SIAM J. SCI. COMPUT.
Vol. 35, No. 1, pp. C99–C121

A RECURSIVE BIPARTITIONING ALGORITHM FOR PERMUTING SPARSE SQUARE MATRICES INTO BLOCK DIAGONAL FORM WITH OVERLAP*

SEHER ACER†, ENVER KAYAASLAN†, AND CEVDET AYKANAT†

Abstract. We investigate the problem of symmetrically permuting a square sparse matrix into a block diagonal form with overlap. This permutation problem arises in the parallelization of an explicit formulation of the multiplicative Schwarz preconditioner and a more recent block overlapping banded linear solver as well as its application to general sparse linear systems. In order to formulate this permutation problem as a graph theoretical problem, we define a constrained version of the multiway graph partitioning by vertex separator (GPVS) problem, which is referred to as the ordered GPVS (oGPVS) problem. However, existing graph partitioning tools are unable to solve the oGPVS problem. So, we also show how the recursive bipartitioning framework can be utilized for solving the oGPVS problem. For this purpose, we propose a left-to-right bipartitioning approach together with a novel vertex fixation scheme so that existing 2-way GPVS tools that support fixed vertices can be effectively and efficiently utilized in the recursive bipartitioning framework. Experimental results on a wide range of matrices confirm the validity of the proposed approach.

Key words. sparse square matrices, block diagonal form with overlap, graph partitioning by vertex separator, recursive bipartitioning, partitioning with fixed vertices, combinatorial scientific computing

AMS subject classifications. 05C50, 05C85, 65F50, 68R10

DOI. 10.1137/120861242

1. Introduction.

Our target problem is to symmetrically permute rows and columns of an $N \times N$ structurally symmetric sparse matrix A into a K-way block diagonal (BDO) form A^π with overlap:

(1.1)

$$A^\pi = PAP^T = A_{BDO} = \begin{bmatrix} A_{1,1} & A_{1,2} & & & & & \\ A_{1,2}^T & C_{1,1} & A_{2,1} & C_{1,2} & & & \\ & A_{2,1}^T & A_{2,2} & A_{2,3} & & & \\ & C_{1,2}^T & A_{2,3}^T & C_{2,2} & \cdots & & \\ & & & \vdots & \ddots & & \\ & & & & & C_{K-1,K-1} & A_{K,K-1} \\ & & & & & A_{K,K-1}^T & A_{K,K} \end{bmatrix}.$$

Here, P denotes an $N \times N$ permutation matrix. The BDO form contains K diagonal blocks $D_1, D_2, \ldots, D_K$, where

$$D_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\ C_{k-1,k}^T & A_{k,k+1}^T & C_{k,k} \end{bmatrix} \text{ for } k = 2, 3, \ldots, K-1, \tag{1.2}$$

*Submitted to the journal's Software and High-Performance Computing section January 3, 2012; accepted for publication (in revised form) October 26, 2012; published electronically February 12, 2013. This work was financially supported by the PRACE project funded in part by the EU's 7th Framework Programme (FP7/2007-2013) under grant agreements RI-211528 and FP7-261557.
http://www.siam.org/journals/sisc/35-1/86124.html

†Computer Engineering Department, Bilkent University, Ankara, Turkey (acer@cs.bilkent.edu.tr, enver@cs.bilkent.edu.tr, aykanat@cs.bilkent.edu.tr).

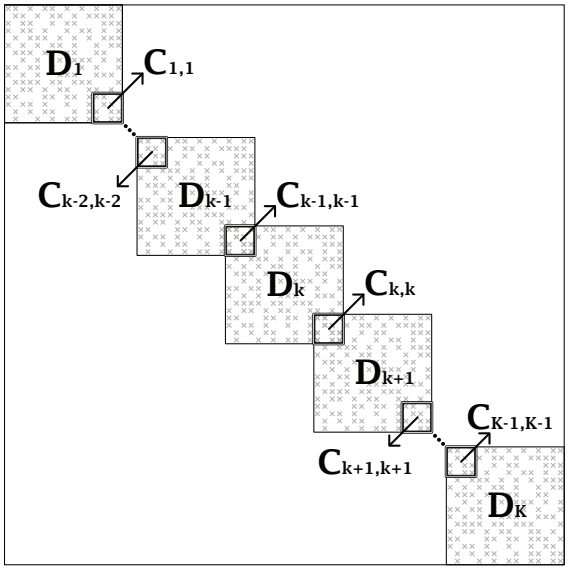

FIG. 1.1. *Block diagonal form with overlap.*

$$D_1 = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{1,2}^T & C_{1,1} \end{bmatrix}, \quad D_K = \begin{bmatrix} C_{K-1,K-1} & A_{K,K-1} \\ A_{K,K-1}^T & A_{K,K} \end{bmatrix}. \tag{1.3}$$

In (1.2), $C_{k,k}$ denotes the coupling diagonal block between the successive kth and $(k+1)$th diagonal blocks D_k and D_{k+1}, respectively. Note that A_{BDO} is also structurally symmetric since symmetric permutation is applied on the symmetric matrix A. Figure 1.1 displays a better visualization of the BDO form of the matrix A.

In the A-to-A_{BDO} permutation problem, the permutation objective is to minimize the total overlap size, which is defined as

$$N_c = \sum_{k=1}^{K-1} n_c^k. \tag{1.4}$$

Here, n_c^k denotes the number of the rows/columns of the coupling diagonal block $C_{k,k}$. The permutation constraint is to maintain balance on the nonzero counts of the diagonal blocks D_k's.

The A-to-A_{BDO} permutation problem arises in the parallelization of an explicit formulation of the multiplicative Schwarz preconditioner [13] and a more recent domain decomposition method proposed by Naumov, Manguoglu, and Sameh [18] and Naumov and Sameh [19]. These overlapping domain decomposition methods have the limitation that each subdomain has only two neighbors, whereas most domain decomposition methods do not have such a limitation. In these parallelizations, each diagonal block D_k of the permuted matrix together with the associated computations is assigned to a distinct processor k. The permutation objective corresponds to minimizing the total communication volume [13, 18, 19] and minimizing the size of the balance system [18, 19], as well as the upper bound on the number of iterations required for convergence of the iterative method [14]. The permutation constraint relates to maintaining balance on the computational loads of processors during the iterations [13].

The problem of permuting sparse rectangular matrices into bordered block diagonal (BBD) forms, which we refer to as the A-to-A_{BBD} permutation problem, was

investigated in the literature (singly BBD form [2, 10] and doubly BBD form [2]). In the A-to-A_{BBD} problem, the permutation objective is to minimize the border size, whereas the permutation constraint is to maintain balance on the dimensions and/or the nonzero counts of diagonal blocks. The A-to-A_{BDO} and A-to-A_{BBD} problems are quite different in terms of both parallel application and combinatorial aspects. In terms of parallelization objective, the A-to-A_{BBD} problem is used in the parallelization of applications where diagonal blocks give rise to subproblems that can be solved independently and the border corresponds to a possibly serial coordination task to combine the subproblem solutions into a solution of the original problem. In terms of combinatorial aspects, the BDO form is a rather constrained version of the BBD forms, because in the BDO form, rows and columns of coupling diagonal blocks link only the successive diagonal blocks, whereas in the BBD forms, the rows and/or columns of the border(s) may link nonconsecutive diagonal blocks and possibly all diagonal blocks.

To our knowledge, the A-to-A_{BDO} permutation problem has only been addressed in a recent work by Kahou, Grigori, and Sosonkina [12]. In that work, they propose a bottom-up graph partitioning algorithm on the standard graph representation of matrix A. Their algorithm first finds a level structure in which the number of levels is maximized. This level structure is considered as a chain, and an initial K-way partition is obtained by running a chain-on-chain partitioning algorithm [20] that minimizes the load of the maximally loaded part. In the resulting K-way partition, each part contains one or more consecutive levels so that all inter-part edges are confined to be between consecutive parts. If the balance of the resulting partition is found to be unsatisfactory, they improve the balance through exchanging vertices between consecutive parts. Then, for each two consecutive parts, a narrow separator is obtained from the wide separator by utilizing the minimum vertex cover algorithm. Finally, using the node separator refinement algorithm of [17], sizes of the separators are decreased by utilizing the first two steps of the Dulmage Mendelsohn decomposition for finding vertex subsets to be moved between separators and parts [21]. Given a level structure with maximum length, the running time of this partitioning algorithm is $O(KlgK + e\sqrt{Kn})$, where n and e, respectively, denote the number of vertices and edges.

The contributions of this paper are as follows. We first define a constrained version of the K-way graph partitioning by vertex separator (GPVS) problem, which is referred to as the ordered GPVS (oGPVS) problem. Then we formulate the A-to-A_{BDO} permutation problem as a K-way oGPVS problem. However, existing graph partitioning tools are unable to solve the oGPVS problem. So, we also show how the recursive bipartitioning (RB) framework, which is successfully and commonly used for K-way graph/hypergraph partitioning, can be utilized for solving the oGPVS problem. For this purpose, we propose a left-to-right bipartitioning approach together with a novel vertex fixation scheme so that existing 2-way GPVS tools that support fixed vertices can be effectively and efficiently utilized in the RB framework.

The rest of the paper is organized as follows. Section 2 provides background information. oGPVS problem formulation is presented in section 3. Section 4 presents and discusses the RB-based algorithm proposed for solving the oGPVS problem. Experimental results are given in section 5. Finally, section 6 concludes the paper.

2. Preliminaries.

2.1. Standard graph model for representing sparse matrices.

In the standard graph model, an $N \times N$ square and symmetric matrix $A = (a_{ij})$ is represented as an undirected graph $G(A) = (\mathcal{V}, \mathcal{E})$ with N vertices. Vertex set $\mathcal{V}$ and edge set

$\mathcal{E}$, respectively, represent the rows/columns and off-diagonal nonzeros of matrix A. For each row/column r_i/c_i, $\mathcal{V}$ contains one vertex v_i. For each symmetric nonzero pair a_{ij} and a_{ji}, $\mathcal{E}$ contains one edge e_{ij} that connects the vertices v_i and v_j.

2.2. Graph partitioning by vertex separator (GPVS). For a given undirected graph $G = (\mathcal{V}, \mathcal{E})$, we use the notation $Adj(v_i)$ to denote the set of vertices that are adjacent to vertex v_i in G. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}' \subseteq \mathcal{V}$, i.e., $Adj(\mathcal{V}') = \bigcup_{v_i \in \mathcal{V}'} Adj(v_i) - \mathcal{V}'$. Two vertex subsets $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{V}'' \subseteq \mathcal{V}$ are said to be adjacent if $Adj(\mathcal{V}') \cap \mathcal{V}'' \neq \emptyset$ (or equivalently $Adj(\mathcal{V}'') \cap \mathcal{V}' \neq \emptyset$) and nonadjacent otherwise.

A vertex subset $\mathcal{S}$ is a K-way *vertex separator* if the subgraph induced by the vertices in $\mathcal{V} - \mathcal{S}$ has at least K connected components. $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{S}\}$ is a K-way vertex partition of G by vertex separator $\mathcal{S} \subseteq \mathcal{V}$ if all parts are nonempty (i.e., $\mathcal{V}_k \neq \emptyset$ for $k = 1, \ldots, K$), all parts and the separator are pairwise disjoint, the union of the parts and the separator gives $\mathcal{V}$, and the vertex parts are pairwise nonadjacent (i.e., $Adj(\mathcal{V}_k) \subseteq \mathcal{S}$ for $k = 1, \ldots, K$). $\mathcal{V}_k \cap Adj(\mathcal{S})$ is said to be the boundary vertex set of part $\mathcal{V}_k$.

In the GPVS problem, the partitioning objective is to minimize the separator size, which is usually defined as the number of vertices in the separator, i.e.,

$$Separatorsize(\Pi_{VS}) = |\mathcal{S}|. \tag{2.1}$$

The partitioning constraint is to maintain a balance criterion on the part weights, which is usually defined as

$$\max_{1 \leq k \leq K} \{W(\mathcal{V}_k)\} \leq (1 + \epsilon) W_{avg}. \tag{2.2}$$

Here, ϵ is the maximum imbalance ratio allowed and $W_{avg} = \sum_{k=1}^{K} W(\mathcal{V}_k)/K$ is the average part weight, where

$$W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i) \tag{2.3}$$

is the weight of part $\mathcal{V}_k$ and $w(v_i)$ is the weight associated with vertex v_i.

2.3. Recursive bipartitioning paradigm. The RB paradigm has been widely and successfully utilized in K-way graph/hypergraph partitioning. In the RB scheme for K-way GPVS, first a 2-way vertex separator $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{S}\}$ of the original graph $G = G[\mathcal{V}]$ is obtained and then this 2-way Π_{VS} is decoded to construct two subgraphs using the separator-vertex removal scheme to capture the K-way separator size. The separator-vertex removal scheme discards all separator vertices of the 2-way Π_{VS}, since they contribute to the K-way separator size only once, thus inducing vertex-induced subgraphs $G[\mathcal{V}_1]$ and $G[\mathcal{V}_2]$. Then 2-way GPVS is recursively applied on both $G[\mathcal{V}_1]$ and $G[\mathcal{V}_2]$. This procedure continues until the desired number of parts is reached in $lg_2 K$ recursion levels, assuming K is a power of 2.

In forthcoming discussions, we utilize the concept of an RB tree which is a full and complete (for K is a power of 2) binary rooted tree. Each node of an RB tree represents a vertex subset of $\mathcal{V}$ as well as the respective induced subgraph on which a 2-way GPVS to be applied. Note that the root node represents both the original vertex set $\mathcal{V}$ and the original graph G.

2.4. Graph/hypergraph partitioning with fixed vertices. Graph/hypergraph partitioning with fixed vertices has been used for solving the repartitioning/remapping problem encountered in the parallelization of irregular applications [1, 7, 8].

In graph/hypergraph partitioning with fixed vertices, there exists an additional constraint on the part assignment of some vertices. That is, some vertices, which are referred to as fixed vertices, are preassigned to parts prior to the partitioning operation, with the constraint that, at the end of the partitioning, fixed vertices will remain in the part to which they are preassigned. We use the notation $\mathcal{F}_k$ to denote the subset of vertices that are fixed to part $\mathcal{V}_k$ for $k = 1, 2, \ldots, K$. The remaining vertices (i.e., vertices in $\mathcal{V} - \bigcup_{k=1}^{K} \mathcal{F}_k$) are referred to as the free vertices since they can be assigned to any part. In GPVS with fixed vertices, free vertices can be assigned to the separator as well as to the parts.

3. Ordered GPVS formulation. In order to formulate the A-to-A_{BDO} permutation problem as a graph theoretical problem, we define a constrained version of the K-way GPVS problem which is referred to as the *ordered GPVS (oGPVS) problem*.

3.1. Ordered GPVS problem definition. In the oGPVS problem, we use a special form of vertex separator which is referred to as the *ordered vertex separator (oVS)*. In oVS of a given graph G, there exists an order on the vertex parts, and the overall separator is partitioned into an ordered set $\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K-1} \rangle$ of mutually disjoint $K-1$ subseparators in such a way that

(i) each vertex in subseparator $\mathcal{S}_k$ connects vertices only in successive parts $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$ for $k = 1, 2, \ldots, K-1$;
(ii) edges between subseparators are restricted to be between only successive supseparators, i.e., $\mathcal{S}_k$ and $\mathcal{S}_{k+1}$ for $k = 1, 2, \ldots, K-2$.

Here, we denote $\mathcal{S}_k$ to be the right subseparator of $\mathcal{V}_k$ and the left subseparator of $\mathcal{V}_{k+1}$. We introduce the following formal definitions for oVS and the oGPVS problem.

DEFINITION 3.1 (ordered vertex separator Π_{oVS}). $\Pi_{oVS} = \{\langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle; \mathcal{S}\}$ *is a K-way ordered vertex partition of $G = (\mathcal{V}, \mathcal{E})$ by an oVS $\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K-1} \rangle$ if each subseparator $\mathcal{S}_k$ is nonempty; all parts and subseparators are pairwise disjoint; the union of parts and subseparators gives $\mathcal{V}$; parts are pairwise nonadjacent; only successive subseparators can be pairwise adjacent; and successive parts $\mathcal{V}_k$ and $\mathcal{V}_{k+1}$ are connected by the vertices of the subseparator $\mathcal{S}_k$ between these two parts.*

DEFINITION 3.2 (oGPVS problem). *Given a graph $G = (\mathcal{V}, \mathcal{E})$, an integer K, and a maximum allowable imbalance ratio ϵ, the oGPVS problem is finding a K-way oVS $\Pi_{oVS}(G) = \{\langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle; \mathcal{S}\}$ of G by a vertex separator $\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{K-1} \rangle$ that minimizes the overall separator size $|\mathcal{S}| = \sum_{k=1}^{K-1} |\mathcal{S}_k|$ while satisfying the balance criterion on the weights of K parts given in* (2.2).

3.2. Formulation. The following theorem shows how the A-to-A_{BDO} permutation problem can be formulated as an oGPVS problem.

THEOREM 3.3. *Let $G(A) = (\mathcal{V}, \mathcal{E})$ be the standard graph representation of a given structurally symmetric sparse matrix A where the weight of each vertex v_i is set to be equal to the number of nonzeros in row/column i. A K-way oVS $\Pi_{oVS} = \{\langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle; \mathcal{S}\}$ of $G(A)$ can be decoded as a partial permutation of A to a K-way BDO form A_{BDO}, where the vertices of part $\mathcal{V}_k$ and subseparator $\mathcal{S}_k$ constitute the rows/columns of the subblock $A_{k,k}$ and $C_{k,k}$, respectively. Thus,*

- *$|\mathcal{S}_k| = n_c^k$, and hence minimizing the separator size $|\mathcal{S}| = \sum_{k=1}^{K-1} |\mathcal{S}_k|$ corresponds to minimizing total overlap size $N_c = \sum_{k=1}^{K-1} n_c^k$;*

- *maintaining balance on the part weights relates to maintaining balance on the nonzero counts of the diagonal blocks.*

Proof. Consider a K-way oVS $\Pi_{oVS} = \{\langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle; \mathcal{S}\}$ of $G(A)$. Π_{oVS} can be decoded as a partial permutation on the rows and columns of A to induce a permuted matrix A^{π} as follows: The rows/columns corresponding to the vertices in $\mathcal{V}_k$ are ordered after the rows/columns corresponding to the vertices in $\mathcal{S}_{k-1}$ and before the rows/columns corresponding to the vertices in $\mathcal{S}_k$. In a dual manner, the rows/columns corresponding to the vertices in $\mathcal{S}_k$ are ordered after the rows/columns corresponding to the vertices in $\mathcal{V}_k$ and before the rows/columns corresponding to the vertices in $\mathcal{V}_{k+1}$. Note that Π_{oVS} induces a partial permutation, since the rows/columns corresponding to the vertices in the same part or in the same subseparator can be ordered arbitrarily. Also note that Π_{oVS} induces a symmetric permutation on the rows and columns of matrix A since each vertex v_i of $G(A)$ represents both row i and column i of A.

In the permuted matrix A^{π}, the vertices of part $\mathcal{V}_k$ constitute the rows/columns of the diagonal subblock $A_{k,k}$ of D_k and the vertices of subseparator $\mathcal{S}_k$ constitute the rows/columns of the coupling diagonal block $C_{k,k}$ between D_k and D_{k+1}. Since we have $Adj(\mathcal{V}_k) = \mathcal{S}_{k-1} \cup \mathcal{S}_k$ and $Adj(\mathcal{V}_k) \cap Adj(\mathcal{V}_{k+1}) = \mathcal{S}_k$ by the definition of oVS, the overlaps between the diagonal blocks D_k's are restricted to be only between the successive D_k's, and $C_{k,k}$ constitute the overlap between D_k and D_{k+1}. Thus permuted matrix A^{π} is a BDO form of matrix A.

Since the vertices in $\mathcal{S}_k$ constitute the rows/columns of the coupling diagonal block $C_{k,k}$, minimizing the overall separator size $|\mathcal{S}|$ corresponds to minimizing the total overlap size N_c.

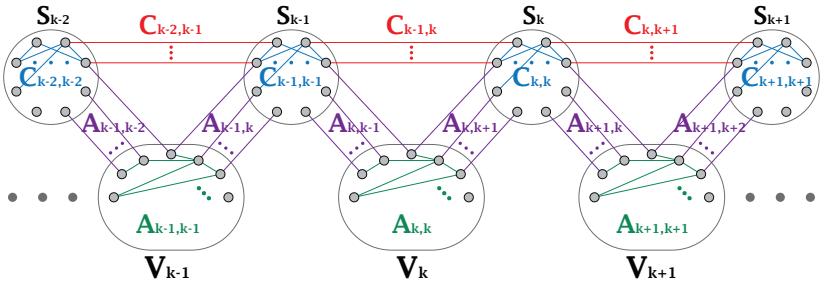

FIG. 3.1. *Correspondence between the nonzeros of block D_k and the edges of $\mathcal{S}_{k-1} \cup \mathcal{V}_k \cup \mathcal{S}_k$.*

Here, we show that balancing on the part weights relates to the balancing of the nonzero counts in the diagonal blocks. For this purpose, we mention the association between the edges of $G(A)$ in oVS form and the nonzeros of $A^{\pi} = A_{BDO}$ induced by Π_{oVS}. We introduce Figure 3.1 in order to clarify the forthcoming discussion. The nonzeros in the diagonal subblocks $C_{k-1,k-1}$, $A_{k,k}$, and $C_{k,k}$ of D_k, respectively, correspond to the internal edges of subseparator $\mathcal{S}_{k-1}$, part $\mathcal{V}_k$, and subseparator $\mathcal{S}_k$. The nonzeros in the off-diagonal subblocks $C_{k-1,k-1}$, and $A^T_{k,k-1}$ of D_k correspond to the edges connecting the vertices in subseparator $\mathcal{S}_{k-1}$, whereas the nonzeros in the off-diagonal subblocks $A_{k,k+1}$ and $A^T_{k,k+1}$ of D_k correspond to the edges connecting the vertices in $\mathcal{V}_k$ and $\mathcal{S}_k$. The nonzeros in the off-diagonal subblocks $C_{k-1,k}$ and $C^T_{k-1,k}$ of D_k correspond to the edges connecting the vertices in successive subseparators $\mathcal{S}_{t1}$ and $\mathcal{S}_k$. Thus, the weight of a part $\mathcal{V}_k$ computed according to (2.3) gives $W(\mathcal{V}_k) = nnz(A_{k,k-1}) + nnz(A_{k,k}) + nnz(A_{k,k+1})$, where $nnz(\cdot)$ denotes the

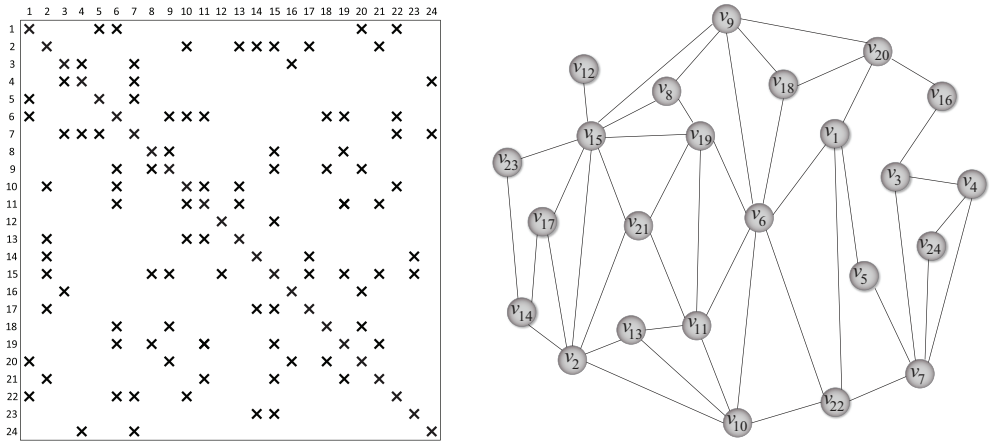

FIG. 3.2. *Sample matrix A and its standard graph representation $G(A)$.*

number of nonzeros in the respective matrix. Since $nnz(A_{k,k-1}^T) = nnz(A_{k,k-1})$ and $nnz(A_{k,k+1}^T) = nnz(A_{k,k+1})$, $W(\mathcal{V}_k)$ represents the sum of the nonzero counts of diagonal block $A_{k,k}$ plus one of the two off-diagonal blocks $A_{k,k-1}$ and $A_{k,k-1}^T$ plus one of the two off-diagonal blocks $A_{k,k+1}$ and $A_{k,k+1}^T$. One possible nonzero-count coverage of $W(\mathcal{V}_k)$ is shown in (3.1) as highlighted submatrices:

$$D_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\ C_{k-1,k}^T & A_{k,k+1}^T & C_{k,k} \end{bmatrix}. \tag{3.1}$$

Note that $W(\mathcal{S}_{k-1}) + W(V_k) + W(\mathcal{S}_k)$ computed in the vertex-induced subgraph $G[\mathcal{S}_{k-1} \cup \mathcal{V}_k \cup \mathcal{S}_k]$ of $G(A)$ gives $nnz(D_k)$. Thus, $W(\mathcal{V}_k)$ can be considered to approximate $nnz(D_k)$ when the number of vertices and edges of vertex-induced subgraph $G[\mathcal{S}_{k-1} \cup \mathcal{S}_k]$ of $G(A)$ are small, which is partially implied by the partitioning objective of minimizing the separator size. ∎

Figure 3.2 shows a sample 24×24 matrix A which contains 116 nonzeros and the standard graph representation G of A which contains 24 vertices and 46 edges. Figure 3.3 shows a 4-way oVS $\Pi_{oVS}(G) = \{\langle \mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3, \mathcal{V}_4 \rangle; \langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3 \rangle\}$ of G, where $\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3$, and $\mathcal{V}_4$, respectively, contain 4, 5, 4, and 4 vertices, and $\mathcal{S}_1, \mathcal{S}_2$, and $\mathcal{S}_3$, respectively, contain 2, 3, and 2 vertices. Figure 3.4 shows a BDO form of the sample matrix A given in Figure 3.2, which is induced by $\Pi_{oVS}(G)$ given in Figure 3.3. As seen in Figure 3.4, the BDO form, respectively, contains diagonal blocks D_1, D_2, D_3, and D_4 of dimensions 6×6, 10×10, 9×9, and 6×6, and coupling diagonal blocks $C_{1,1}$, $C_{2,2}$, and $C_{3,3}$ of dimensions 2×2, 3×3, and 2×2 between diagonal blocks D_1 and D_2, D_2 and D_3, and D_3 and D_4.

3.3. Parallel requirements for a sample application. Here, we will briefly examine the communication and computation requirements of the parallel implementation of an explicit formulation of the multiplicative Schwarz preconditioner given in [13] in order to show the correspondence between its efficient parallelization and the constraint and objective of the proposed oGPVS formulation. In this parallel implementation, each processor k stores diagonal block D_k and its LU factors as well as

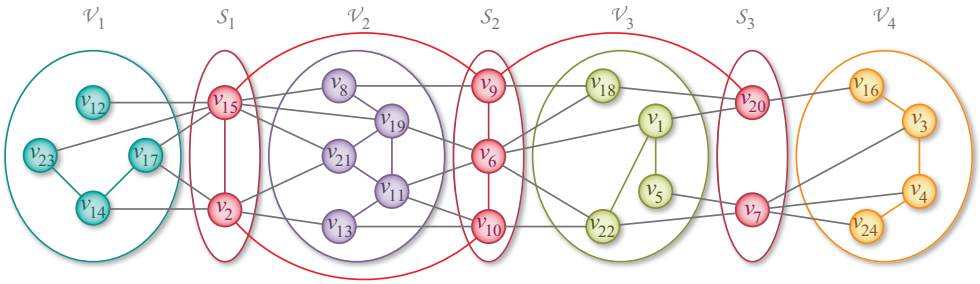

FIG. 3.3. *A 4-way oVS form of $G(A)$ given in Figure 3.2.*

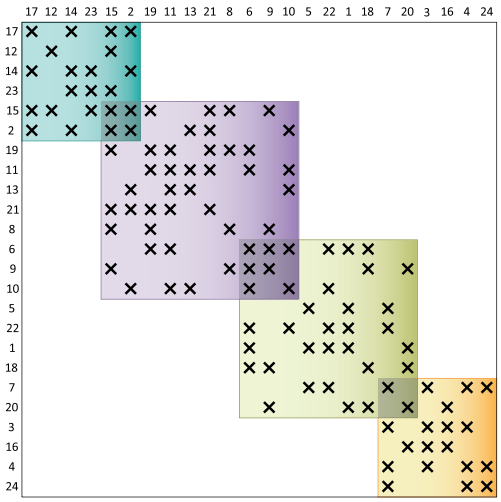

FIG. 3.4. *A 4-way BDO form of the sample matrix A induced by the 4-way oVS given in Figure 3.3.*

the kth overlapping subvectors of all column vectors involved in the iterative solution of $A^\pi x^\pi = b^\pi$, where $x^\pi = P^T x$ and $b^\pi = Pb$. To simplify the notation of the forthcoming discussion, we will omit the "π" superscripts which denote the permuted matrix and vectors. For example, x_k denotes the subvector of x that corresponds to the columns of D_k, where x_k is partitioned into three subsubvectors x_k^1, x_k^2, and x_k^3 that, respectively, correspond to the columns of $C_{k-1,k-1}$, $A_{k,k}$, and $C_{k,k}$. So x_k overlaps with x_{k-1} through x_{k-1}^3 and x_k^1, and overlaps with x_{k+1} through x_k^3 and x_{k+1}^1. Each iteration involves a residual computation step and a preconditioning step [13].

The residual computation step involves a local sparse matrix-vector multiply (SpMxV) operation of the form $z_k = \hat{D}_k x_k$ for updating the local residual vector through the local linear vector operation $r_k = b_k - z_k$ in each processor k. Here $\hat{D}_k$ is the diagonal block D_k from which the coupling diagonal subblock $C_{k,k}$ is zeroed as shown below:

$$\hat{D}_k = \begin{bmatrix} C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\ A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\ C_{k-1,k}^T & A_{k,k+1}^T & 0 \end{bmatrix}. \tag{3.2}$$

The preconditioning step involves the solution of a local linear system of the form $D_k y_k = r_k$ for the update of the local solution vector through the linear vector operation $x_k = x_k + y_k$ in each processor k. y_k is obtained through performing local forward and backward substitution operations on the LU factors of D_k. The local LU factorizations of D_k matrices are performed in a parallel preprocessing step [13]. The preconditioning step also involves a SpMxV operation of the form $y_k^3 = C_{k,k} y_k^3$, where y_k^3 is the subvector of y_k that corresponds to the rows of $C_{k,k}$.

The nonzero count $nnz(D_k)$ of a diagonal block D_k precisely defines the amount of work associated with these two SpMxV operations $z_k = \hat{D}_k x_k$ and $y_k^3 = C_{k,k} y_k^3$. However, $nnz(D_k)$ can only be used as an estimate for the work associated with the LU factorization of D_k, as well as the nonzero counts of the LU factors of D_k (due to fill-ins). So $nnz(D_k)$ only relates to the local forward and backward substitutions performed on the LU factors of D_k throughout the iterations. Hence maintaining balance on the part weights relates to maintaining balance on the computational loads of processors during the iterations.

In each residual computation step, processor k sends z_k^1 to processor $k-1$, and sends z_k^3 to processor $k+1$. In each preconditioning step, processor k sends y_k^1 to processor $k-1$, and sends y_k^3 to processor $k+1$. Hence, the partitioning objective of minimizing the overall separator size corresponds to minimizing the total communication volume. Furthermore, as mentioned in [14], minimizing the overall separator size corresponds to minimizing the upper bound on the number of iterations for convergence of the iterative method. Thus minimizing the overall separator size relates to minimizing the number of iterations for convergence.

4. Recursive graph bipartitioning model with fixed vertices. In this section, we show how we solve the oGPVS problem by utilizing the 2-way GPVS problem with fixed vertices within the RB paradigm.

4.1. Theoretical foundations. The following theorem and corollary lay down the basis for our formulation to obtain a K-way oVS of a given graph $G = (\mathcal{V}, \mathcal{E})$.

THEOREM 4.1. *For any disjoint vertex subset pair $\mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, G has a K-way oVS $\Pi_{oVS} = \{\langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle; \mathcal{S}\}$ such that $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $\mathcal{B}_R \subseteq \mathcal{S}_{K-1} \cup \mathcal{V}_K$ if and only if the distance between any two vertices $v_i \in \mathcal{B}_L$ and $v_j \in \mathcal{B}_R$ is at least $K-2$.*

Proof. (If) Consider the level structure $\{\mathcal{L}_0 = \mathcal{B}_L, \mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_i, \ldots\}$ rooted at the vertex subset $\mathcal{B}_L$, where $\mathcal{L}_i$ contains the vertices that have a shortest path distance of i to the vertices of $\mathcal{B}_L$. Since the shortest path distance between any vertex of $\mathcal{B}_L$ and any vertex of $\mathcal{B}_R$ is at least $K-2$, the vertices of $\mathcal{B}_R$ will be placed in levels $\mathcal{L}_{K-2}, \mathcal{L}_{K-1}, \mathcal{L}_K, \mathcal{L}_{K+1}, \ldots$. So we can construct a K-way oVS $\Pi_{\text{oVS}} = \{\langle \emptyset, \ldots, \emptyset \rangle; \langle \mathcal{L}_0, \ldots, \mathcal{L}_{K-3}, \bigcup_{k \geq K-2} \mathcal{L}_k \rangle\}$, where $\mathcal{V}_k = \emptyset$ (i.e., empty part) for $1 \leq k < K$, $\mathcal{S}_k = \mathcal{L}_{k-1}$ for $1 \leq k < K-1$, and $\mathcal{S}_{K-1} = \bigcup_{k \geq K-1} \mathcal{L}_{k-1}$. Since $\mathcal{B}_L = \mathcal{S}_1$, $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$. Due to the construction, $\mathcal{B}_R \subseteq \mathcal{V}_K \cup \mathcal{S}_{K-1}$ since $v_j \in \mathcal{S}_{K-1}$ for any $v_j \in \mathcal{B}_R$.

(Only If) Consider a K-way oVS such that $\mathcal{B}_L \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $\mathcal{B}_R \subseteq \mathcal{V}_K \cup \mathcal{S}_{K-1}$. Consider any vertex pair $v_i \in \mathcal{B}_L$ and $v_j \in \mathcal{B}_R$. It is clear that the minimum distance between v_i and v_j occurs when $v_i \in \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1}$. Due to the oVS structure, any path between a vertex of $\mathcal{S}_1$ and a vertex of $\mathcal{S}_{K-1}$ contains at least $K-3$ intermediate vertices one from each subseparator $\mathcal{S}_k$ (for $k = 2, 3, \ldots, K-2$). So, the minimum distance between v_i and v_j is at least $K-2$. ∎

COROLLARY 4.2. *A graph G has a K-way oVS if and only if the diameter of G is at least $K-2$.*

Algorithm 1 Initialization.

Require: Graph $G = (\mathcal{V}, \mathcal{E})$, integer K
1: Find a pseudoperipheral vertex v_L and a furthest vertex v_R from v_L
2: **if** distance between v_L and v_R is less than $K-2$ **then**
3: **return** "G is not partitionable into K-way oVS"
4: **else**
5: $\mathcal{B}_L \leftarrow \{v_L\}$
6: $\mathcal{B}_R \leftarrow \{v_R\}$
7: $\Pi_{oVS} \leftarrow$oGPVS$(G, \mathcal{B}_L, \mathcal{B}_R, K)$
8: **return** Π_{oVS}

Proof. G has a diameter of at least $K-2$ if and only if there exist two vertices v_i and v_j such that $\delta(v_i, v_j) \geq K-2$. Having two such vertices implies the existence of a K-way oVS of G such that $v_i \in \mathcal{V}_1 \cup \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1} \cup \mathcal{V}_K$ due to Theorem 4.1. On the other hand, by definition, if G has a K-way oVS, then there exist two vertices $v_i \in \mathcal{S}_1$ and $v_j \in \mathcal{S}_{K-1}$. Then Theorem 4.1 implies that $\delta(v_i, v_j) \geq K-2$. ∎

4.2. Recursive oGPVS algorithm. Theorem 4.1 and Corollary 4.2 give the necessary and sufficient conditions for finding a K-way oVS of a given graph $G = (\mathcal{V}, \mathcal{E})$. However, a new scheme needs to be applied during each RB step to satisfy the feasibility condition for the resulting K-way GPVS to be a K-way oVS. For this purpose, we propose a left-to-right bipartitioning approach together with a novel vertex fixation scheme so that a GPVS tool that supports partitioning with fixed vertices can be effectively and efficiently utilized. Algorithm 1 shows the initial invocation of the recursive oGPVS algorithm, where Algorithm 2 displays the basic steps of the proposed RB-based oGPVS algorithm that utilizes the proposed vertex fixation scheme.

The proposed oGPVS algorithm runs in $O((n+e)lgK)$-time, where each RB level runs in $O(n+e)$-time, under the assumption of using the successful multilevel graph partitioning tool MeTiS [15]. This running time is favorable compared to the running time $O(KlgK + e\sqrt{Kn})$ of the baseline algorithm proposed by Kahou, Grigori, and Sosonkina [12]. However, as mentioned in section 5.1, due to the lack of fixed vertexes support in graph partitioning tools, we implemented the hypergraph partitioning-based GPVS algorithm [6] in this work. In this implementation, the running time of each RB step can be as expensive as $O(\sum_{v_i \in \mathcal{V}} \deg(v_i)^2)$, where $\deg(v_i)$ denotes the degree of vertex v_i in $G(A)$. The high complexity of the operations in hypergraph partitioning mainly stems from the matching algorithm used in the coarsening phase of the hypergraph partitioning tool [4].

As seen in Algorithm 1, for the first RB step of the recursive oGPVS algorithm, $\mathcal{B}_L$ consists of a single pseudoperipheral vertex v_L which is found by using the pseudoperipheral node finder algorithm given in [11]. One of the vertices that has a maximum distance to the selected pseudoperipheral vertex is taken as the single vertex v_R constituting $\mathcal{B}_R$. According to Theorem 4.1, the oGPVS algorithm can be terminated at this initial stage if the shortest path distance between v_L and v_R is less than $K-2$. As seen in line 1 of Algorithm 2, the oGPVS function first checks whether the current bipartitioning is an intermediate or final level bipartitioning in the RB tree. Note that $K > 2$ for intermediate level bipartitionings, whereas $K = 2$ for final level bipartitionings, where K denotes the number of parts to be obtained from the current graph through further RB steps. As seen in line 3 of Algorithm 2, at the beginning of

Algorithm 2 oGPVS$(G, \mathcal{B}_L, \mathcal{B}_R, K)$.

Require: Graph $G = (\mathcal{V}, \mathcal{E})$, boundary vertex sets $\mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, integer K

1: **if** $K > 2$ **then**
2: $K' \leftarrow K/2$
3: $(\mathcal{F}_L, \mathcal{F}_R) \leftarrow$FIX-INT-LEVEL$(G, \mathcal{B}_L, \mathcal{B}_R, K')$
4: $\Pi_{VS} \leftarrow$GPVS$(G, \{\mathcal{F}_L, \mathcal{F}_R\}, 2)$ ▷ $\Pi_{VS} = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$
5: $G_L \leftarrow G[\mathcal{V}_L]$
6: $G_R \leftarrow G[\mathcal{V}_R]$
7: $\mathcal{B}_{LL} \leftarrow \mathcal{B}_L$
8: $\mathcal{B}_{LR} \leftarrow Adj(\mathcal{S}) \cap \mathcal{V}_L$
9: $\mathcal{B}_{RL} \leftarrow Adj(\mathcal{S}) \cap \mathcal{V}_R$
10: $\mathcal{B}_{RR} \leftarrow \mathcal{B}_R$
11: $\Pi^L_{oVS} \leftarrow$oGPVS$(G_L, \mathcal{B}_{LL}, \mathcal{B}_{LR}, K')$ ▷ $\Pi^L_{oVS} = \{\langle \mathcal{V}^L \rangle : \langle \mathcal{S}^L \rangle\}$
12: $\Pi^R_{oVS} \leftarrow$oGPVS$(G_R, \mathcal{B}_{RL}, \mathcal{B}_{RR}, K')$ ▷ $\Pi^R_{oVS} = \{\langle \mathcal{V}^R \rangle : \langle \mathcal{S}^R \rangle\}$
13: $\Pi_{oVS} \leftarrow \{\langle \mathcal{V}^L, \mathcal{V}^R \rangle : \langle \mathcal{S}^L, \mathcal{S}, \mathcal{S}^R \rangle\}$
14: **else**
15: $(G', \{v_\ell\}, \{v_r\}) \leftarrow$FIX-FINAL-LEVEL$(G, \mathcal{B}_L, \mathcal{B}_R)$
16: $\Pi_{VS} \leftarrow$GPVS$(G', \{\{v_\ell\}, \{v_r\}\}, 2)$ ▷ $\Pi_{VS} = \{\mathcal{V}'_L, \mathcal{V}'_R; \mathcal{S}\}$
17: $\mathcal{V}_L \leftarrow \mathcal{V}'_L - \{v_\ell\}$
18: $\mathcal{V}_R \leftarrow \mathcal{V}'_R - \{v_r\}$
19: $\Pi_{oVS} \leftarrow \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$
20: **return** Π_{oVS}

each intermediate RB step, the oGPVS function applies the proposed vertex fixation scheme by invoking the FIX-INT-LEVEL function on the current graph G with $\mathcal{B}_L$ and $\mathcal{B}_R$ to obtain the left and right fixed-vertex sets $\mathcal{F}_L$ and $\mathcal{F}_R$. Then in line 4, a 2-way GPVS is invoked on $(G, \{\mathcal{F}_L, \mathcal{F}_R\})$ to obtain $\Pi_{VS}(G) = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$, where $\mathcal{V}_L$ and $\mathcal{V}_R$ are used to denote the left and right parts. In lines 5 and 6, we construct left and right vertex-induced subgraphs $G_L = G[\mathcal{V}_L]$ and $G_R = G[\mathcal{V}_R]$ on which further RB steps will be applied, since this partitioning belongs to an intermediate level of the RB tree. Note that in order to construct G_L and G_R, we effectively apply the vertex removal scheme on the vertices of subseparator $\mathcal{S}$. That is, each subseparator vertex $v_s \in \mathcal{S}$ is removed during forming G_L and G_R.

In lines 7–10 of Algorithm 2, we determine left and right boundary vertices of both left and right subgraphs G_L and G_R. G_L and G_R, respectively, inherit their left and right boundary vertex sets from the left and right boundary vertex sets of the parent graph G. That is, the left boundary vertex set $\mathcal{B}_L$ of the current graph G becomes the left boundary vertex set $\mathcal{B}_{LL}$ of G_L, whereas the right boundary vertex set $\mathcal{B}_R$ of G becomes the right boundary vertex set $\mathcal{B}_{RR}$ of G_R. The boundary vertex sets $\mathcal{B}_{LR}$ and $\mathcal{B}_{RL}$, which are formed by the subseparator $\mathcal{S}$ of $\Pi_{VS}(G)$, respectively, constitute the right and left boundary vertex sets of G_L and G_R. That is, $Adj(\mathcal{S}) \cap \mathcal{V}_L$ constitutes the right boundary vertex set $\mathcal{B}_{LR}$ of G_L, whereas $Adj(\mathcal{S}) \cap \mathcal{V}_R$ constitutes the left boundary vertex set $\mathcal{B}_{RL}$ of G_R. We should note here that $\mathcal{S}$ will be the right subseparator of the rightmost vertex part and left subseparator of the leftmost vertex part obtained from RB trees rooted at G_L and G_R, respectively.

In lines 11 and 12 of Algorithm 2, we recursively invoke the oGPVS function on the left and right subgraphs G_L and G_R to, respectively, obtain Π^L_{oVS} and Π^R_{oVS}. Here, $\Pi^L_{oVS} = \{\langle \mathcal{V}^L \rangle : \langle \mathcal{S}^L \rangle\}$ denotes the resulting $K/2$-way oVS of the left subgraph

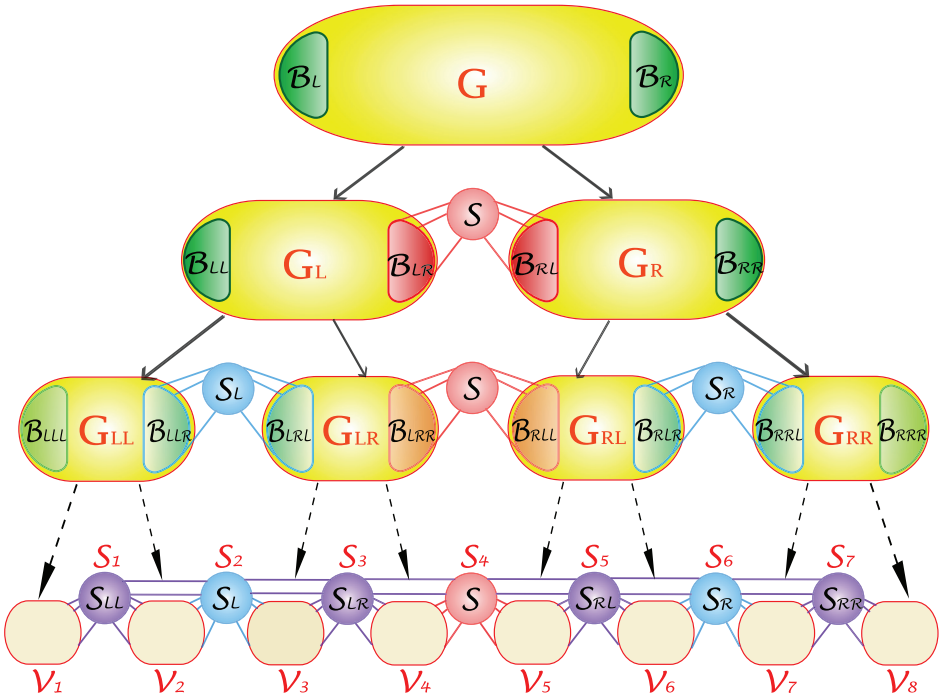

FIG. 4.1. *A three-level RB tree for producing an* 8*-way oVS of an initial graph* G.

G_L, where $\langle \mathcal{V}^L \rangle$ and $\langle \mathcal{S}^L \rangle$ denote the ordered $K/2$ vertex parts and $K/2-1$ subseparators. Similarly, $\Pi^R_{oVS} = \{\langle \mathcal{V}^R \rangle : \langle \mathcal{S}^R \rangle\}$ denotes the resulting $K/2$-way oVS of the right subgraph G_R, where $\langle \mathcal{V}^R \rangle$ and $\langle \mathcal{S}^R \rangle$, respectively, denote the ordered $K/2$ vertex parts and $K/2-1$ subseparators. Line 13 forms a K-way oVS of G by combining Π^L_{oVS} and Π^R_{oVS} together with the current level subseparator $\mathcal{S}$ as $\Pi_{oVS} = \{\langle \mathcal{V}^L, \mathcal{V}^R \rangle : \langle \mathcal{S}^L, \mathcal{S}, \mathcal{S}^R \rangle\}$.

For the final level bipartitionings (lines 15–19 in Algorithm 2), the oGPVS function applies the proposed vertex fixation scheme by invoking the FIX-FINAL-LEVEL function (in line 15) on the current graph G with $\mathcal{B}_L$ and $\mathcal{B}_R$ to obtain augmented graph G'. As will become clear later in Algorithm 4, G' is produced by adding two vertices v_L and v_R, which are, respectively, fixed to the left and right parts, and a number of associated edges to the current graph G. Then in line 16, a 2-way GPVS is invoked on $(G', \{\{v_L\}, \{v_R\}\})$ to obtain $\Pi_{VS}(G') = \{\mathcal{V}'_L, \mathcal{V}'_R; \mathcal{S}\}$. Lines 17–18 exclude v_L and v_R from the left and right vertex parts, respectively, to obtain the 2-way oVS in line 19.

Figure 4.1 displays a diagram of three levels of the RB process applied on a graph G with left and right boundary vertex sets $\mathcal{B}_L$ and $\mathcal{B}_R$. Solid directed edges connecting graphs to their subgraphs correspond to the edges of the RB tree. Note that $\mathcal{B}_L$ and $\mathcal{B}_R$, respectively, determine the left and right boundary vertex sets of the leftmost and rightmost graphs at each level of the RB tree rooted at G. That is, $\mathcal{B}_L = \mathcal{B}_{LL} = \mathcal{B}_{LLL}$ is the left boundary vertex set of graphs G, G_L, and G_{LL}, whereas $\mathcal{B}_R = \mathcal{B}_{RR} = \mathcal{B}_{RRR}$ is the right boundary vertex set of graphs G, G_R, and G_{RR}. The internal boundary vertex sets of the RB tree rooted at G are determined by the subseparators obtained, for example, $\mathcal{B}_{LRR} = \mathcal{B}_{LR} = Adj(\mathcal{S}) \cap \mathcal{V}_L$ and $\mathcal{B}_{RLL} = \mathcal{B}_{RL} =$

$Adj(\mathcal{S}) \cap \mathcal{V}_R$. The last level of Figure 4.1 shows the final 2-way GPVS operations performed on the subgraphs of the last level of the RB tree to obtain an 8-way oVS of the initial graph G.

As seen in Algorithm 2, we apply two different types of fixation schemes, FIX-INT-LEVEL and FIX-FINAL-LEVEL, for the intermediate level and final level bipartitionings, respectively. Here, an intermediate level bipartitioning refers to a 2-way GPVS to be applied on a graph at an internal node of the RB tree, whereas a final level bipartitioning refers to a 2-way GPVS to be applied on a graph at a leaf node.

The FIX-INT-LEVEL function invokes the FIX-VERTICES function twice with K' being equal to $K/2-1$, where K is the input of the current oGPVS function. Here, K' denotes the number of vertex levels to be fixed from the left and right boundary vertex sets—including the boundary vertex sets—of the current graph G. The FIX-VERTICES function utilizes a breadth-first search like algorithm to identify the vertices whose shortest path distances to a given vertex subset $\mathcal{B}$ are strictly less than a given K' value. The shortest path distance of a vertex v to a vertex subset $\mathcal{U}$ is defined as $\delta(v,\mathcal{U}) = \min_{u\in\mathcal{U}}\{\delta(u,v)\}$, where $\delta(u,v)$ denotes the shortest path distance between two vertices u and v. In the first invocation of the FIX-VERTICES function, vertices whose shortest path distances to $\mathcal{B}_L$ are strictly less than K' are fixed to the left part, whereas in the second invocation vertices whose shortest distances to $\mathcal{B}_R$ are strictly less than K' are fixed to the right part. That is, $\mathcal{F}_L = \{u : \delta(u,\mathcal{B}_L) < K'\}$ and $\mathcal{F}_R = \{u : \delta(u,\mathcal{B}_R) < K'\}$.

For the final level bipartitionings, the FIX-FINAL-LEVEL function augments graph G with two zero-weight vertices v_ℓ having $Adj(v_\ell) = \mathcal{B}_L$ and v_r having $Adj(v_r) = \mathcal{B}_R$ and fixes them to the left and right parts, respectively. This vertex fixation scheme introduces the flexibility of assigning the vertices of $\mathcal{B}_L$ and $\mathcal{B}_R$ to the subseparator.

Although the discussion given so far considers only exact power-of-two K values, the proposed oGPVS algorithm can be extended to non-power-of-two K values as follows: The bipartition at each recursion level is performed with left and right target part weights, respectively, proportional to $\lfloor K/2 \rfloor$ and $\lceil K/2 \rceil$, where K denotes the number of the parts to be obtained from the current graph through further RB steps. Then the vertices whose shortest path distances to $\mathcal{B}_L$ are strictly less than $\lfloor K/2 \rfloor - 1$ are fixed to the left part and the vertices whose shortest path distances to $\mathcal{B}_R$ are strictly less than $\lceil K/2 \rceil - 1$ are fixed to the right part.

4.3. A discussion on the correctness of oGPVS algorithm. We provide the following discussion on the correctness of the proposed RB-based oGPVS algorithm for exact power-of-two K values. The correctness discussion easily follows for non-power-of-two K values.

The left-to-right bipartitioning approach together with the proposed vertex fixation scheme adopted in the recursive oGPVS algorithm given in Algorithm 2 induces a natural ordering on both vertex parts and subseparators of a graph G in such a way that the final partition is a K-way oVS of G. We should also note that this scheme also induces a restricted 2^ℓ-way oVS at the ℓth level of the RB tree for $\ell = 0, 1, \ldots, lg_2K - 1$. Here the restriction refers to the nonadjacency of the consecutive subseparators. As will become clear later, 2-way GPVS operations to be invoked on the leaf level graphs of the RB tree make the consecutive subseparators adjacent in the final K-way oVS.

We include Figure 4.2 for a better understanding of the forthcoming discussion. Without loss of generality, let G be a graph in an intermediate level of the RB tree.

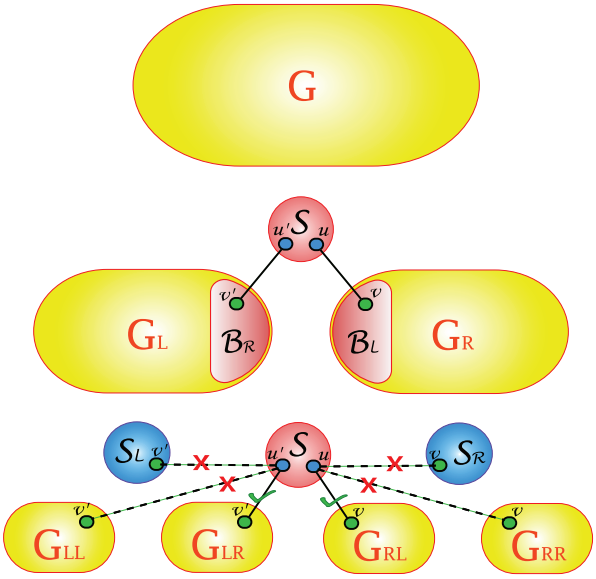

FIG. 4.2. *Restrictions for boundary vertices.*

Consider a 2-way vertex separator $\Pi_{VS}(G) = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$ of G and let G_L and G_R be the vertex-induced subgraphs by $\mathcal{V}_L$ and $\mathcal{V}_R$, respectively. Let $B_L = Adj(\mathcal{S}) \cap \mathcal{V}_R$ be the left boundary vertex set of G_R and $B_R = Adj(\mathcal{S}) \cap \mathcal{V}_L$ the right boundary vertex set of G_L.

For the sake of correctness of the oGPVS algorithm, the following restrictions should be maintained in any 2-way vertex separator $\Pi_{VS}(G_L)$ of G_L and $\Pi_{VS}(G_R)$ of G_R:

(a) If G_L and G_R are intermediate level graphs of the RB tree, the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R can only be assigned to the left part of $\Pi_{VS}(G_R)$, whereas the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L can only be assigned to the right part of $\Pi_{VS}(G_L)$.

(b) If G_L and G_R are final level graphs of the RB tree, the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R can be assigned to the subseparator as well as the left part of $\Pi_{VS}(G_R)$, whereas the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L can be assigned to the subseparator as well as the right part of $\Pi_{VS}(G_L)$.

We provide the following discussion for the need of restriction (a) on the assignment of the vertices in the left boundary vertex set $\mathcal{B}_L$ of G_R. Consider an edge $(u, v) \in \mathcal{E}(G)$, where $u \in \mathcal{S}$ and $v \in \mathcal{B}_L$ in $\Pi_{VS}(G)$. There are three cases according to the assignment of vertex v in $\Pi_{VS}(G_R) = \{\mathcal{V}_{RL}, \mathcal{V}_{RR}; \mathcal{S}\}$, namely, $v \in \mathcal{V}_{RL}$, $v \in \mathcal{V}_{RR}$, and $v \in \mathcal{S}_R$. Case $v \in \mathcal{V}_{RL}$ does not violate the oVS structure at the current level. Case $v \in \mathcal{S}_R$ makes two consecutive subseparators adjacent in the current level. Although this situation does not violate the oVS structure in the current level, it is guaranteed to violate the oVS structure in the subsequent bipartitions of the left and right subgraphs of G_R in the next level since these adjacent subseparators $\mathcal{S}$ and $\mathcal{S}_R$ will no longer be consecutive in the following levels. Case $v \in \mathcal{V}_{RR}$ immediately violates the oVS structure since edge (u, v) makes subseparator $\mathcal{S}$ connect two nonconsecutive vertex parts, namely, a vertex part in the current level oVS rooted

Algorithm 3 FIX-INT-LEVEL $(G, \mathcal{B}_L, \mathcal{B}_R, K')$.

Require: Graph $G = (\mathcal{V}, \mathcal{E}), \mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$, integer K'

1: $K' \leftarrow K' - 1$
2: $\mathcal{F}_L \leftarrow$FIX-VERTICES$(G, \mathcal{B}_L, K')$ ▷ fixing vertices to the left part
3: $\mathcal{F}_R \leftarrow$FIX-VERTICES$(G, \mathcal{B}_R, K')$ ▷ fixing vertices to the right part
4: **return** $(\mathcal{F}_L, \mathcal{F}_R)$

Algorithm 4 FIX-FINAL-LEVEL,$(G, \mathcal{B}_L, \mathcal{B}_R)$.

Require: Graph $G = (\mathcal{V}, \mathcal{E}), \mathcal{B}_L, \mathcal{B}_R \subseteq \mathcal{V}$

1: $\mathcal{V}' \leftarrow \mathcal{V} \cup \{v_\ell\} \cup \{v_r\}$
2: $\mathcal{E}' \leftarrow \mathcal{E} \cup \{(v_\ell, v) : v \in \mathcal{B}_L\} \cup \{(v, v_r) : v \in \mathcal{B}_R\}$
3: $w(v_\ell) \leftarrow w(v_r) \leftarrow 0$
4: $G' = (\mathcal{V}', \mathcal{E}')$
5: **return** $(G', \{v_\ell\}, \{v_r\})$

at G_L and the right vertex part of $\Pi_{VS}(G_R)$. A dual discussion holds for the need of restriction (a) on the assignment of the vertices in the right boundary vertex set $\mathcal{B}_R$ of G_L. In Figure 4.2, allowable and disallowable assignments of vertex v are identified by labeling the (u, v) edges with "✓" and "×".

The restriction (b) is a relaxed version of the restriction (a), where the vertices in $\mathcal{B}_L$ and $\mathcal{B}_R$ can also be assigned to the subseparator of $\Pi_{VS}(G_R)$ and $\Pi_{VS}(G_L)$, respectively. This relaxation is valid, because it has the potential of disturbing the oVS structure only if the left and right subgraphs of $\Pi_{VS}(G_L)$ and $\Pi_{VS}(G_R)$ are to be further bipartitioned, which is not the case since $\Pi_{VS}(G_L)$ and $\Pi_{VS}(G_R)$ are final level bipartitionings of the RB tree.

It is clear that the fixation scheme given in Algorithms 3 and 4 already achieves fixing the left and right boundary vertex sets in such a way to satisfy restrictions (a) and (b), respectively. Furthermore, at an intermediate level of the RB tree, Algorithm 3 fixes the vertices whose shortest path distances from the left and right boundary vertex sets are strictly less than $K' = K/2 - 1$ to the left and right parts, respectively, where K is the input of the current oGPVS function. Note that the shortest path distance between any two vertices in $\mathcal{B}_L$ and $\mathcal{B}_R$ is at least $K - 2$ due to this additional vertex fixing. So, this additional vertex fixing ensures that the vertex sets that are fixed to the left and right parts are disjoint and there always exists a free vertex on any path from a vertex fixed to the left part to a vertex fixed to the right part. This in turn ensures the existence of a valid vertex separator for partitioning the current graph.

This additional vertex fixing is also needed to guarantee that a K-way oVS will be obtained from RB-based partitioning of the left and right subgraphs according to Theorem 4.1 because of the following reasons. The above-mentioned fixing to the left part ensures that the shortest path distance between any two vertices $v_h \in \mathcal{B}_L$ and $v_i \in \mathcal{S}$ is at least $K' = K/2 - 1$ in the following $\Pi_{VS} = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$. In other words, the shortest path distance between any two vertices $v_h \in \mathcal{B}_{LL} = \mathcal{B}_L$ and $v_j \in B_{LR} = Adj(S) \cap \mathcal{V}_L$ will be at least $K/2 - 2$, where B_{LL} and B_{LR} are the left and right boundary vertex sets of left subgraph G_L, respectively. Then G_L has a $(K/2)$-way oVS such that $B_{LL} \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ and $B_{LR} \subseteq \mathcal{V}_{K/2} \cup \mathcal{S}_{K/2-1}$ by Theorem 4.1.

A similar discussion also holds for fixing to the right part, and consequently for the right subgraph G_R. Combining these two $(K/2)$-way oVS partitions of the left and right subgraphs G_L and G_R gives a K-way oVS for the original graph G by placing the subseparator $\mathcal{S}$ (as $\mathcal{S}_{K/2}$) in between the rightmost vertex part of the left oVS and the leftmost vertex part of the right oVS. Note that having $\mathcal{B}_{LR} \subseteq \mathcal{V}_{K/2} \cup \mathcal{S}_{K/2-1}$ for the left $(K/2)$-way oVS does not violate the final K-way oVS of G, but makes consecutive subseparators adjacent via the vertices in $\mathcal{B}_{LR} \cap \mathcal{S}_{K/2-1}$. A dual discussion holds for having $\mathcal{B}_{RL} \subseteq \mathcal{V}_1 \cup \mathcal{S}_1$ for the right $K/2$-way oVS.

4.4. A better load balancing scheme. The vertex weighting scheme adopted in the above-mentioned RB-based oGPVS algorithm does not totally encapsulate the nonzero counts of the diagonal blocks in balancing the part weights as discussed in section 3.2. For the sake of a better load balancing in the A-to-A_{BDO} permutation, we enhance our RB-based oGPVS algorithm as follows. Consider a 2-way vertex separator $\Pi_{VS}(G) = \{\mathcal{V}_L, \mathcal{V}_R; \mathcal{S}\}$ of the current graph G. After forming the left and right vertex-induced subgraphs G_L and G_R, we add two isolated vertices s_L and s_R to G_L and G_R, with weights

$$w(s_L) = \sum_{s \in \mathcal{S}} |Adj(s) \cap (\mathcal{S} \cup \mathcal{V}_L)| \quad \text{and} \tag{4.1}$$

$$w(s_R) = \sum_{s \in \mathcal{S}} |Adj(s) \cap (\mathcal{S} \cup \mathcal{V}_R)|, \tag{4.2}$$

respectively. Then we fix s_L to the right part of G_L and s_R to the left part G_R.

We provide the following discussion to show how the proposed enhancement leads to better load balancing. It is clear that $\mathcal{S}$ will be the right subseparator of the rightmost part of the oVS to be obtained from the RB-tree rooted at G_L. Without loss of generality, let this rightmost part be $\mathcal{V}_k$, which means that $\mathcal{S}$ will be $\mathcal{S}_k$. Note that vertex s_L fixed to G_L will remain as fixed to all rightmost graphs of the RB-tree rooted at G_L, and hence it will contribute its weight $w(s_L)$ to $W(\mathcal{V}_k)$. Because of (4.1), contribution of $w(s_L)$ to the weight of part $\mathcal{V}_k$ makes $W(\mathcal{V}_k)$ encapsulate the nonzero counts of submatrices $C^T_{k-1,k}, A^T_{k,k+1}$, and $C_{k,k}$ in modeling the nonzero count of diagonal block D_k given in (3.1). In a dual manner, $w(s_R)$ contributes to the weight of part $\mathcal{V}_{k+1}$, and this contribution makes $W(\mathcal{V}_{k+1})$ encapsulate the nonzero counts of submatrices $C_{k,k}, A_{k+1,k}$, and $C_{k,k+1}$ in modeling the nonzero count of diagonal block D_{k+1}. Hence, this discussion can be generalized to show that $W(\mathcal{V}_k) = nnz(D_k)$ for each part $\mathcal{V}_k$.

5. Experiments.

5.1. A GPVS implementation that supports fixed vertices. Currently, existing GPVS tools such as onmetis [15] do not support fixed vertices. So we utilized the hypergraph partitioning (HP) based GPVS formulation proposed in [6] based on the existence of a number of HP tools such as PaToH [5], Zoltan [3], and hMeTiS [16] that support fixed vertices. Here, we briefly summarize the HP-based GPVS formulation [6] utilized in our experimentation and describe how the vertex fixing scheme is implemented in the HP model.

First, we briefly review hypergraph and hypergraph partitioning for the sake of completeness. A hypergraph $H = (\mathcal{U}, \mathcal{N})$ is defined as a set $\mathcal{U}$ of nodes and a set $\mathcal{N}$ of nets (hyperedges) among those nodes. We use nodes for referring to the vertices of a hypergraph in order to avoid the confusion between the vertices of a graph and a hypergraph. Every net $n_i \in \mathcal{N}$ connects a subset of nodes, i.e., $n_i \subseteq \mathcal{U}$. The graph

is a special instance of hypergraph such that each net connects exactly two nodes. $\Pi_{\mathcal{U}} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ is a K-way node partition of H if parts are pairwise disjoint and exhaustive. In a node partition $\Pi_{\mathcal{U}}$ of H, n_i is said to be an internal net of node part $\mathcal{U}_k$ if all nodes that are connected by n_i belong to $\mathcal{U}_k$. n_i is called a cut-net (external) if the nodes that are connected by n_i belong to at least two different parts. In the HP problem, the objective is to minimize the number of cut-nets, whereas the partitioning constraint is to maintain a balance on the part weights. Node-part weight is usually defined as the sum of the weights of the nodes in a part as in the definition given in (2.3) for vertex-part weight used in graph partitioning.

The HP-based GPVS formulation of [6] relies on finding an edge clique cover on a given graph G, then using this clique cover to construct a clique-node hypergraph H, and finally partitioning H. Among the three edge clique covers investigated in [6], we implemented the basic one, which is referred to as the 2-clique cover. In this basic scheme, each edge $e_{i,j}$, which is a 2-clique of G, induces a node $u_{i,j} \in \mathcal{U}$ of degree 2 in H, whereas each vertex v_h of G induces a net n_h in H. Net n_h connects all nodes corresponding to the edges that are incident to v_h in G.

A K-way node partition $\Pi_{\mathcal{U}}(H)$ of H is decoded as inducing a K-way vertex separator $\Pi_{VS}(G)$ of G as follows: The internal nets of a node part $\mathcal{U}_k$ of $\Pi_{\mathcal{U}}$ constitute the vertices of a vertex part $\mathcal{V}_k$ of Π_{VS}, whereas the external nets of $\Pi_{\mathcal{U}}$ constitute the vertices of the separator of Π_{VS}.

It is shown in [6] that the partitioning objective of minimizing the number of cut-nets corresponds to minimizing the number of separator vertices. It is also shown that the partitioning constraint of balancing on the number of internal nets of node parts infers balance on the vertex counts of vertex parts. So, in HP-based GPVS formulation, although the partitioning objective exactly matches the partitioning objective of oGPVS formulation, the partitioning constraint does not match the partitioning constraint of oGPVS formulation. Since nodes of H correspond to the edges of G, balance on the vertices of G cannot be directly enforced during the partitioning of H. We propose the following node weighting scheme for the clique-node hypergraph H so that the weight of a node part in $\Pi_{\mathcal{U}}$ is as close as possible to the weight of the respective vertex part in Π_{VS}. The weight $w(v_h)$ of a vertex in G is evenly distributed among the nodes that are connected by net n_h in H. That is, each vertex v_h of G contributes $w(v_h)/|n_h|$ to all nodes that are connected by n_h, where $|n_h|$ denotes the degree of net n_h. Hence, the weight of a hypergraph node $u_{i,j}$ is defined as follows in terms of weights of graph vertices v_i and v_j:

$$w(u_{i,j}) = \frac{w(v_i)}{|n_i|} + \frac{w(v_j)}{|n_j|}. \tag{5.1}$$

It can be shown that node-part weight $\mathcal{W}(\mathcal{U}_k)$ of $\mathcal{U}_k$ in $\Pi_{\mathcal{U}}$ will be equal to vertex-part weight $\mathcal{W}(\mathcal{V}_k)$ of $\mathcal{V}_k$ in Π_{VS} if node part $\mathcal{U}_k$ has no external nets. However, external nets of a node part $\mathcal{U}_k$ of $\Pi_{\mathcal{U}}$ will make $\mathcal{W}(\mathcal{U}_k)$ smaller than $\mathcal{W}(\mathcal{V}_k)$. Since the node-part weights of different parts of $\Pi_{\mathcal{U}}$ will involve similar errors, the proposed method can be expected to infer a sufficiently good balance on the vertex-part weights of Π_{VS}.

Since the above-mentioned HP-based GPVS implementation is used within the RB framework, we will now discuss how the graph-vertex fixation scheme is handled during bipartitioning a clique-node hypergraph H into left and right parts. Fixing a vertex v_i to the left part of $\Pi_{VS}(G)$ corresponds to enforcing the corresponding net n_i to be an internal net of the left part $\mathcal{U}_L$ of $\Pi_{\mathcal{U}}(H) = \{\mathcal{U}_L, \mathcal{U}_R\}$. Enforcing n_i to

be an internal net of $\mathcal{U}_L$ can only be achieved by fixing all nodes that are connected by n_i to $\mathcal{U}_L$. A similar discussion holds for fixing a vertex to the right part of Π_{VS}.

5.2. Experimental results. We have tested the performance of the proposed oGPVS algorithm on a wide range of square sparse matrices of the University of Florida (UFL) Sparse Matrix Collection [9]. We excluded the small matrices that have fewer than 1,000 rows/columns for the sake of sufficiently coarse-grained parallel processing. We also excluded matrices that have more than 10,000,000 rows/columns since we used a sequential partitioning environment. For the sake of simplicity, we considered only the matrices whose corresponding graphs are connected. There were 237 matrices in the UFL collection satisfying these properties at the time of experimentation. We tested with $K \in \{8, 16, 32, 64, 128, 256\}$. For a given K value, a K-way A-to-A_{BDO} permutation of a test matrix constitutes a permutation instance. The permutation instances in which $N < 100 \times K$ were discarded, as the diagonal blocks D_k's would become too small to be meaningful for parallel processing (e.g., fewer than 100 rows/columns per processor).

To our knowledge, the algorithm proposed by Kahou, Grigori, and Sosonkina [12], which is described in section 1, is the only work introduced in the literature for solving the A-to-A_{BDO} permutation problem. So we compared the performance of our oGPVS algorithm against this baseline algorithm. For unsymmetric matrices, matrix A is symmetrized with $A + A^T$ in both the baseline and oGPVS algorithms. Since the first step of both algorithms is to find a pseudoperipheral vertex, we ran the pseudoperipheral node finder algorithm [11] once on the standard graph representation of each matrix and used the root vertex of the resulting level structure in both algorithms. For a given K value, the RB process is terminated if the length of the level structure is less than K, since the graph cannot be partitioned into K parts by the baseline algorithm, whereas it cannot be partitioned by the oGPVS algorithm if the length of the level structure rooted at the pseudoperipheral vertex is less than $K-1$. So, such partitioning instances are discarded from the results of both of these algorithms to make the comparison meaningful.

As a result of the former selection criteria and the latter feasibility criteria, the experiments are conducted for a total of 880 permutation instances (237, 220, 173, 125, 80, and 45 instances for 8-, 16-, 32-, 64-, 128-, and 256-way permutations, respectively). In addition, neither algorithm guarantees the nonemptiness of the parts in the resulting oVS, although the length of the level structure is larger than K. Hence, any partitioning instance, for which at least one algorithm yields a partition with an empty part, is discarded from the results of both of these algorithms for the sake of a fair comparison. Note that an empty part $\mathcal{V}_k$ in an oVS corresponds to the fact that there exists no row/column in $A_{k,k}$ of the diagonal block D_k in the permuted matrix induced by oVS. The baseline algorithm fails on 7, 13, 17, 22, 35, and 38 percent of the remaining test matrices due to empty parts for 8-, 16-, 32-, 64-, 128-, and 256-way permutations, respectively. The oGPVS algorithm fails on 21, 34, 41, 50, 46, and 46 percent of the remaining test matrices due to empty parts for 8-, 16-, 32-, 64-, 128-, and 256-way permutations, respectively. So, experimental results are reported for a total of 569 permutation instances (183, 155, 106, 63, 38, and 24 instances for 8-, 16-, 32-, 64-, 128-, and 256-way permutations, respectively).

For the bipartitioning of clique-node hypergraphs, we used the HP tool PaToH with default parameters of PATOH_SUGPARAM_SPEED (see PaToH manual [5]) recommended for faster partitionings except for the coarsening algorithm. As the coarsening algorithm, we used scaled heavy connectivity matching (SHCM) instead

of absorption clustering using nets (ABSHCC). Our experimental results showed that SHCM leads to considerably smaller overlap sizes than ABSHCC.

As PaToH involves randomized algorithms, we obtained 10 different partitions for each partitioning instance of the oGPVS algorithm, and the geometric averages of the load imbalance and separator size values over 10 resulting partitions are reported as the representative result for the oGPVS method on that particular partitioning instance. In all oGPVS partitioning instances, the maximum allowable imbalance ratio ϵ in (2.2) is set to 0.10.

Table 5.1 displays the performance comparison of the proposed oGPVS algorithm against the baseline algorithm in terms of percent load imbalance and percent overlap size ratio for 8-, 16-, 32-, 64-, 128-, and 256-way A-to-A_{BDO} permutation problems. As seen in the second column of Table 5.1, matrices are categorized according to their type, where each type represents a different problem domain, and the average results of each type of problem are given for each K value. In the third column we display the number of matrices that belong to the corresponding problem type, where we included the results only for the types of problem that contain three or more resulting partitions for the respective K value.

In Table 5.1, the percent load imbalance value of a permutation is computed as $100 \times (Z_{max} - Z_{avg})/Z_{avg}$, where Z_{max} denotes the nonzero count of the diagonal block with maximum nonzero count and Z_{avg} denotes the average nonzero count of diagonal blocks. The percent overlap size ratio of a permutation is computed as $100 \times N_c/N$. For a better relative performance comparison of these two algorithms in terms of overlap size, Table 5.1 also displays the N_c^o/N_c^b values which denote ratios of the overlap sizes of the permutations found by the oGPVS algorithm to those of the baseline algorithm. Note that $N_c^o/N_c^b < 1$ indicates that the oGPVS algorithm performs better than the baseline algorithm in terms of overlap size. Table 5.2 summarizes the overall permutation results as averages over different K values.

As seen in Table 5.2, on average the baseline algorithm achieves better load balance than the oGPVS algorithm for $K \in \{8, 16, 128, 256\}$, whereas the oGPVS algorithm achieves better load balance than the baseline algorithm for $K \in \{32, 64\}$. This finding can be attributed to the fact that the baseline algorithm pays more attention to load balancing by identifying the separators after the partition is balanced enough and refining the separators only if the refinement does not produce a more imbalanced partition.

We will now discuss the relative performance of the baseline algorithm and the proposed oGPVS algorithm in terms of total overlap size. In 8-way and 16-way permutations, the oGPVS algorithm performs better than the baseline algorithm in almost all problem types (in 13 out of 13 and in 11 out of 12 types, respectively). In 32-way permutations, the oGPVS algorithm performs better than the baseline algorithm in 4 out of 10 problem types; both algorithms perform nearly the same in 2 problem types, whereas the baseline algorithm performs better in the remaining 4 problem types. In 64-way permutations, the oGPVS algorithm performs better than the baseline algorithm in 3 out of 7 problem types, whereas the baseline algorithm performs better in the remaining 4 problem types. In 128-way and 256-way permutations, the oGPVS algorithm performs better than the baseline algorithm in 1 out of 3 and 2 out of 2 problem types, respectively. In general, the oGPVS algorithm performs drastically better than the baseline algorithm in such problem types as circuit simulation, power network, and undirected graphs. As seen in Table 5.2, on average, the oGPVS algorithm produces matrices in BDO form with 30%, 30%, 23%, 23%, 26%, and

TABLE 5.1

Performance comparison in terms of load imbalance and total overlap size ratio as averages over problem kinds.

		# of	Baseline algorithm		oGPVS algorithm		oGPVS vs. base
K	Problem type	matrices	Imbal.	N_c/N	Imbal.	N_c/N	N_c^o/N_c^b
	2D/3D	18	3.23%	4.47%	3.90%	4.02%	0.90
	circuit simulation	6	4.01%	4.36%	4.51%	1.57%	0.36
	computational fluid dynamics	21	7.05%	10.33%	6.13%	7.37%	0.71
	directed graph	12	61.34%	31.03%	43.03%	26.96%	0.87
	economic	6	24.73%	24.33%	42.30%	18.46%	0.76
	electromagnetics	10	3.44%	5.52%	7.52%	4.65%	0.84
8	model reduction	12	3.77%	6.20%	4.83%	5.79%	0.93
	optimization	11	0.95%	1.16%	0.45%	1.02%	0.88
	power network	3	12.99%	20.89%	5.09%	1.87%	0.09
	semiconductor device	10	13.11%	23.87%	20.49%	22.52%	0.94
	structural	30	5.66%	9.08%	9.71%	8.69%	0.96
	thermal	4	2.67%	2.98%	3.76%	2.81%	0.94
	undirected graph	29	2.14%	2.31%	10.42%	0.77%	0.33
	2D/3D	16	5.96%	7.83%	5.06%	7.03%	0.90
	circuit simulation	5	3.52%	5.20%	4.03%	1.81%	0.35
	computational fluid dynamics	18	14.11%	17.77%	9.90%	14.14%	0.80
	directed graph	9	147.09%	38.16%	94.77%	31.73%	0.83
	electromagnetics	9	6.74%	12.31%	12.27%	11.75%	0.95
16	model reduction	11	8.73%	11.91%	6.45%	11.09%	0.93
	optimization	11	2.17%	2.43%	0.86%	2.14%	0.88
	power network	3	40.23%	41.49%	14.87%	8.05%	0.19
	semiconductor device	7	31.43%	37.32%	21.28%	39.14%	1.05
	structural	22	10.12%	14.17%	11.90%	12.37%	0.87
	thermal	4	5.81%	6.31%	7.28%	6.02%	0.95
	undirected graph	28	5.48%	3.87%	13.77%	1.37%	0.36
	2D/3D	15	9.68%	14.01%	8.80%	15.33%	1.09
	circuit simulation	5	8.07%	10.59%	11.57%	4.88%	0.46
	computational fluid dynamics	10	15.22%	20.10%	9.51%	22.45%	1.12
	directed graph	3	98.18%	36.87%	61.59%	23.43%	0.64
32	electromagnetics	3	6.99%	7.71%	2.93%	8.25%	1.07
	model reduction	8	13.72%	14.50%	10.47%	14.60%	1.01
	optimization	11	5.35%	4.95%	2.51%	4.96%	1.00
	structural	15	18.08%	22.25%	16.15%	19.29%	0.87
	thermal	4	11.51%	12.86%	13.39%	13.49%	1.05
	undirected graph	21	4.31%	3.32%	9.28%	1.22%	0.37
	2D/3D	8	11.71%	14.23%	8.35%	14.84%	1.04
	circuit simulation	3	9.44%	11.84%	7.74%	6.32%	0.53
	computational fluid dynamics	5	14.68%	19.62%	7.15%	23.60%	1.20
64	model reduction	5	15.80%	18.67%	13.88%	19.42%	1.04
	optimization	9	6.35%	6.34%	3.42%	7.05%	1.11
	structural	6	29.80%	34.46%	24.59%	34.02%	0.99
	undirected graph	20	8.59%	5.72%	13.03%	2.42%	0.42
	2D/3D	6	25.51%	21.76%	77.84%	29.10%	1.34
128	optimization	5	4.57%	3.34%	6.03%	3.56%	1.06
	undirected graph	15	9.30%	6.54%	21.02%	2.70%	0.41
256	optimization	3	2.36%	1.41%	2.83%	1.34%	0.95
	undirected graph	12	14.61%	10.45%	24.50%	4.06%	0.39

TABLE 5.2
Performance comparison in terms of load imbalance and total overlap size as averages over K values.

	# of	Baseline algorithm		oGPVS algorithm		oGPVS vs. base
K	matrices	Imbal.	N_c/N	Imbal.	N_c/N	N_c^o/N_c^b
8	183	5.13%	6.55%	7.42%	4.60%	0.70
16	155	9.07%	9.42%	9.57%	6.62%	0.70
32	106	9.45%	9.91%	9.38%	7.63%	0.77
64	63	10.26%	9.63%	9.34%	7.39%	0.77
128	38	10.89%	8.27%	22.22%	6.11%	0.74
256	24	12.64%	9.13%	17.86%	5.46%	0.60

TABLE 5.3
The effect of the better load balancing (bb) scheme in the performance of the oGPVS algorithm.

	# of	oGPVS-w/o-bb		oGPVS	
K	matrices	Imbal.	N_c/N	Imbal.	N_c/N
8	183	9.56%	4.27%	7.42%	4.60%
16	155	11.49%	6.20%	9.43%	6.50%
32	106	10.55%	7.22%	9.13%	7.61%
64	63	10.66%	7.31%	9.34%	7.39%
128	43	25.54%	8.00%	24.18%	7.73%
256	26	20.82%	6.64%	19.85%	6.56%

40% smaller overlap size than the baseline algorithm for 8-, 16-, 32-, 64-, 128-, and 256-way A-to-A_{BDO} permutations, respectively.

We provide Table 5.3 to show the average success of the better load balancing (bb) scheme (described in section 4.4) in improving the load balancing performance of the oGPVS algorithm. In this table, oGPVS-w/o-bb refers to the oGPVS algorithm that does not utilize the bb scheme, whereas oGPVS refers to the oGPVS algorithm that utilizes the bb scheme. We should note here that the performance results given in Tables 5.1 and 5.2 are obtained by running the latter one. As seen in Table 5.3, the bb scheme considerably improves the load balancing performance of the oGPVS algorithm at the expense of slightly degrading the overlap-size performance of oGPVS. Because of this trade-off between the two schemes, oGPVS-w/o-bb can be recommended instead of oGPVS only when the workload associated with the diagonal blocks cannot be precisely defined.

6. Conclusion. We examined symmetrically permuting a sparse square matrix A into a K-way block diagonal form A_{BDO} with overlap. The permutation objective is to minimize the total overlap size, whereas the permutation constraint is to maintain balance on the nonzero counts of diagonal blocks. We defined the ordered graph partitioning by vertex separator (oGPVS) problem, which is a constrained version of the GPVS problem, and showed that the A-to-A_{BDO} permutation problem can be modeled as an oGPVS problem on the standard graph representation of matrix A. The existing graph partitioning tools do not solve the oGPVS problem. We proposed a left-to-right bipartitioning method that utilizes a novel vertex fixation scheme for the recursive bipartitioning (RB) framework. The proposed RB-based method enables the use of existing 2-way GPVS tools that support fixed vertices for solving the oGPVS problem and hence the A-to-A_{BDO} permutation problem. We have tested the performance of the proposed A-to-A_{BDO} permutation problem on 237

square sparse matrices selected from the UFL matrix collection. Experimental results reported on 569 permutation instances for $K \in \{8, 16, 32, 64, 128, 256\}$ confirmed the validity of the proposed model and method.

As a future work, one-way dissection ordering, which is widely used for sparse direct solvers, can be considered as a potential application of the proposed oGPVS approach. The consecutive subseparators are not adjacent in the one-way dissection ordering, whereas they are adjacent in the oGPVS ordering. Due to this difference, the proposed oGPVS algorithm should be enhanced with a more restricted vertex fixation scheme in order to produce one-way dissection orderings.

REFERENCES

[1] C. Aykanat, B. B. Cambazoglu, F. Findik, and T. Kurc, *Adaptive decomposition and remapping algorithms for object-space-parallel direct volume rendering of unstructured grids*, J. Parallel Distrib. Comput., 67 (2007), pp. 77–99.

[2] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek, *Permuting sparse rectangular matrices into block-diagonal form*, SIAM J. Sci. Comput., 25 (2004), pp. 1860–1879.

[3] E. Boman, K. Devine, L. A. Fisk, R. Heaphy, B. Hendrickson, C. Vaughan, Ü. V. Çatalyürek, D. Bozdag, W. Mitchell, and J. Teresco, *Zoltan* 3.0: *Parallel Partitioning, Load-balancing, and Data Management Services: User's Guide*, Sandia National Laboratories, Albuquerque, NM, 2007.

[4] Ü. V. Çatalyürek and C. Aykanat, *Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication*, IEEE Trans. Parallel Distrib. Comput., 10 (1999), pp. 673–693.

[5] Ü. V. Çatalyürek and C. Aykanat, *PaToH: A Multilevel Hypergraph Partitioning Tool, Version* 3.0, Bilkent University, Department of Computer Engineering, Ankara, Turkey, 1999; PaToH is available online from http://bmi.osu.edu/∼umit/software.htm.

[6] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, *Hypergraph partitioning-based fill-reducing ordering for symmetric matrices*, SIAM J. Sci. Comput., 33 (2011), pp. 1996–2023.

[7] Ü. V. Çatalyürek, E. G. Boman, K. D. Devine, D. Bozdağ, R. T. Heaphy, and L. A. Riesen, *A repartitioning hypergraph model for dynamic load balancing*, J. Parallel Distrib. Comput., 69 (2009), pp. 711–724.

[8] A. Cevahir, C. Aykanat, A. Turk, and B. B. Cambazoglu, *Site-based partitioning and repartitioning techniques for parallel PageRank computation*, IEEE Trans. Parallel Distrib. Systems, 22 (2011), pp. 786–802.

[9] T. Davis, *The University of Florida Sparse Matrix Collection*, Tech. Report REP-2007-298, CISE Department, University of Florida, Gainesville, FL, 2007.

[10] M. C. Ferris and J. D. Horn, *Partitioning mathematical programs for parallel solution*, Math. Programming, 80 (1998), pp. 35–61.

[11] A. George and J. W. H. Liu, *An implementation of a pseudoperipheral node finder*, ACM Trans. Math. Softw., 5 (1979), pp. 284–295.

[12] G. A. A. Kahou, L. Grigori, and M. Sosonkina, *A partitioning algorithm for block-diagonal matrices with overlap*, Parallel Comput., 34 (2008), pp. 332–344.

[13] G. A. A. Kahou, E. Kamgnia, and B. Philippe, *Parallel implementation of an explicit formulation of the multiplicative Schwarz preconditioner*, in Proceedings of the 17th IMACS World Congress: Scientific Computation, Applied Mathematics and Simulation, Paris, CD-ROM, 2005.

[14] G. A. A. Kahou, E. Kamgnia, and B. Philippe, *An explicit formulation of the multiplicative Schwarz preconditioner*, Appl. Numer. Math., 57 (2007), pp. 1197–1213.

[15] G. Karypis and V. Kumar, *MeTiS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version* 4.0, Department of Comp. Sci. and Eng., Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.

[16] G. Karypis, V. Kumar, R. Aggarwal, and S. Shekhar, *hMeTiS: A Hypergraph Partitioning Package, Version* 1.0.1, Department of Comp. Sci. and Eng., Army HPC Research Center, University of Minnesota, Minneapolis, MN, 1998.

[17] J. W. H. Liu, *A graph partitioning algorithm by node separators*, ACM Trans. Math. Softw., 15 (1989), pp. 198–219.

[18] M. Naumov, M. Manguoglu, and A. H. Sameh, *A tearing-based hybrid parallel sparse linear system solver*, J. Comput. Appl. Math., 234 (2010), pp. 3025–3038.
[19] M. Naumov and A. H. Sameh, *A tearing-based hybrid parallel banded linear system solver*, J. Comput. Appl. Math., 226 (2009), pp. 306–318.
[20] A. Pinar and C. Aykanat, *Fast optimal load balancing algorithms for* $1D$ *partitioning*, J. Parallel Distrib. Comput., 64 (2004), pp. 974–996.
[21] A. Pothen and C.-J. Fan, *Computing the block triangular form of a sparse matrix*, ACM Trans. Math. Softw., 16 (1990), pp. 303–324.