computing and combinatorial scientific computing with a focus on partitioning sparse irregular computations.

Cevdet Aykanat received the B.S. and M.S. degrees from Middle East Technical University, Ankara, Turkey, both in Electrical Engineering, and the Ph.D. degree from Ohio State University, Columbus, in Electrical and Computer Engineering. He worked at the Intel Supercomputer Systems Division, Beaverton, Oregon, as a research associate. Since 1989, he has been affiliated with Computer Engineering Department, Bilkent University, Ankara, Turkey, where he is currently a professor. His research interests mainly include parallel computing and its combinatorial aspects. He is the recipient of the 1995 Investigator Award of The Scientific and Technological Research Council of Turkey and 2007 Parlar Science Award. He has served as an Associate Editor of IEEE Transactions of Parallel and Distributed Systems between 2008 and 2012.