Journal of Parallel and Distributed Computing 140 (2020) 99–109

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

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

ARTICLE INFO

Article history:
Received 30 May 2018
Received in revised form 24 March 2019
Accepted 9 March 2020
Available online 14 March 2020

Keywords:
Underdetermined systems
Sparse matrices
Block-diagonal form
Column overlap
Hypergraph partitioning

ABSTRACT

Many scientific and engineering applications necessitate computing the minimum norm solution of a sparse underdetermined linear 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

$$Ax = f, \tag{1}$$

where A is an $m \times n$ 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 $k = 2, 3, \ldots, K-1$, whereas the first and last diagonal blocks E_1 and E_K are respectively given by

$$E_1 = [A_1\ B_1] \quad \text{and} \quad E_K = [C_K\ A_K].$$

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

$$E_k z_k = f_k, \tag{2}$$

☆ Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525. The work was done when Seher Acer was with Bilkent University.

* Corresponding author.
E-mail addresses: sacer@sandia.gov (S. Acer), aykanat@cs.bilkent.edu.tr (C. Aykanat).

https://doi.org/10.1016/j.jpdc.2020.03.002
0743-7315/

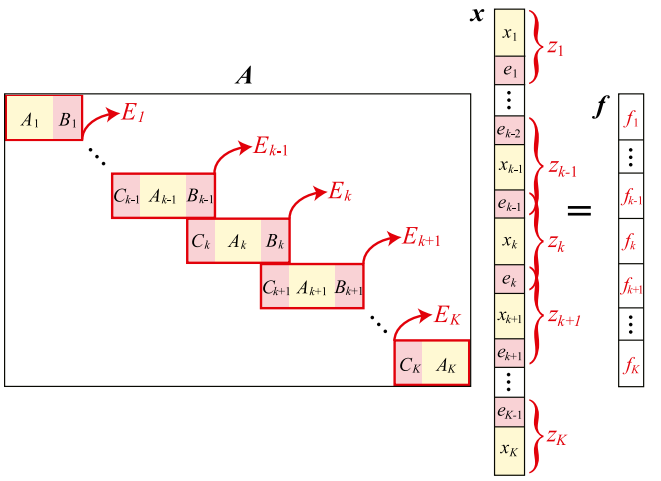

Fig. 1. A sample underdetermined system of equations $Ax = f$ where matrix A is in a K-way BDCO form.

where

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

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

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 called *reduced system* 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 BDCO 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 $Pins(n)$. 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 $Nets(v)$.

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 $Adj(v)$ denote the set of vertices adjacent to vertex v. Then,

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

In a hypergraph, a *path* is defined as a sequence $\langle v_1, v_2, \ldots, v_p \rangle$ 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 $\langle v_i, v_k, v_j \rangle$ 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,

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

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 $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, net n is said to connect part $\mathcal{V}_k$ if it connects at least one vertex in $\mathcal{V}_k$, i.e., $Pins(n) \cap \mathcal{V}_k \neq \emptyset$. Let $\Lambda(n)$ denote the set of the parts connected by n, that is, $\Lambda(n) = \{\mathcal{V}_k : Pins(n) \cap \mathcal{V}_k \neq \emptyset\}$. The number of parts connected by n is denoted by $\lambda(n)$, hence, $\lambda(n) = |\Lambda(n)|$. 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 $\mathcal{V}_k$, then n is said to be *internal to* $\mathcal{V}_k$. A vertex v is said to be *boundary* if it is connected by at least one cut net.

In $\mathcal{H}$, each vertex $v \in \mathcal{V}$ is assigned a non-negative weight denoted by $w(v)$, whereas each net is assigned a non-negative cost denoted by $c(n_j)$. Then, the *cutsize* of Π is defined as the sum of the costs of the cut nets in Π, that is,

$$cutsize(\Pi) = \sum_{n:\lambda(n)>1} c(n). \tag{3}$$

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

For given K and ϵ values and a hypergraph $\mathcal{H}$, the *HP problem* is defined as finding a partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}$ with the objective of minimizing the cutsize (3) and the constraint of maintaining the balance on the part weights, which is given by

$$W_k < W_{avg}(1+\epsilon) \quad \text{for} \quad k = 1, 2, \ldots, K. \tag{4}$$

Here, W_{avg} and ϵ denote the average part weight in Π 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/remapping problem in the context of parallelizing irregular applications [3,4,9,11]. In this problem, vertices in $\mathcal{F}_k \subseteq \mathcal{V}$ are pre-assigned to part $\mathcal{V}_k$ in a K-way partition so that $\mathcal{F}_k \subseteq \mathcal{V}_k$ for $k = 1, 2, \ldots, K$ after the partitioning. A vertex v is referred to as "fixed to $\mathcal{V}_k$" if $v \in \mathcal{F}_k$. Vertices can be fixed to at most one part, that is, $\mathcal{F}_k \cap \mathcal{F}_\ell = \emptyset$ holds for any fixed-vertex set pair $(\mathcal{F}_k, \mathcal{F}_\ell)$. A vertex v is referred to as "free" if it is not fixed, i.e., $v \in \mathcal{V} - \bigcup_{k=1}^{K} \mathcal{F}_k$.

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 $\Pi = \{\mathcal{V}_1, \mathcal{V}_2\}$ of the current hypergraph $\mathcal{H}$ is obtained and two new hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ are formed using vertex parts $\mathcal{V}_1$ and $\mathcal{V}_2$. 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 $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$.

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_{BDCO}, which utilizes the column-net hypergraph model [8]. Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ denote the column-net hypergraph of A with vertex and net sets

$$\mathcal{V} = \{v_1, v_2, \ldots, v_m\} \text{ and } \mathcal{N} = \{n_1, n_2, \ldots, n_n\}.$$

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,

$$Pins(n_j) = \{v_i : a_{ij} \neq 0\}.$$

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., $w(v_i) = |\{a_{ij} : a_{ij} \neq 0\}|$.

Consider a K-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}$. In Section 3.1, we introduce a new partitioning constraint, which we refer to as the *BDCO constraint*, for Π to be utilized for reordering A into K-way BDCO form. We refer to Π 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 $\mathcal{H}$. Finally in Section 3.3, we propose the RB-based HP_{BDCO} algorithm which obtains a K-way BDCO partition of a given hypergraph $\mathcal{H}$, if it is feasible.

3.1. BDCO constraint

We define the BDCO constraint as restricting each cut net in Π to connect two consecutive parts. That is, for each $n \in \mathcal{N}$ with $\lambda(n) > 1$, $\Lambda(n) = \{\mathcal{V}_k, \mathcal{V}_{k+1}\}$ 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 $\mathcal{V}_k$ are ordered after the rows that are represented by the vertices in $\mathcal{V}_{k-1}$. The columns that are represented by the internal nets of $\mathcal{V}_k$ are ordered after the columns that are represented by the cut nets connecting $\mathcal{V}_{k-1}$ and $\mathcal{V}_k$ and before the columns that are represented by the cut nets connecting $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$. In a dual manner, the columns that are represented by the cut nets connecting $\mathcal{V}_{k-1}$ and $\mathcal{V}_k$ are ordered after the columns that are represented by the internal nets of $\mathcal{V}_{k-1}$ and before the columns that are represented by the internal nets of $\mathcal{V}_k$. Internal orderings of the above-mentioned row/column groups can be arbitrary. Consequently, the vertices and internal nets of $\mathcal{V}_k$ respectively correspond to the rows and columns of A_k in Fig. 1, whereas the cut nets connecting $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$ correspond to the coupling columns shared by diagonal blocks E_k and E_{k+1} in Fig. 1.

Fig. 2 displays a sample 10×12 sparse matrix A, a 4-way BDCO partition Π of the column-net hypergraph of A, and matrix A_{BDCO} in a 4-way BDCO form which is obtained via decoding Π as described above. For example, consider vertex part $\mathcal{V}_3 = \{v_2, v_5\}$ in Π and nets connecting $\mathcal{V}_3$, namely, n_1, n_6, n_8, n_{11}, and n_{12}. Then, diagonal block $E_3 = [C_3\ A_3\ B_3]$ in A_{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_{BDCO} since cut net n_1 connects both $\mathcal{V}_2$ and $\mathcal{V}_3$ in Π. Similarly, columns 11 and 12 are coupling columns and they are shared by E_3 and E_4 since cut nets n_{11} and n_{12} connect both $\mathcal{V}_3$ and $\mathcal{V}_4$.

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 $\mathcal{H}$. 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 $\mathcal{V}_k$ for some k, that is, $Pins(n_j) \subseteq \mathcal{V}_k$. Since the vertices in $\mathcal{V}_k$ 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 $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$ for some k. Then, the vertices in $Pins(n_j)$ are either in $\mathcal{V}_k$ or in $\mathcal{V}_{k+1}$. 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}.

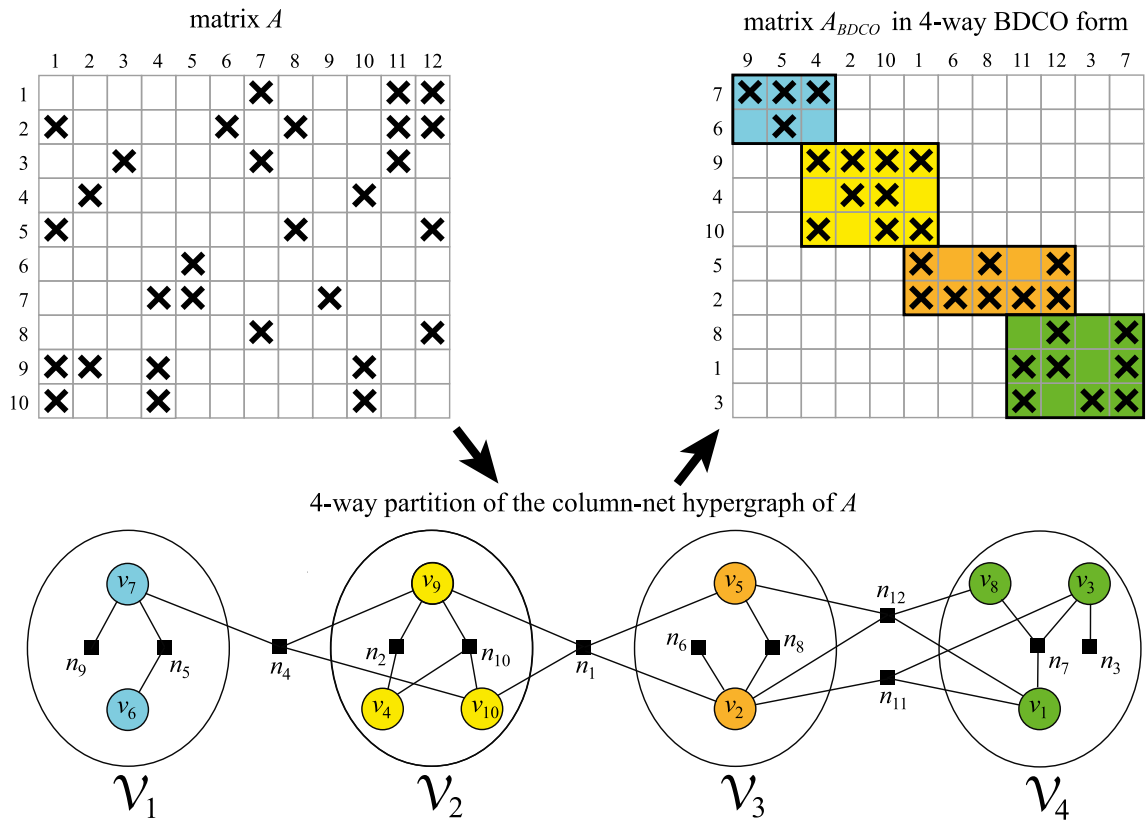

Fig. 2. A sample matrix A, a 4-way BDCO partition Π of the column-net hypergraph of A, and the permuted matrix A_{BDCO} in 4-way BDCO form, where the corresponding permutation is induced by Π.

3.2. Feasibility of a K-way BDCO partition

Consider a K-way BDCO partition Π 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 Π of $\mathcal{H}$. Assume that v_i and v_j are assigned to the Ith part $\mathcal{V}_I$ and the Jth part $\mathcal{V}_J$ in Π, respectively. We first claim that $|I-J| \leq \delta$, that is, the parts containing v_i and v_j can have at most $\delta - 1$ parts in between. Assume to the contrary that $|I-J| = \delta' > \delta$ and consider the case $I < J$. Then, there exists a path $\langle v_I, v_{I+1}, \ldots, v_{I+\delta'=J}\rangle$ with end vertices $v_i = v_I$ and $v_j = v_J$ such that $v_{I+k} \in \mathcal{V}_{I+k}$ 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 $d(v_i, v_j) > \delta$, 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_{BDCO} algorithm

The HP_{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_{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_{BDCO} algorithm is invoked to obtain a K-way BDCO partition of $\mathcal{H}$.

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

1. Obtain a bipartition $\{\mathcal{V}_1, \mathcal{V}_2\}$ of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$.
2. Construct two hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively on the vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$.
3. Invoke the HP_{BDCO} algorithm on $\mathcal{H}_1$, which returns a $K/2$-way BDCO partition $\Pi_1 = \langle \mathcal{V}_1^1, \ldots, \mathcal{V}_{K/2}^1\rangle$ of $\mathcal{H}_1$.
4. Invoke the HP_{BDCO} algorithm on $\mathcal{H}_2$, which returns a $K/2$-way BDCO partition $\Pi_2 = \langle \mathcal{V}_1^2, \ldots, \mathcal{V}_{K/2}^2\rangle$ of $\mathcal{H}_2$.
5. Concatenate Π_1 and Π_2 to obtain a K-way BDCO partition Π of $\mathcal{H}$ and return Π.

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 Π_1 and Π_2 being BDCO partitions does not automatically guarantee that Π too is a BDCO partition. In order for Π to satisfy the BDCO requirement, the cross-partition adjacencies between Π_1 and Π_2 should be confined to be between the last part of Π_1 and the first part of Π_2.

In other words, the vertices in $\mathcal{V}_1$ that are adjacent to vertices in $\mathcal{V}_2$ should be assigned to $\mathcal{V}_{K/2}^1$ in Π_1. In a dual manner, the vertices in $\mathcal{V}_2$ that are adjacent to vertices in $\mathcal{V}_1$ should be assigned to $\mathcal{V}_1^2$ in Π_2. The HP$_{\text{BDCO}}$ algorithm confines the cross-partition adjacencies to be between $\mathcal{V}_{K/2}^1$ and $\mathcal{V}_1^2$ 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 Π_1 and Π_2 is not guaranteed. That is, depending on the bipartition $\{\mathcal{V}_1, \mathcal{V}_2\}$ of $\mathcal{H}$, the diameters of hypergraphs $\mathcal{H}_1$ or $\mathcal{H}_2$ may be less than $K/2-1$ even though the diameter of $\mathcal{H}$ is at least $K-1$. Assume that two pseudo-peripheral vertices u and v in $\mathcal{H}$ are given, where the distance between u and v is at least $K-1$, i.e., $d(u,v) \geq K-1$. The HP$_{\text{BDCO}}$ algorithm ensures that the diameters of both $\mathcal{H}_1$ and $\mathcal{H}_2$ are at least $K/2-1$ by

- fixing vertex u' to $\mathcal{V}_1$ for each vertex $u' \in \mathcal{V}$ with $d(u',u) < K/2$ and
- fixing vertex v' to $\mathcal{V}_2$ for each vertex $v' \in \mathcal{V}$ with $d(v',v) < K/2$.

Note that the vertices that are fixed to $\mathcal{V}_1$ and the vertices that are fixed to $\mathcal{V}_2$ are disjoint due to $d(u,v) \geq K-1$. Having the above-mentioned fixed vertices during the bipartitioning, $\mathcal{H}_1$ and $\mathcal{H}_2$ 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 Π_1 and Π_2 is guaranteed by the discussion given in Section 3.2.

Algorithm 1 displays the proposed recursive HP$_{\text{BDCO}}$ 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 $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, the desired number of parts K, and two vertex subsets $\mathcal{P}_1 \subseteq \mathcal{V}$ and $\mathcal{P}_2 \subseteq \mathcal{V}$ that are respectively referred to as the left and the right boundary vertex sets of $\mathcal{H}$. At the initial invocation of this recursive algorithm, the input boundary vertex sets $\mathcal{P}_1$ and $\mathcal{P}_2$ are respectively set to $\{v\}$ and $\{u\}$, where v and u are two pseudo-peripheral vertices of $\mathcal{H}$ with $d(u,v) = \mathit{diameter}(\mathcal{H})$.

In Algorithm 1, the first step is to perform the proposed vertex fixation scheme (lines 1–2). We obtain sets of vertices fixed to $\mathcal{V}_1$ and $\mathcal{V}_2$, i.e., $\mathcal{F}_1$ and $\mathcal{F}_2$, by invoking the FIX function on $\mathcal{H}$ and boundary vertex sets $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Algorithm 2 displays the basic steps of the FIX function. This algorithm performs multi-source BFS sourced at a given boundary vertex set $\mathcal{P}$ in order to find vertices in $\mathcal{H}$ whose distances to $\mathcal{P}$ are smaller than $D = K/2$. In other words, for each vertex $v \in \mathcal{V}$ with $d(v,\mathcal{P}_1) < K/2$, the first invocation of FIX includes v in $\mathcal{F}_1$, whereas for each vertex $v \in \mathcal{V}$ with $d(v,\mathcal{P}_2) < K/2$, the second invocation of FIX includes v in $\mathcal{F}_2$. Note that $\mathcal{P}_1 \subseteq \mathcal{F}_1$ and $\mathcal{P}_2 \subseteq \mathcal{F}_2$, 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 Π_1 and Π_2 are confined to be between the last part of Π_1 and the first part of Π_2, which is required for maintaining the BDCO structure throughout the overall RB process.

After fixing vertices, Algorithm 1 obtains a bipartition $\{\mathcal{V}_1, \mathcal{V}_2\}$ of hypergraph $\mathcal{H}$ with fixed vertices assigned to their respective parts, i.e., $\mathcal{F}_1 \subseteq \mathcal{V}_1$ and $\mathcal{F}_2 \subseteq \mathcal{V}_2$ (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 $\Pi = \langle \mathcal{V}_1, \mathcal{V}_2 \rangle$ 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 HP$_{\text{BDCO}}$ algorithm is performed (lines 6–21).

In the recursive step of Algorithm 1, new hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1)$ and $\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2)$ are formed on vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$ and the HP$_{\text{BDCO}}$ algorithm is invoked on $\mathcal{H}_1$ and $\mathcal{H}_2$ to obtain $K/2$-way BDCO partitions Π_1 and Π_2, respectively. Net sets $\mathcal{N}_1$ and $\mathcal{N}_2$ 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 $\mathcal{V}_1$ is included in $\mathcal{N}_1$, whereas each internal net of $\mathcal{V}_2$ is included in $\mathcal{N}_2$ (lines 9–13). Note that none of the net sets include cut nets. Also note that the pin set of each net included in $\mathcal{N}_1$ and $\mathcal{N}_2$ remains unchanged.

Before performing the recursive invocations on newly formed hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, Algorithm 1 forms boundary vertex sets for $\mathcal{H}_1$ and $\mathcal{H}_2$. $\mathcal{H}_1$ inherits its left boundary vertex set $\mathcal{P}_{11}$ from $\mathcal{H}$, i.e., $\mathcal{P}_{11} = \mathcal{P}_1$. In a similar manner, $\mathcal{H}_2$ inherits its right boundary vertex set $\mathcal{P}_{22}$ from $\mathcal{H}$, i.e., $\mathcal{P}_{22} = \mathcal{P}_2$. The right boundary vertex set of $\mathcal{H}_1$ and the left boundary vertex set of $\mathcal{H}_2$, which are respectively denoted by $\mathcal{P}_{12}$ and $\mathcal{P}_{21}$, are newly formed by utilizing bipartition Π (lines 15–18). Each boundary vertex in $\mathcal{V}_1$ that is connected by at least one cut net in Π is included in $\mathcal{P}_{12}$. In a dual manner, each boundary vertex in $\mathcal{V}_2$ that is connected by at least one cut net in Π is included in $\mathcal{P}_{21}$. Note that except for the boundary vertex sets $\mathcal{P}_1 = \{v\}$ and $\mathcal{P}_2 = \{u\}$ 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 $\mathcal{H}_1$ and $\mathcal{H}_2$, the recursive invocations of the HP$_{\text{BDCO}}$ algorithm on $\mathcal{H}_1$ and $\mathcal{H}_2$ are performed to obtain $K/2$-way BDCO partitions $\Pi_1 = \langle \mathcal{V}_1^1, \ldots, \mathcal{V}_{K/2}^1 \rangle$ and $\Pi_2 = \langle \mathcal{V}_1^2, \ldots, \mathcal{V}_{K/2}^2 \rangle$ (lines 19–20). These BDCO partitions are then concatenated to obtain a K-way

Algorithm 1 HP$_{\text{BDCO}}$

Require: Hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, integer K, boundary vertex sets $\mathcal{P}_1$ and $\mathcal{P}_2$

▷ *Fix vertices*
1: $\mathcal{F}_1 \leftarrow$ FIX$(\mathcal{H}, \mathcal{P}_1, K/2)$ ▷ *fix some vertices to* $\mathcal{V}_1$
2: $\mathcal{F}_2 \leftarrow$ FIX$(\mathcal{H}, \mathcal{P}_2, K/2)$ ▷ *fix some vertices to* $\mathcal{V}_2$
▷ *Bipartition* $\mathcal{H}$ *with fixed vertices*
3: $\{\mathcal{V}_1, \mathcal{V}_2\} \leftarrow$ PARTITIONER$(\mathcal{H}, \mathcal{F}_1, \mathcal{F}_2, 2)$
4: **if** $K = 2$ **then** ▷ *base case*
5: $\Pi \leftarrow \langle \mathcal{V}_1, \mathcal{V}_2 \rangle$ ▷ Π *is a 2-way BDCO partition*
6: **else** ▷ *form* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *and recurse on them*
7: $\mathcal{H}_1 \leftarrow (\mathcal{V}_1, \mathcal{N}_1 = \emptyset)$
8: $\mathcal{H}_2 \leftarrow (\mathcal{V}_2, \mathcal{N}_2 = \emptyset)$
▷ *Form* $\mathcal{N}_1$ *and* $\mathcal{N}_2$ *using cut-net removal technique*
9: **for each** $n \in \mathcal{N}$ **do**
10: **if** n is internal to $\mathcal{V}_1$ **then**
11: $\mathcal{N}_1 \leftarrow \mathcal{N}_1 \cup \{n\}$
12: **else if** n is internal to $\mathcal{V}_2$ **then**
13: $\mathcal{N}_2 \leftarrow \mathcal{N}_2 \cup \{n\}$
▷ *Form boundary vertex sets of* $\mathcal{H}_1$ *and* $\mathcal{H}_2$
14: $\mathcal{P}_{11} \leftarrow \mathcal{P}_1, \mathcal{P}_{22} \leftarrow \mathcal{P}_2$ ▷ *inherited boundary vertex sets*
15: $\mathcal{P}_{12} \leftarrow \mathcal{P}_{21} \leftarrow \emptyset$ ▷ *new boundary vertex sets*
16: **for each** cut-net $n \in \mathcal{N}$ **do**
17: $\mathcal{P}_{12} \leftarrow \mathcal{P}_{12} \cup (\mathit{Pins}(n) \cap \mathcal{V}_1)$
18: $\mathcal{P}_{21} \leftarrow \mathcal{P}_{21} \cup (\mathit{Pins}(n) \cap \mathcal{V}_2)$
▷ *Recursive invocations on* $\mathcal{H}_1$ *and* $\mathcal{H}_2$
19: $\Pi_1 = \langle \mathcal{V}_1^1, \ldots, \mathcal{V}_{K/2}^1 \rangle \leftarrow$ HP$_{\text{BDCO}}(\mathcal{H}_1, K/2, \mathcal{P}_{11}, \mathcal{P}_{12})$
20: $\Pi_2 = \langle \mathcal{V}_1^2, \ldots, \mathcal{V}_{K/2}^2 \rangle \leftarrow$ HP$_{\text{BDCO}}(\mathcal{H}_2, K/2, \mathcal{P}_{21}, \mathcal{P}_{22})$
▷ *Concatenate* Π_1 *and* Π_2 *to obtain* Π
21: $\Pi \leftarrow \langle \Pi_1, \Pi_2 \rangle$
22: **return** Π

Algorithm 2 FIX

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

▷ *Initialize fixed vertex set*
1: $\mathcal{F} \leftarrow \emptyset$
▷ *Initialize distance and mark arrays*
2: **for each** vertex $v \in \mathcal{V}$ **do**
3: $dist[v] \leftarrow \infty$
4: $mark[v] \leftarrow$ `false`
▷ *Initialize the queue for multi-source BFS*
5: $\mathcal{Q} \leftarrow \emptyset$
6: **for each** vertex $v \in \mathcal{P}$ **do**
7: ENQUEUE($\mathcal{Q}, v$)
8: $dist[v] \leftarrow 0$
9: $mark[v] \leftarrow$ `true`
▷ *Run BFS*
10: **while** $\mathcal{Q} \neq \emptyset$ **do**
11: $v \leftarrow$ DEQUEUE(Q)
12: $\mathcal{F} \leftarrow \mathcal{F} \cup \{v\}$ ▷ *Add v to fixed vertex set*
13: **if** $dist[v] < D - 1$ **then**
14: **for each** vertex $u \in Adj(v)$ **do**
15: **if** $mark[u] =$ `false` **then**
16: ENQUEUE($\mathcal{Q}, u$)
17: $dist[u] \leftarrow dist[v] + 1$
18: $mark[u] \leftarrow$ `true`
19: **return** $\mathcal{F}$

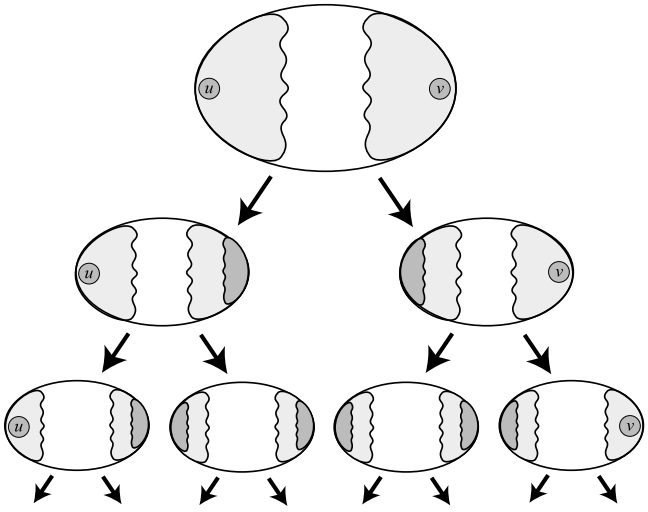

Fig. 3. The proposed vertex fixation scheme illustrated on the first three levels of the RB tree.

BDCO partition $\Pi = \langle \Pi_1, \Pi_2 \rangle = \langle \mathcal{V}_1, \ldots, \mathcal{V}_K \rangle$ (line 21). Finally, partition Π is returned.

Fig. 3 illustrates the proposed vertex fixation scheme on the first three levels of the corresponding RB tree. In the figure, boundary vertices are shown in dark gray, whereas the fixed vertices are shown in light gray. As seen in the figure, the left and right boundary vertex sets at the initial invocation of the HP_{BDCO} algorithm consist of pseudo-peripheral vertices u and v, respectively. Note that $\{u\}$ and $\{v\}$ remain to constitute the left and right boundary vertex sets of each of the further invocations on the leftmost and rightmost hypergraphs formed, respectively. As seen in lines 16–18 of Algorithm 1, each of the remaining boundary vertex sets utilized throughout the proposed algorithm is introduced by a bipartition.

Recall that the HP_{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]. Also recall from Section 3.1 that the cut nets in a BDCO partition of the column-net hypergraph $\mathcal{H}$ of a given matrix A correspond to the coupling columns in the resulting BDCO form. Then, the partitioning objective of minimizing the sum of the cutsizes of the RB steps corresponds to the permutation objective of minimizing the total overlap size of the BDCO form.

Recall that for each vertex v_i in the column-net hypergraph $\mathcal{H}$ of A, we set the weight of v_i to the number of nonzeros in row i of A. Then, the weight of a part $\mathcal{V}_k$ in a K-way BDCO partition of $\mathcal{H}$ corresponds to the total number of nonzeros in diagonal block E_k in the permuted matrix A_{BDCO}. So, maintaining balance on the part weights in (4) corresponds to maintaining balance on the number of nonzeros in diagonal blocks of the BDCO form. Since maintaining balance on the part weights of each bipartition in the proposed algorithm corresponds to maintaining balance on the part weights of the resulting K-way BDCO partition, it also corresponds to the permutation constraint of maintaining balance on the number of nonzeros of diagonal blocks of the BDCO form.

The proposed algorithm introduces an additional overhead of finding fixed vertices to the partitioning cost of the standard recursive hypergraph bipartitioning approach. The runtime of the FIX algorithm on a particular hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is $O(V + E)$, where E denotes the number of adjacency relations in $\mathcal{H}$, i.e., $E = \sum_{v \in \mathcal{V}} |Adj(v)|$. Since the vertices considered on the two invocations of the FIX algorithm in lines 1 and 2 of Algorithm 1 cannot overlap, each adjacency relation in $\mathcal{H}$ is considered only once during the invocation of the HP_{BDCO} algorithm on $\mathcal{H}$. Similarly, the vertices processed in separate HP_{BDCO} invocations at the same level of the RB tree cannot overlap. Therefore, the proposed vertex fixation scheme introduces an $O(V + E)$-time overhead to each level of the RB-tree, where V and E denote the number of vertices and the number of adjacency relations in the initial hypergraph, respectively. For K being a power of 2, which leads to $\log K$ levels in the RB tree, the total overhead introduced by the proposed algorithm is $O(\log K(V + E))$.

4. Experiments

Recall that the HP_{BDCO} algorithm is the first algorithm in the literature that addresses the BDCO problem. For evaluating the performance of HP_{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_{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_{BDCO} and HP_{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_{BDCO} performs relatively close to HP_{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

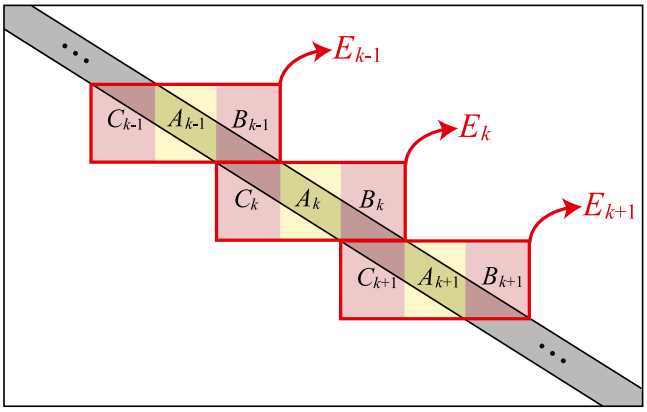

Fig. 4. Obtaining a BDCO form on a banded matrix.

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_{BDCO}

In the baseline RCM_{BDCO} 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 $(m+n) \times (m+n)$ is obtained using A as follows:

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}.$$

Row i of B represents row i of A if $i \leq m$, 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., $j \leq m$), 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.

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 $o \in \{5, 10, 20, 50, 100\}$. Among the obtained semi-real matrices, we excluded matrices with disconnected hypergraphs. In order to prevent the RCM_{BDCO} and HP_{BDCO} 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_o`. Note that for each matrix `M`, the number of rows in semi-real matrices `M_5`, `M_10`, `M_20`, `M_50`, and `M_100` 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 columns decreases as overlap size o increases.

Table 1 displays the results obtained by the RCM_{BDCO} and HP_{BDCO} algorithms along columns 5–7 and 8–11, respectively. The RCM_{BDCO} algorithm was run on each matrix `M_o` once as described in Section 4.1. In the HP_{BDCO} algorithm, the column-net hypergraph of matrix `M_o` was partitioned into 64 parts and then the resulting partition was utilized to permute `M_o` 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_{BDCO} 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_o`, 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

Table 1
Properties of the semi-real test matrices and permutation results obtained by the RCM_{BDCO} and HP_{BDCO} algorithms.

Matrix	#rows	#columns	#nonzeros	RCM_{BDCO}			HP_{BDCO}			
				imbalance (%)	overlap†	overlap/ #columns	imbalance (%)	overlap†	overlap/ #columns	#ideal runs*
photogrammetry2_5	59,904	285,893	2,371,584	0.1	199,801	0.6989	0.0	**315**	0.0011	10
photogrammetry2_10	59,904	285,578	2,371,584	0.0	190,726	0.6679	0.0	**630**	0.0022	10
photogrammetry2_20	59,904	284,948	2,371,584	0.0	191,730	0.6729	0.0	8,929	0.0313	1
photogrammetry2_50	59,904	283,058	2,371,584	0.0	193,401	0.6833	0.0	3,605	0.0127	9
photogrammetry2_100	59,904	279,908	2,371,584	0.4	219,117	0.7828	9.1	20,446	0.0730	0
gemat1_5	315,456	677,765	3,031,616	0.0	92,146	0.1360	0.0	5,298	0.0078	3
gemat1_10	315,456	677,450	3,031,616	0.0	536,989	0.7927	2.8	14,783	0.0218	0
gemat1_20	315,456	676,820	3,031,616	0.0	157,881	0.2333	4.5	25,890	0.0383	0
gemat1_50	315,456	674,930	3,031,616	0.0	577,555	0.8557	0.0	13,294	0.0197	3
gemat1_100	315,456	671,780	3,031,616	10.3	319,351	0.4754	0.0	29,304	0.0436	3
psse1_5	705,792	916,037	3,672,064	0.0	106,617	0.1164	0.0	632	0.0007	1
psse1_10	705,792	915,722	3,672,064	0.0	104,075	0.1137	3.7	1,666	0.0018	0
psse1_20	705,792	915,092	3,672,064	0.0	184,804	0.2020	3.0	2,138	0.0023	0
psse1_50	705,792	913,202	3,672,064	0.0	290,077	0.3176	2.8	**2,233**	0.0024	10
psse1_100	705,792	910,052	3,672,064	0.0	413,962	0.4549	2.9	**2,168**	0.0024	10
Kemelmacher_5	620,352	1,820,613	6,456,000	0.0	843,697	0.4634	0.0	355	0.0002	8
Kemelmacher_10	620,352	1,820,298	6,456,000	0.0	427,100	0.2346	0.0	**630**	0.0003	10
Kemelmacher_20	620,352	1,819,668	6,456,000	0.0	834,819	0.4588	0.0	**1,374**	0.0008	6
Kemelmacher_50	620,352	1,817,778	6,456,000	0.0	567,986	0.3125	0.0	**2,960**	0.0016	10
Kemelmacher_100	620,352	1,814,628	6,456,000	0.0	864,040	0.4762	1.3	10,919	0.0060	0
psse0_5	705,792	1,709,893	6,555,648	0.0	79,383	0.0464	0.0	369	0.0002	7
psse0_10	705,792	1,709,578	6,555,648	0.0	1,459,255	0.8536	0.2	712	0.0004	7
psse0_20	705,792	1,708,948	6,555,648	0.0	319,190	0.1868	1.7	2,265	0.0013	1
psse0_50	705,792	1,707,058	6,555,648	0.0	313,960	0.1839	1.0	**2,735**	0.0016	8
psse0_100	705,792	1,703,908	6,555,648	0.0	414,302	0.2431	4.0	**6,313**	0.0037	9
psse2_5	705,792	1,832,261	7,376,768	0.0	240,840	0.1314	0.0	575	0.0003	4
psse2_10	705,792	1,831,946	7,376,768	0.0	297,952	0.1626	0.0	990	0.0005	4
psse2_20	705,792	1,831,316	7,376,768	0.1	458,841	0.2506	0.0	4,048	0.0022	1
psse2_50	705,792	1,829,426	7,376,768	0.0	483,611	0.2644	3.1	4,968	0.0027	3
psse2_100	705,792	1,826,276	7,376,768	0.0	723,853	0.3964	2.6	**4,918**	0.0027	10
graphics_5	756,608	1,887,237	7,549,056	0.0	22,908	0.0121	0.0	**315**	0.0002	10
graphics_10	756,608	1,886,922	7,549,056	0.0	4,780	0.0025	0.0	1,068	0.0006	8
graphics_20	756,608	1,886,292	7,549,056	0.0	6,406	0.0034	0.0	1,559	0.0008	9
graphics_50	756,608	1,884,402	7,549,056	0.0	8,275	0.0044	0.1	3,629	0.0019	8
graphics_100	756,608	1,881,252	7,549,056	0.0	44,322	0.0236	0.3	**5,657**	0.0030	8
image_interp_5	7,680,000	15,359,685	45,547,712	0.0	95,484	0.0062	0.0	**315**	0.0000	10
image_interp_10	7,680,000	15,359,370	45,547,712	0.0	92,576	0.0060	0.0	**630**	0.0000	10
image_interp_20	7,680,000	15,358,740	45,547,712	0.0	87,563	0.0057	0.0	**1,259**	0.0001	10
image_interp_50	7,680,000	15,356,850	45,547,712	0.0	73,343	0.0048	0.0	**3,149**	0.0002	10
image_interp_100	7,680,000	15,353,700	45,547,712	0.0	49,695	0.0032	0.0	**6,173**	0.0004	10
mesh_deform_5	601,152	14,977,157	54,645,056	0.0	1,007,329	0.0673	0.0	**315**	0.0000	10
mesh_deform_10	601,152	14,976,842	54,645,056	0.0	921,691	0.0615	0.0	**630**	0.0000	10
mesh_deform_20	601,152	14,976,212	54,645,056	0.0	755,161	0.0504	0.0	**1,259**	0.0001	10
mesh_deform_50	601,152	14,974,322	54,645,056	0.0	2,222,376	0.1484	0.0	**3,149**	0.0002	10
mesh_deform_100	601,152	14,971,172	54,645,056	0.0	1,714,018	0.1145	0.0	**6,299**	0.0004	10
Average	–	–	–	0.2	–	0.2752	1.0	–	0.0065	

(†): boldface denotes achieving an ideal BDCO permutation on average.
(*): #ideal runs denotes the number of runs that HP_{BDCO} achieves an ideal BDCO permutation out of ten runs.

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 RCM_{BDCO} algorithm, the HP_{BDCO} algorithm always obtains smaller total overlap size while usually obtaining slightly larger imbalance. On average, RCM_{BDCO} places 27.5% of the columns in overlaps, while this ratio is only 0.6% for HP_{BDCO}. The overlap size obtained by RCM_{BDCO} gets closest to those obtained by HP_{BDCO} on matrices `graphic_10`, `graphic_20`, and `graphic_50`, still being 4.48x, 4.11x, and 2.28x worse, respectively. For RCM_{BDCO} and HP_{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 HP_{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 HP_{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 HP_{BDCO} algorithm obtains

Table 2
Runtimes of the parallel code [24] on three semi-real matrices for $K = 64$ (in seconds).

Matrix	RCM_{BDCO}	HP_{BDCO}
graphics_10	1.70	0.87
graphics_20	2.54	0.88
graphics_50	3.00	1.16

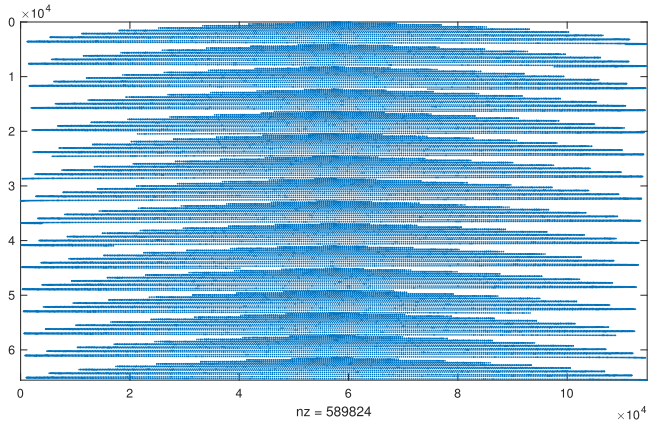

Fig. 5. Matrix mri1.

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 psse1_10, psse1_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 RCM_{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

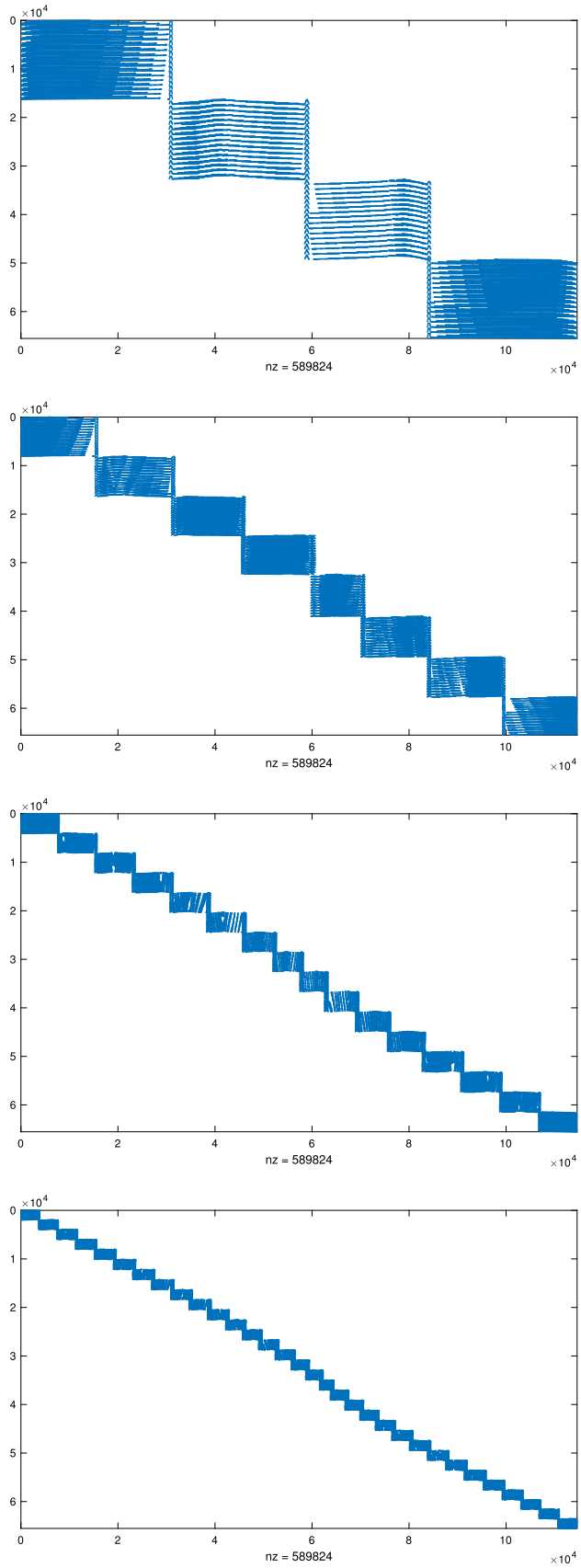

Fig. 6. 4-, 8-, 16-, and 32-way BDCO forms obtained on matrix mri1 by the HP_{BDCO} algorithm.

Table 3
Properties of real matrices and permutation results obtained by RCM_{BDCO} and HP_{BDCO}.

Matrix	#rows	#columns	#nonzeros	K	RCM_{BDCO}			HP_{BDCO}		
					imbalance (%)	overlap	overlap/ #columns	imbalance (%)	overlap	overlap/ #columns
watson_1	201,155	386,992	1,055,093	2	0.0	98,866	0.2555	0.0	26	0.0001
watson_2	352,013	677,224	1,846,391	2	0.0	160,097	0.2364	4.8	16	0.0000
lp_ganges	1,309	1,706	6,937	4	4.7	628	0.3681	4.3	212	0.1243
model8	2,896	6,464	25,277	8	0.2	3,459	0.5351	6.7	2,005	0.3102
l30	2,701	16,281	52,070	8	0.3	5,052	0.3103	2.2	434	0.0267
lp_truss	1,000	8,806	27,836	16	1.6	6,487	0.7367	6.3	3,410	0.3872
progas	1,650	1,900	8,897	16	2.0	1,303	0.6858	0.4	737	0.3879

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 98,866 and 160,097 for $K = 2$, whereas they are only 26 and 16 for HP_{BDCO}, respectively. On lp_ganges, 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 model8 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 38.7% 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.

Table 4
Permutation results obtained on matrix mri1.

K	RCM_{BDCO}			HP_{BDCO}		
	imb. (%)	overlap	overlap/ #columns	imb. (%)	overlap	overlap/ #columns
4	0.0	2,381	0.0208	0.6	2,174	0.0190
8	0.0	5,342	0.0466	5.7	5,066	0.0442
16	0.0	11,103	0.0969	2.0	9,648	0.0842
32	0.0	22,485	0.1961	2.1	19,687	0.1717
64	0.1	45,247	0.3947	2.1	41,067	0.3582

Table 5
Runtimes of the parallel code [24] on real matrices (in seconds).

Matrix	K	RCM_{BDCO}	HP_{BDCO}
lp_ganges	4	0.11	0.03
model8	8	3.03	1.69
l30	8	9.52	0.57
lp_truss	16	7.58	3.78
progas	16	0.10	0.06
mri1	64	100.83	75.40

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, model8, 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, model8, l30, lp_truss, progas, and mri1 are 0.03, 1.69, 0.57, 3.78, 0.06, and 75.40 seconds, 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

[1] S. Acer, E. Kayaaslan, C. Aykanat, A recursive bipartitioning algorithm for permuting sparse square matrices into block diagonal form with overlap, SIAM J. Sci. Comput. 35 (1) (2013) C99–C121, http://dx.doi.org/10.1137/120861242, arXiv:https://doi.org/10.1137/120861242.

[2] P.R. Amestoy, I.S. Duff, C. Puglisi, Multifrontal QR factorization in a multiprocessor environment, Numer. Linear Algebra Appl. 3 (4) (1996) 275–300.

[3] C. Aykanat, B.B. Cambazoglu, F. Findik, T. Kurc, Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids, J. Parallel Distrib. Comput. 67 (1) (2007) 77–99, http://dx.doi.org/10.1016/j.jpdc.2006.05.005, URL http://www.sciencedirect.com/science/article/pii/S0743731506001122.

[4] C. Aykanat, B.B. Cambazoglu, B. Uçar, Multi-level direct K-way hypergraph partitioning with multiple constraints and fixed vertices, J. Parallel Distrib. Comput. 68 (5) (2008) 609–625, http://dx.doi.org/10.1016/j.jpdc.2007.09.006, URL http://www.sciencedirect.com/science/article/pii/S0743731507001724.

[5] C. Aykanat, A. Pinar, U.V. Catalyurek, Permuting sparse rectangular matrices into block-diagonal form, SIAM J. Sci. Comput. 25 (6) (2004) 1860–1879, http://dx.doi.org/10.1137/S1064827502401953, arXiv:http://dx.doi.org/10.1137/S1064827502401953.

[6] A.M. Bruckstein, D.L. Donoho, M. Elad, From sparse solutions of systems of equations to sparse modeling of signals and images, SIAM Rev. 51 (1) (2009) 34–81, http://dx.doi.org/10.1137/060657704, arXiv:http://dx.doi.org/10.1137/060657704.

[7] A. Buttari, Fine-grained multithreading for the multifrontal QR factorization of sparse matrices, SIAM J. Sci. Comput. 35 (4) (2013) C323–C345, http://dx.doi.org/10.1137/110846427, arXiv:http://dx.doi.org/10.1137/110846427.

[8] U.V. Catalyurek, C. Aykanat, Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication, IEEE Trans. Parallel Distrib. Syst. 10 (7) (1999) 673–693, http://dx.doi.org/10.1109/71.780863.

[9] U.V. Catalyurek, E.G. Boman, K.D. Devine, D. Bozdağ, R.T. Heaphy, L.A. Riesen, A repartitioning hypergraph model for dynamic load balancing, J. Parallel Distrib. Comput. 69 (8) (2009) 711–724, http://dx.doi.org/10.1016/j.jpdc.2009.04.011, URL http://www.sciencedirect.com/science/article/pii/S0743731509000847.

[10] U.V. Çatalyürek, C. Aykanat, PaToH: Partitioning Tool for Hypergraphs, Tech. Rep., Department of Computer Engineering, Bilkent University, 1999.

[11] A. Cevahir, C. Aykanat, A. Turk, B.B. Cambazoglu, Site-based partitioning and repartitioning techniques for parallel Page Rank computation, IEEE Trans. Parallel Distrib. Syst. 22 (5) (2011) 786–802, http://dx.doi.org/10.1109/TPDS.2010.119.

[12] S.F. Cotter, B.D. Rao, K. Engan, K. Kreutz-Delgado, Sparse solutions to linear inverse problems with multiple measurement vectors, IEEE Trans. Signal Process. 53 (7) (2005) 2477–2488, http://dx.doi.org/10.1109/TSP.2005.849172.

[13] T.A. Davis, User's guide for SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package, 2009.

[14] T.A. Davis, Algorithm 915, SuiteSparseQR: multifrontal multithreaded rank-revealing sparse QR factorization, ACM Trans. Math. Software 38 (1) (2011) 8:1–8:22, http://dx.doi.org/10.1145/2049662.2049670, URL http://doi.acm.org/10.1145/2049662.2049670.

[15] T.A. Davis, Y. Hu, The University of Florida sparse matrix collection, ACM Trans. Math. Softw. 38 (1) (2011) http://dx.doi.org/10.1145/2049662.2049663, https://doi.org/10.1145/2049662.2049663.

[16] A. George, J.W.H. Liu, An implementation of a pseudoperipheral node finder, ACM Trans. Math. Software 5 (3) (1979) 284–295, http://dx.doi.org/10.1145/355841.355845, URL http://doi.acm.org/10.1145/355841.355845.

[17] G.H. Golub, A.H. Sameh, V. Sarin, A parallel balance scheme for banded linear systems, Numer. Linear Algebra Appl. 8 (5) (2001) 297–316.

[18] C.L. Lawson, R.J. Hanson, Solving Least Squares Problems, SIAM, 1995.

[19] K. Matsuura, Y. Okabe, Selective minimum-norm solution of the biomagnetic inverse problem, IEEE Trans. Biomed. Eng. 42 (6) (1995) 608–615, http://dx.doi.org/10.1109/10.387200.

[20] J. Reid, J. Scott, Reducing the total bandwidth of a sparse unsymmetric matrix, SIAM J. Matrix Anal. Appl. 28 (3) (2006) 805–821, http://dx.doi.org/10.1137/050629938, arXiv:https://doi.org/10.1137/050629938.

[21] A.H. Sameh, V. Sarin, Parallel algorithms for indefinite linear systems, Parallel Comput. 28 (2) (2002) 285–299, http://dx.doi.org/10.1016/S0167-8191(01)00140-5, URL http://www.sciencedirect.com/science/article/pii/S0167819101001405.

[22] M.K. Sen, P.L. Stoffa, Global Optimization Methods in Geophysical Inversion, Cambridge University Press, 2013.

[23] T.E. Tezduyar, A. Sameh, Parallel finite element computations in fluid mechanics, Comput. Methods Appl. Mech. Engrg. 195 (13–16) (2006) 1872–1884, http://dx.doi.org/10.1016/j.cma.2005.05.038, A Tribute to Thomas J.R. Hughes on the Occasion of his 60th Birthday, URL http://www.sciencedirect.com/science/article/pii/S0045782505003105.

[24] F.S. Torun, M. Manguoglu, C. Aykanat, Parallel minimum norm solution of sparse block diagonal column overlapped underdetermined systems, ACM Trans. Math. Software 43 (4) (2017) 31:1–31:21, http://dx.doi.org/10.1145/3004280, URL http://doi.acm.org/10.1145/3004280.

[25] J.Z. Wang, S.J. Williamson, L. Kaufman, Magnetic source images determined by a lead-field analysis: the unique minimum-norm least-squares estimation, IEEE Trans. Biomed. Eng. 39 (7) (1992) 665–675, http://dx.doi.org/10.1109/10.142641.

[26] A. Yzelman, R. Bisseling, Cache-oblivious sparse matrix–vector multiplication by using sparse matrix partitioning methods, SIAM J. Sci. Comput. 31 (4) (2009) 3128–3154, http://dx.doi.org/10.1137/080733243, arXiv:https://doi.org/10.1137/080733243.

[27] M. Zhdanov, P. Wannamaker, in: M.S. Zhdanov, P.E. Wannamaker (Eds.), Geophysical Inverse Theory and Regularization Problems, in: Methods in Geochemistry and Geophysics, vol. 35, Elsevier, 2002, pp. v–vi.

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.