*oGPVS/BDO*: A Software Package for *Ordered Graph Partitioning by Vertex Separators* and *Permuting Matrices into Block Diagonal Form with Overlap*

Seher Acer  
Computer Engineering Department  
Bilkent University  
Ankara, 06533 Turkey  
acer@cs.bilkent.edu.tr

Enver Kayaaslan  
Computer Engineering Department  
Bilkent University  
Ankara, 06533 Turkey  
enver@cs.bilkent.edu.tr

Cevdet Aykanat  
Computer Engineering Department  
Bilkent University  
Ankara, 06533 Turkey  
aykanat@cs.bilkent.edu.tr

June 28, 2013
1 Introduction

A matrix in \( K \)-way block diagonal form with overlap (BDO form) has \( K \) possibly-overlapping diagonal blocks \( D_1, D_2, \ldots, D_K \). Figure 1 displays an illustration of BDO form.

Figure 1: Block diagonal form with overlap

\[
A_{BDO} = \begin{bmatrix}
A_{1,1} & A_{1,2} & & & \\
A_{1,2}^T & C_{1,1} & A_{2,1} & C_{1,2} & \\
& A_{2,1}^T & A_{2,2} & A_{2,3} & \\
& & C_{2,2}^T & A_{2,3} & & \\
& & & \vdots & & \\
& & & & C_{K-1,K-1} & \ A_{K,K-1}^T & \\
& & & & \ & & \ & C_{K,K}^T \\
\end{bmatrix}
\]  

(1)

A formal representation of a symmetric matrix in \( K \)-way BDO form can be found in (1), where

\[
D_k = \begin{bmatrix}
C_{k-1,k-1} & A_{k,k-1} & C_{k-1,k} \\
A_{k,k-1}^T & A_{k,k} & A_{k,k+1} \\
C_{k-1,k}^T & A_{k,k+1}^T & C_{k,k}
\end{bmatrix}
\]

for \( k = 2, 3, \ldots, K \) (2)

\[
D_1 = \begin{bmatrix}
A_{1,1} & A_{1,2} \\
A_{1,2}^T & C_{1,1}
\end{bmatrix}, \quad \text{and} \quad
D_K = \begin{bmatrix}
C_{K-1,K-1} & A_{K,K-1} \\
A_{K,K-1}^T & A_{K,K}
\end{bmatrix}
\]

(3)

In (2), \( C_{k,k} \) denotes the coupling diagonal block between \( D_k \) and \( D_{k+1} \), for \( k = 1, 2, \ldots, K-1 \).

The \( A \)-to-\( A_{BDO} \) permutation problem if defined as symmetrically permuting the rows and columns of a sparse square matrix so that the permuted matrix is in a \( K \)-way BDO form. The permutation objective is to minimize the total overlap size, which is defined as \( N_c = \sum_{k=1}^{K-1} n_c^k \), where \( n_c^k \) denotes the number of the rows/columns of the coupling diagonal block \( C_{k,k} \). The permutation constraint is to maintain balance on the number of nonzeros in diagonal blocks \( D_k \)'s. This permutation problem
arises in the parallelization of an explicit formulation of the multiplicative Schwarz preconditioner [6] and a more recent domain decomposition method proposed by Naumov et al. [7, 8].

We solve the $A$-to-$A_{BDO}$ permutation problem by solving the ordered Graph Partitioning by Vertex Separator (oGPVS) problem on the standard graph representation of $A$. Section 3 provides detailed discussion on the oGPVS problem definition and the formulation of an $A$-to-$A_{BDO}$ problem as an oGPVS problem.

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 $G(A) = (V, E)$ with $N$ vertices. Vertex set $V$ and edge set $E$ respectively represent the rows/columns and off-diagonal nonzeros of matrix $A$. For each row/column $r_i/c_i$, $V$ contains one vertex $v_i$. For each symmetric nonzero pair $a_{ij}$ and $a_{ji}$, $E$ 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 $G = (V, E)$, we use the notation $\text{Adj}(v_i)$ 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 $V' \subseteq V$, i.e., $\text{Adj}(V') = \bigcup_{v_i \in V'} \text{Adj}(v_i) - V'$. Two vertex subsets $V' \subseteq V$ and $V'' \subseteq V$ are said to be adjacent if $\text{Adj}(V') \cap V'' \neq \emptyset$ (or equivalently $\text{Adj}(V'') \cap V' \neq \emptyset$) and non-adjacent otherwise.

A vertex subset $S$ is a $K$-way vertex separator if the subgraph induced by the vertices in $V - S$ has at least $K$ connected components. $\Pi_{VS} = \{V_1, V_2, \ldots, V_K; S\}$ is a $K$-way vertex partition of $G$ by vertex separator $S \subseteq V$ if all parts are nonempty (i.e., $V_k \neq \emptyset$ for $k = 1, \ldots, K$), all parts and the separator are pairwise disjoint, the union of the parts and the separator gives $V$, and the vertex parts are pairwise nonadjacent (i.e., $\text{Adj}(V_k) \subseteq S$ for $k = 1, \ldots, K$). $V_k \cap \text{Adj}(S)$ is said to be the boundary vertex set of part $V_k$.

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.,

$$\text{Separator size}(\Pi_{VS}) = |S|. \quad (4)$$

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

$$\max_{1 \leq k \leq K} \{W(V_k)\} \leq (1 + \epsilon)W_{\text{avg}}. \quad (5)$$

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

$$W(V_k) = \sum_{v_i \in V_k} w(v_i) \quad (6)$$

is the weight of part $V_k$ 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 VS $\Pi_{VS} = \{V_1, V_2; S\}$ of the original graph $G = G[V]$ is obtained and then this 2-way $\Pi_{VS}$ 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 $\Pi_{VS}$, since they contribute to the $K$-way separator size only once, thus inducing vertex induced subgraphs $G[V_1]$ and $G[V_2]$. Then 2-way GPVS is recursively applied on both $G[V_1]$ and $G[V_2]$. 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 the 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 $V$ 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 $V$ 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, 4, 5].

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 pre-assigned 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 pre-assigned. We use the notation $F_k$ to denote the subset of vertices that are fixed to part $V_k$, for $k = 1, 2, ..., K$. The remaining vertices (i.e., vertices in $V - \bigcup_{k=1}^{K-1} F_k$) 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 Problem

In the ordered GPVS (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 $S = < S_1, S_2, ..., S_{K-1} >$ of mutually disjoint $K-1$ subseparators in such a way that:

(i) each vertex in subseparator $S_k$ connects vertices only in successive parts $V_k$ and $V_{k+1}$, for $k = 1, 2, ..., K-1$.

(ii) edges between subseparators are restricted to be between only successive supseparators, i.e., $S_k$ and $S_{k+1}$ for $k = 1, 2, ..., K-2$.

Here, we refer $S_k$ as the right subseparator of $V_k$ and the left subseparator of $V_{k+1}$. We introduce the following formal definitions for oVS and the oGPVS problem:

**Ordered Vertex Separator** $\Pi_{oVS}$: $\Pi_{oVS} = \{<V_1, V_2, ..., V_K > ; S \}$ is a $K$-way ordered vertex partition of $G = (V, E)$ by an ordered vertex separator $S = < S_1, S_2, ..., S_{K-1} >$ if each subseparator $S_k$ is nonempty; all parts and subseparators are pairwise disjoint; the union of parts and subseparators gives $V$; parts are pairwise non-adjacent; only successive subseparators can be pairwise adjacent; successive parts $V_k$ and $V_{k+1}$ are connected by the vertices of the subseparator $S_k$ between these two parts.

**oGPVS Problem**: Given a graph $G = (V, E)$, an integer $K$, and a maximum allowable imbalance ratio $\epsilon$, the oGPVS problem is finding a $K$-way ordered vertex separator $\Pi_{oVS}(G) = \{<V_1, V_2, ..., V_K > ; S \}$ of $G$ by a vertex separator $S = < S_1, S_2, ..., S_{K-1} >$ that minimizes the overall separator size $|S| = \sum_{k=1}^{K-1} |S_k|$ while satisfying the balance criterion on the weights of $K$ parts given in (5).
3.1 Formulation of an $A$-to-$A_{BDO}$ Problem as an oGPVS Problem

Let $G(A) = (\mathcal{V}, \mathcal{E})$ be the standard graph representation of a given structurally symmetric sparse matrix $A$ where weight of each vertex $v_i$ is set to be equal to the number of nonzeros in row/column $i$. A $K$-way oVS $\Pi_{oVS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ 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 $S_k$ constitute the rows/columns of the subblock $A_{k,k}$ and $C_{k,k}$ respectively. Thus,

- $|S_k| = n_c^k$ and hence minimizing the separator size $|S| = \sum_{k=1}^{K-1} |S_k|$ 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.

See section 3.2 of [9] for a detailed discussion.

Figure 2 shows a sample $24 \times 24$ matrix $A$ which contains 116 nonzeros and the standard graph representation $G(A)$ which contains 24 vertices and 46 edges. Figure 3 shows a 4-way oVS $\Pi_{oVS}(G) =$
{< V_1, V_2, V_3, V_4 >; < S_1, S_2, S_3 >} of G, where V_1, V_2, V_3 and V_4 respectively contain 4, 5, 4 and 4 vertices, and S_1, S_2 and S_3 respectively contain 2, 3 and 2 vertices. Figure 4 shows a BDO form of the sample matrix A given in Figure 2, which is induced by \( \Pi_{oVS}(G) \) given in Figure 3. As seen in Figure 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 \( D_4 \).

3.2 Diameter Limitation

Diameter of a graph \( G = (V, E) \) is defined as the maximum shortest path distance between \( v_i \in V \) and \( v_j \in V \), for any vertex pair \( v_i \) and \( v_j \). \( G \) has a \( K \)-way oVS if and only if the diameter of \( G \) is at least \( K - 2 \). See Section 4.1 of [9] for the proof.

3.3 oGPVS Algorithm

The oGPVS algorithm consists of two phases; finding pseudo-peripheral vertices and recursive bipartitioning.

A peripheral vertex in a given graph \( G \) is one of the end vertices of a path whose length is equal to the diameter of \( G \). Since there is no known efficient procedure that always finds peripheral vertices, we use the algorithm given in [3] for finding a pseudo-peripheral vertex in a given graph \( G \). If the shortest path distance between a pseudo-peripheral vertex \( v \) and one of the vertices furthest to \( v \) is less than \( K - 2 \), oGPVS algorithm cannot partition \( G \) into \( K \)-way oVS (see Section 3.2), so the algorithm terminates.

In the recursive bipartitioning (RB) phase, \( G \) is bipartitioned recursively, producing a subseparator at each RB step, until the final oVS has the desired number of parts. Figure 5 visualizes the RB-tree for producing an 8-way oVS of a graph. At each RB step, it needs to be guaranteed that the current subgraph \( G' \) of \( G \) has a diameter of at least \( K' - 2 \) if a \( K' \)-way oVS is going to be produced from \( G' \) through further bipartitions. At each RB step, some vertices of the current subgraph \( G' \) are fixed to left (right) vertex part of the forthcoming bipartition so that the left (right) subgraph induced by the this left (right) vertex part is guaranteed to have some desired diameter, say \( K_L - 2 \) (\( K_R - 2 \)). Hence a \( K_L \)-way (\( K_R \)-way) oVS can be produced from this left (right) subgraphs. Combining the oVS’s of
the left and the right subgraphs, a \((K_L + K_R)\)-way oVS of the \(G'\) is produced. See Section 4.2 of [9] for the details of the algorithm.

We utilized the hypergraph partitioning (HP) based GPVS formulation proposed in [10] due to the absence of a GPVS tool that supports fixed vertices. For hypergraph partitioning, we used PaToH [2] which is an HP tool supporting fixed vertices. See Section 5.1 of [9] for the details of HP-based GPVS formulation.

4 Library Interface

oGPVS library interface consists of a single library file, libogpvs.a. Since we used PaToH in our HP based GPVS formulation, PaToH library, libpatoh.a, is also needed at compilation. Graph representation used by the oGPVS library interface is described in Section 4.1, then detailed description of the functions are presented in Section 4.2.

Before starting to discuss the details, lets look at a simple C program that partitions an input graph into 16-way ordered Vertex Separator using oGPVS functions. The program is displayed in Figure 6. First two statements set the partitioning parameters, namely the number of parts and the desired imbalance of the part weights in the resulting oVS, to 16 and 10%. Then, oGPVS_read_graph function is called to read the input graph which is given by the first command line argument. The format for the input graph is Matrix Market file format. Besides reading the input graph, this function also finds pseudo-peripheral vertices, namely leftroot and rightroot, and pseudo-diameter of the graph. Since finding pseudo-peripheral vertices takes a considerable time, and the same pseudo-peripheral vertices of a graph can be used for different oGPVS runs with/and different \(K\)-values, oGPVS_read_graph function runs the pseudo-peripheral node finder algorithm once for a graph and save the results in a file for that graph. This file holding the pseudo-peripheral vertices and pseudo-diameter lengths for the graphs seen so far is given by the second command line argument to the program. After reading the graph and finding its pseudo-peripheral vertices and pseudo-diameter length, oGVP partitioning function, namely oGPVS_partition is called to accomplish a 16-way oGPVS of the given graph with the provided pseudo-peripheral vertices. Call to oGPVS_partition will partition the graph and the resulting partition vector, its corresponding BDO permutation vector, final imbalance ratio on
part weights, and separator size, i.e., the number of vertices in the separator, will be returned in the parameters.

```c
#include <stdio.h>
#include <stdlib.h>
#include "ogpvs.h"

int main(int argc, char *argv[]) {

    int n, *xadj, *adjncy, leftroot, rightroot, pdiameter; // graph variables
    int kway, nseparator, nemptyparts; double dimbal, fimbal; // ogpvs variables

    dimbal = 0.10; // desired imbalance is set to 0.10
    kway = 16; // desired number of parts is set to 16

    if(!oGPVS_read_graph(argv[1], argv[2], &n, &xadj, &adjncy, &leftroot, &rightroot, &pdiameter))
    {
        int *permute = (int *)calloc(sizeof(int), n);
        int *partition = (int *)calloc(sizeof(int), n);

        if(!oGPVS_partition(n, &xadj, &adjncy, leftroot, rightroot, pdiameter, kway, dimbal,
                            partition, permute, &nseparator, &fimbal, &nemptyparts))
        {
            printf("%d-way oGPVS results:
Separator Size: %d, Imbalance Ratio: %.2f
", kway, nseparator, fimbal);
        }

        free(permute);
        free(partition);
    }

    return 0;
}
```

Figure 6: A simple C program that partitions an input graph into 16-way oVS

4.1 Graph Representation

A graph and its representation can be seen in Figure 7. `xadj` and `adjncy` arrays stores, the begin-
ing index of vertices adjacent to each vertex, and IDs of the vertices, respectively. Hence, `xadj`
is an array of size number of vertices plus one, and `adjncy` is an array of size twice the number of
edges in the undirected graph. Vertices adjacent to vertex \( v_i \) are stored in `adjncy[xadj[i]]` through
`adjncy[xadj[i+1]-1]`.

In the matrix theoretical view, this representation scheme corresponds to the Compressed Storage
by Rows/Columns (CSR/CSC) scheme of the standard adjacency matrix representation \( A \) of the given
graph \( G \). Note that matrix \( A \) is symmetric with equivalent CSR and CSC structures when \( G \) is
undirected.

4.2 Functions

Current oGPVS interface contains five functions; one for reading input graph, one for partitioning, two
for writing the output vectors (partition of vertices and permutation of rows/columns of corresponding
matrix), and one for writing the permuted BDO matrix.
int oGPVS_read_graph(char *inputfile, char *perfile, int *n, int **xadj, int **adjncy, int *leftroot, int *rightroot, int *pdiameter)

Description
This function first reads a matrix/graph from the given inputfile. The inputfile has to be in Matrix Market file format. If the input matrix is not square, the function terminates since its standard graph representation can not be produced, and returns 1. If the input matrix is square, its standard graph representation is produced regardless of its symmetricity, considering all edges undirected. xadj and adjncy arrays are filled with the information of the graph structure as expressed in Section 4.1. After having the graph in xadj and adjncy arrays, the function reads the pseudo-peripheral vertices leftroot and rightroot and pseudo-diameter length pdiameter associated with the graph from the file perfile, if the related information exists. Otherwise, it computes pseudo-peripheral vertices leftroot and rightroot, and pseudo-diameter length pdiameter of the graph, and writes this information to perfile together with the graph name for further use. In either cases, the function returns 0 and each of the output parameters (n, leftroot, rightroot, pdiameter, xadj and adjncy) hold the related information in return. If the graph is discovered to be disconnected during peripheral-node finding process, the function terminates and returns 1.

Parameters:

- **inputfile**: input name of the Matrix Market file that contains the matrix/graph. name of the file that contains peripheral vertex information.
- **perfile**: input name of the file that contains peripheral vertex information. If the file does not exists, it will be created with the given name.
- **n**: output number of the vertices of the graph.
- **xadj**: output array of size n+1 that stores the beginning index of adjacent vertices.
- **adjncy**: output array that stores the adjacency list of each vertex. Vertices adjacent to vertex \( v_i \) are stored in adjncy[xadj[i]] through adjncy[xadj[i+1]-1].
- **leftroot**: output one of the pseudo-peripheral vertices of the graph.
- **rightroot**: output one of the vertices that are furthest to leftroot.
- **pdiameter**: output the shortest path distance between leftroot and rightroot.
int oGPVS_partition(int n, int **xadj, int **adjncy, int leftroot, int rightroot,
                   int pdiameter, int kway, double dimbalance, int *partition,
                   int *permute, int *nseparator, double *imbalance, int *nemptyparts)

Description
This function partitions the graph into kway ordered Vertex Separator (oVS) so that the number of
vertices in separator (nseparator) is minimized and the balance on the part weights is maintained.
If the pseudo-diameter length (pdiameter) is less than kway-2, the function terminates and returns 1
since the graph can not be partitioned into kway-oVS (see Section 3.2).

Parameters:
n input number of vertices in the graph.
xadj input array of size n+1 that stores the beginning index of adjacent vertices.
adjncy input array that stores the adjacency list of each vertex. Vertices adjacent to vertex \( v_i \) are stored in \( \text{adjncy}[\text{xadj}[i]] \) through \( \text{adjncy}[\text{xadj}[i+1]-1] \).
leftroot input one of the pseudo-peripheral vertices of the graph.
rightroot input one of the vertices that are furthest to leftroot.
pdiameter input the shortest path distance between leftroot and rightroot.
kway input number of parts in oVS.
dimbalance input desired imbalance ratio of the part weights.
partition output array of size n that holds the partition of vertices into parts.
permute output array of size n that holds the permutation of rows/columns that produces the BDO form of the input matrix.
nseparator output number of vertices in the separator, i.e., number of rows/columns in the coupling diagonal blocks \( C_{k,k} \)’s in (2, 3).
imbalance output final imbalance ratio of the part weights, i.e. number of nonzeros in diagonal blocks \( D_k \)’s in (2, 3).
nemptyparts output number of empty parts, i.e., empty \( A_{k,k} \)’s in (2, 3).
void oGPVS_write_partition(int n, int *partition, char *outputfile)

Description
This function writes the partition vector into outputfile.

Parameters:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>input</td>
<td>number of vertices in the graph.</td>
</tr>
<tr>
<td>partition</td>
<td>input</td>
<td>array of size n that holds the partition of vertices into parts.</td>
</tr>
<tr>
<td>outputfile</td>
<td>input</td>
<td>name of the file that will contain the partition vector.</td>
</tr>
</tbody>
</table>

void oGPVS_write_permutation(int n, int *permute, char *outputfile)

Description
This function writes the permutation vector into outputfile.

Parameters:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>input</td>
<td>number of vertices in the graph.</td>
</tr>
<tr>
<td>permute</td>
<td>input</td>
<td>array of size n that holds the permutation of rows/columns that produces the BDO form of the input matrix.</td>
</tr>
<tr>
<td>outputfile</td>
<td>input</td>
<td>name of the file that will contain the permutation vector.</td>
</tr>
</tbody>
</table>

void oGPVS_permute_and_write_matrix(char *inputfile, int *permute, char *outputfile)

Description
This function reads the input matrix from inputfile, permutes the matrix with permute vector, and writes the permuted matrix into outputfile in Matrix Market file format.

Parameters:
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputfile</td>
<td>input</td>
<td>name of the Matrix Market file that contains the matrix.</td>
</tr>
<tr>
<td>permute</td>
<td>input</td>
<td>array of size n that holds the permutation of rows/columns that produces the BDO form of the input matrix.</td>
</tr>
<tr>
<td>outputfile</td>
<td>input</td>
<td>name of file that will contain the permuted BDO matrix in Matrix File format.</td>
</tr>
</tbody>
</table>
5 The oGPVS/BDO Package

In our oGPVS/BDO distribution, we provide our library `libogpvs.a` together with PaToH library `libpatoh.a`, a stand alone program, our source codes and our profile `peripherals.txt` where pseudo-peripheral vertex and pseudo-diameter length information of 233 matrices (graphs) of University of Florida (UFL) Sparse Matrix Collection [11] can be found.

oGPVS functions are used via `libogpvs.a`. These oGPVS functions use PaToH functions for partitioning, hence `libpatoh.a`. You can compile your test code, e.g. simple test code in Figure 6, with the following command:

```bash
> gcc -o simpletest simpletest.c -L -logpvs -lpatoh -lm
```

You can run simpletest on `1138_bus` matrix with peripheral file `peripherals.txt` as follows:

```bash
> simpletest 1138_bus mtx peripherals.txt
```

Note that `kway`, i.e. number of parts, is set to 16 in this code. So, a 16-way oVS partitioning of `1138_bus` is produced.

We provide a stand-alone oGPVS/BDO program, called `ogpvs`, that gets all of its parameters from command line arguments. You can run `ogpvs` from command line as follows:

```bash
> ogpvs <inputfile> [[parameter1] [parameter2] ....]
```

Parameters are desired number of parts and imbalance on part weights, file name of peripheral file, and file name that the permuted BDO matrix will be written to. If the given peripheral file does not exist, the file will be created and pseudo-peripheral vertices and pseudo-diameter length of `<inputfile>` will be computed and written to it. If the file does exist but does not include the corresponding information for `<inputfile>`, it will be computed and written to the file. If the file does exist and has the corresponding information for `<inputfile>`, it will be just read from the file.

For a complete example, lets say we would like to partition the graph `1138_bus` into 7 parts with 5% imbalance on part weights using the peripheral file `peripherals.txt` and the corresponding permuted matrix into BDO form will be written to the file `1138_bus_7-way_BDO.mtx`. Below is the command you need to execute and a sample output:

```bash
> ogpvs 1138_bus mtx -p peripherals.txt -k 7 -i 0.05 -o 1138_bus_7-way_BDO.mtx
```

```
*******************************************************************************
oGPVS/BDO: ordered Graph Partitioning by Vertex Separator for Permuting Matrices into Block Diagonal Form with Overlap*******************************************************************************
Matrix: 1138_bus, #Rows: 1138, #Columns: 1138, #Nonzeros: 4054
7-way oGPVS results:
Number of vertices(rows/columns) in separator(overlaps): 125
Imbalance ratio on the part weights (number of nonzeros in diagonal blocks): 24%
Permutated 7-way BDO matrix written into 1138_bus_7-way_BDO.mtx
*******************************************************************************
```
Parameters of oGPVS program except from the <inputfile> have their default values: number of parts (-k) is 2, imbalance (-i) is 0.10, peripheral file name (-p) is peripherals.txt, output file name (-o) is permutedmatrix.mtx.

References


