PaToH Matlab Interface*

Bora Uçar
Centre National de la Recherche Scientifique,
Laboratoire de l'Informatique du Parallélisme,
(UMR CNRS-ENS Lyon-INRIA-UCBL),
Université de Lyon,
46, allée d'Italie, ENS Lyon, F-69364, Lyon, France
bora.ucar@ens-lyon.fr

Ümit V. Çatalyürek
Department of Biomedical Informatics
The Ohio State University
Columbus, OH 43210
umit@gatech.edu

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

June, 2009

For additional information and documents on PaToH
http://cc.gatech.edu/~umit/software.html

*This work is partially supported by the National Science Foundation Grants CNS-0643969, CCF-0342615, CNS-0403342, the Department of Energy's Office of Science through the CSCAPES SciDAC Institute DE-FC02-06ER25775, and "Agence Nationale de la Recherche" through SOLSTICE project ANR-06-CIS6-010.

Contents

0 The fast-track

If you put `PaToH.h` and `libpatoh.a` (these files are in PaToH C-library distribution) in the directory containing the files of the Matlab distribution, issuing the following commands in a Matlab prompt would get you going.

```
mex PaToH.c -lpatoh -L. -I. -lm
help PaToHMatrixPart
```

On a 64-bit machine add `-largeArrayDims` to `mex` compilation.

1 Introduction

The aim of the PaToH Matrix Partitioning Interface is to provide sparse matrix partitioning routines in Matlab. The partitioning routines are based on hypergraph models [1, 3, 4, 5] and use PaToH hypergraph partitioning tool [2] within a `mex` function. Apart from the `mex` function routine that builds a hypergraph and calls PaToH, everything else is based on matrices and vectors. Therefore, we refer the reader to the papers cited above for the details of the hypergraph models and the algorithms to partition the hypergraphs. We refer the reader to [6] for a sample use of this interface for developing partitioning algorithms. This document covers only matrix partitioning routines and some other utility functions concerning matrix partitioning.

The most common use of hypergraph partitioning-based sparse matrix partitions is to efficiently parallelize sparse matrix-vector multiply (SpMxV) operation $\mathbf{y} \leftarrow \mathbf{Ax}$. Here, $\mathbf{A}$ is a sparse matrix, $\mathbf{x}$ is the input-vector and $\mathbf{y}$ is the output-vector of the multiply operation (it is understood that the vectors are of appropriate sizes). As is clear, a parallelization of this computation requires partitioning of the input- and output-vectors as well. *Symmetric vector partitioning* (for short symmetric partitioning) refers to the case in which the input- and output-vectors have the same partitions. In a similar vein, *unsymmetric vector partitioning* refers to the case in which the input- and output-vectors have different partitions. The parallelization is achieved by partitioning the matrix, and the input and output-vectors among K processors. That is, a processor holds a set of nonzeros of the matrix, a portion of the input-vector $\mathbf{x}$, and is set to be responsible for holding a portion of the output-vector $\mathbf{y}$. In this setting the parallel matrix-vector multiply operation $\mathbf{y} \leftarrow \mathbf{Ax}$ proceeds as follows (we are essentially rewriting the exposure in [7]; for a more detailed account see [5]):

1. Each processor sends x_j to those processors that have a nonzero in column j of $\mathbf{A}$.
2. Each processor P_k computes the scalar products $a_{ij}x_j$ with each a_{ij} it owns and computes a partial result $y_i^{(k)}$ by adding the results of the scalar products for each i.
3. Each processor sends its partial results $y_i^{(k)}$ to the processor which is responsible for computing the resulting y_i.
4. Each processor adds the contributions received for the vector entries y_i it is responsible for, i.e., it forms $y_i = \sum_k y_i^{(k)}$.

The communication operations that take place before the scalar products are referred to here as *expand* operations, and the communication operations that take place after the scalar products are referred to here as *fold operations*. The *expand phase* and the *fold phase*, respectively, refer to the assembly of the expand and fold operations. Notice that in a one dimensional (1D) partitioning of the matrix $\mathbf{A}$, there can be either expand operations (rowwise partitioning) or fold operations (columnwise partitioning). In the expand phase, all x-vector entries to be sent by a given processor to the same destination processor are packed and sent in a single message. A similar packing is performed in the fold phase. A few definitions regarding the communication operations are in order.

totVolE(k)**:** the total volume of the expand messages sent by the processor P_k.

totNumE(k)**:** the total number of the expand messages sent by the processor P_k.

totVolF(k)**:** the total volume of the fold messages sent by the processor P_k.

totNumF(k)**:** the total number of the fold messages sent by the processor P_k.

totVolE: the total sum of the sizes of messages sent in the expand phase, i.e., $\text{totVolE} = \sum_k \text{totVolE}(k)$.

totVolF: the total sum of the sizes of messages sent in the fold phase, i.e., $\text{totVolF} = \sum_k \text{totVolF}(k)$.

totNumE: the total number of messages sent in the expand phase, i.e., $\text{totNumE} = \sum_k \text{totNumE}(k)$.

totNumF: the total number of messages sent in the fold phase, i.e., $\text{totNumF} = \sum_k \text{totNumF}(k)$.

maxVolE: the maximum volume of messages sent by a processor in the expand phase, i.e., $\text{maxVolE} = \max_k \text{totVolE}(k)$.

maxVolF: the maximum volume of messages sent by a processor in the fold phase, i.e., $\text{maxVolF} = \max_k \text{totVolF}(k)$.

The computational load of a processor is defined as the number of scalar products it performs, i.e., the number of nonzero entries a_{ij} it holds. Therefore, the *load imbalance* can be defined as

$$\frac{\max_k |\mathbf{A}^{(k)}| - |\mathbf{A}|/K}{|\mathbf{A}|/K} \quad , \tag{1}$$

where $|\mathbf{A}^{(k)}|$ denotes the number of nonzeros owned by processor P_k, and $|\mathbf{A}|$ denotes the number of nonzeros in matrix $\mathbf{A}$. Notice that the equation (1) measures how far the maximum load is from the average load. It is a number between 0 (all processors have the same number of nonzeros) and $K-1$ (a single processor holds all of the nonzeros). We will refer to this ratio as *higher imbalance ratio*, as opposed to *lower imbalance ratio*:

$$\frac{\min_k |\mathbf{A}^{(k)}| - |\mathbf{A}|/K}{|\mathbf{A}|/K} \quad . \tag{2}$$

Notice that the lower imbalance ratio varies between -1 (there is a processor which does not have any nonzeros) and 0 (all processors have the same number of nonzeros). These two metrics can also play a role in comparing two partitions.

2 The contents of the distribution

The distribution should contain the files specified below

`PaToHMatrixPart.m`: The main function that is used to partition matrices.

`PaToHSpy.m`: A utility function to visualize partitions on matrices.

`PaToHComputeImbal.m`: A utility function to compute the load imbalance among processors.

`PaToHComputeVolume.m`: A utility function to compute the total volume of messages and the maximum volume of messages sent by a processor.

`PaToHComputeNumber.m`: A utility function to compute the total number of messages and the maximum number of messages sent by a processor.

`PaToH.c`: A `mex` file, which calls PaToH. It can be used in developing new matrix partitioning algorithms.

`PaToH.m`: A dummy function which contains the `help` text for PaToH.c.

`PaToHNaiveVectorPart.m`: A utility function to partition the input- and output-vectors of the sparse matrix-vector multiply operation with a given partitioned matrix.

The main function provided in C-file `PaToH.c` should be compiled with `mex` (easier in a Matlab prompt) using the following command

```
mex PaToH.c -lpatoh -L<PATH-TO-LIBPATOH.A> -I<PATH-TO-PATOH.H> -lm
```

by replacing "`<PATH-TO-LIBPATOH.A>`" to the directory that holds `libpatoh.a` and "`<PATH-TO-PATOH.H>`" to the directory that holds `patoh.h`. These two files are parts of the PaToH distribution which can be downloaded from http://cc.gatech.edu/~umit/software.html. If you have a 64-bit installation of Matlab, you should use 64-bit distribution of PaToH and add `-largeArrayDims` to the `mex` compilation command.

3 The interface

The main function in the interface is in the file `PaToHMatrixPart.m`.

```
function [nnzpv,outpv,inpv,ptime] = PaToHMatrixPart(mat, K, dim,
                                                    partitioner, imbal_in)
```

Description:
Partitions the matrix `mat` into `K` parts.

Parameters:

`mat`	required	a sparse matrix.
`K`	required	the number of parts. Should be at least 1.
`dim`	required	A string describing the partitioning method. Should be one of • `'RWS'`: Rowwise symmetric partitioning • `'RWU'`: Rowwise unsymmetric partitioning • `'CWS'`: Columnwise symmetric partitioning • `'CWU'`: Columnwise unsymmetric partitioning • `'FGS'`: Fine-grain symmetric partitioning • `'FGU'`: Fine-grain unsymmetric partitioning • `'JLS'`: Jagged-like symmetric partitioning • `'JLU'`: Jagged-like unsymmetric partitioning • `'CHS'`: Checkerboard symmetric partitioning • `'CHU'`: Checkerboard unsymmetric partitioning
`partitioner`	optional	should be one of • `0`: hypergraph-based partitioning with PaToH (default value). • `1`: random • `2`: Block-row balanced • `3`: Block-nonzero balanced • `4`: Round-robin
`imbal_in`	optional	permitted higher imbalance ratio (1). Default is 3%.
`nnzpv`	return	the part vector on nonzero basis. This is a sparse matrix, with the same sparsity of the input matrix `mat`.
`outpv`	return	the part vector of the output vector of SpMxV with `mat`
`inpv`	return	the part vector of the input vector of SpMxV with `mat`.
`ptime`	return	the partitioning time in seconds, measure the time spent in PaToH.

We note that not all `partitioner`s are implemented for all possible `dim` for the time being. The default `partitioner` is PaToH for which all partitioning alternatives are implemented. The

outputs `outpv` and `inpv` are dense vectors, describing the partition on the output- and input-vectors of the SpMxV operation, and `nnzpv` is a sparse matrix of the same sparsity pattern as the matrix `mat` and describes the partition of the nonzeros of the input matrix. The part numbers are integers from 1 to `K`.

4 Utility Functions

One of the most useful utility functions is `PaToHSpy` that visualizes the partitioned matrix.

```
function PaToHSpy(nnzpv, outpv, inpv, K, symb, ttl, printperm)
function PaToHSpy(nnzpv, symb)
```

Description:
Plots the given partitioned sparse matrix. Each part is displayed with a different color (and symbol if `symb = '?'`). If input and output partitioning vectors are given, matrix is permuted to reflect vector partitioning, under the assumption that each input or output vector element is assigned to a processor which owns at least one nonzero in the respective column or row. After permutation, the i-th block of input and output vector elements are owned by processor i.

Parameters:

`nnzpv`	required	the part vector on nonzero basis. This is a sparse matrix, each nonzero entry has an integer value between 1 and `K`.
`outpv`	optional	the part vector of the output vector of SpMxV with a sparse matrix whose partition is provided in `nnzpv`.
`inpv`	optional	the part vector of the input vector of SpMxV with a sparse matrix whose partition is provided in `nnzpv`.
`K`	optional	the number of parts, computed as `max(full(max(nnzpv)))` in case it is not given.
`symb`	optional	marker symbol. If it is '?', a different marker is used for each part.
`ttl`	optional	title for the plot.
`printperm`	optional	if 1, permuted row/column indices are displayed. We only recommend the use of this option for small matrices.

`function [imbalanceLower, imbalanceUpper] = PaToHComputeImbal(nnzpv, K)`

Description:
Calculates the imbalance ratios shown in (1) and (2).

Parameters:

`nnzpv`	required	the part vector on nonzero basis. This is a sparse matrix, each nonzero entry has an integer value between 1 and `K`
`K`	optional	the number of parts, computed as `max(full(max(nnzpv)))` in case it is not given.
`imbalanceLower`	return	Imbalance ratio shown in (2). In case second output argument is missing, imbalance ratio of (1).
`imbalanceUpper`	return	Imbalance ratio shown in (1).

`[totvolE,maxvolE, totvolF, maxvolF] = PaToHComputeVolume(nnzpv, outpv, inpv, K)`

Description:
Calculates the total and the maximum communication volume both for fold and expand operations.

Parameters:

`nnzpv`	required	the part vector on nonzero basis. This is a sparse matrix, each nonzero entry has an integer value between 1 and `K`.
`outpv`	required	the part vector of the output vector of SpMxV with a sparse matrix whose partition is provided in `nnzpv`.
`inpv`	required	the part vector of the input vector of SpMxV with a sparse matrix whose partition is provided in in `nnzpv`.
`K`	optional	the number of parts, computed as `max(full(max(nnzpv)))` in case it is not given.
totvolE	return	the total communication volume of expand operations. If the other output arguments are not present, then represents the total communication volume of expand and fold operations.
maxvolE	return	the maximum volume of expand messages sent by a single processor.
totvolF	return	the total communication volume of fold operations.
maxvolF	return	the maximum volume of fold messages sent by a single processor.

`[totnumE,maxnumE, totnumF, maxnumF] = PaToHComputeNumber(nnzpv, outpv, inpv, K)`

Description:
Calculates the total and the maximum number of messages both for fold and expand operations.

Parameters:

`outpv`	required	the part vector of the output vector of SpMxV with a sparse matrix whose partition is provided in `nnzpv`
`inpv`	required	the part vector of the input vector of SpMxV with a sparse matrix whose partition is provided in in `nnzpv`
`nnzpv`	required	part vector on nonzero basis. This is a sparse matrix, each nonzero entry has an integer value between 1 and `K`
`K`	optional	the number of parts, computed as `max(full(max(nnzpv)))` in case it is not given.
totnumE	return	the total number of expand operations. If the other output arguments are not present, then represents the total number of expand and fold operations.
maxnumE	return	the maximum number of expand messages sent by a single processor.
totnumF	return	the total number of fold operations
maxnumF	return	the maximum number of fold messages sent by a single processor.

`function [outpv, inpv] = PaToHNaiveVectorPart(nnzpv, isSym, K)`

Description:
Computes vector partitions for the SpMxV operation with a matrix whose nonzeros are partitioned according to `nnzpv`

Parameters:

`nnzpv`	required	the part vector on nonzero basis. This is a sparse matrix, each nonzero entry has an integer value between 1 and `K`
`isSym`	required	if `isSym=0` a nonsymmetric vector partitioning is returned; else a symmetric one is returned.
`K`	optional	the number of parts, computed as `max(full(max(nnzpv)))` in case it is not given.
`outpv`	return	the part vector of the output vector of SpMxV with a sparse matrix whose partition is provided in `nnzpv`
`inpv`	return	the part vector of the input vector of SpMxV with a sparse matrix whose partition is provided in in `nnzpv`

The function provided in `PaToH.m` is a dummy function and called only if the `mex` file for `PaToH.c` was not built. It also contains the documentation for PaToH interface provided in `PaToH.c`. Hence,

the following description applies to the main function `PaToH.c` as well (this functions transfers Matlab structures into PaToH's hypergraph data structures, calls PaToH, and transfers partitioning information into Matlab structures).

```
function [partv, ptime] = PaToH(hyp, K, nconst, cw, nw, imbal, cuttype)
```

Description:
Partitions the hypergraph `hyp` into `K` parts. The vertices of `hyp` have `nconst` many weights which are stored in `cw`. The nets have costs which are stored in `nw`. The partition should meet a balance criterion of `imbal`. The objective function to minimize is described by the string `cuttype`.

Parameters:

`hyp`	required	a hypergraph given as a sparse matrix. The rows of the matrix represent the vertices of the hypergraph, and the columns of the matrix represent the nets of the hypergraph. The matrix entry at position (i, j) is nonzero if and only if vertex i is in net j.
`K`	required	the number of parts
`nconst`	optional	the number of constraints. Default is `1`
`cw`	optional	the cell weights. `cw((i-1)*nconst+1)` to `cw((i-1)*nconst+nconst-1)` contains the weights of the cell `i`. Here `i`≥ 1. Default weights are 1.
`nw`	optional	the net costs. The net `j` has cost `nw(j)`. Here `j`≥ 1. Default costs are 1.
`imbal`	optional	permitted higher imbalance ratio (1). Default is 3%.
`cuttype`	optional	With the option 'CON' , the hypergraph partitioning objective function is the sum of the connectivity-1 values of the nets. With the option 'CUT', the same function is the sum of the costs of the cut nets. Default is 'CON'.
`partv`	return	the partition vector for the vertices of the hypergraph `hyp`. The values are between `0` and `K-1` (not to be confused with the other part vectors used in the matrix partitioning interface).
`ptime`	return	the partitioning time in seconds, measures the time spent in PaToH.

---

5 Examples

Assuming the `mex` file for PaToH was built, we start by loading a matrix (`west0479` is available in Matlab) and having a look at its plot

```
load west0479
spy(west0479)
```

Then, we partition the matrix among 4 processors using different partitioning methods:

```
[nnzpvcwu, outpvcwu, inpvcwu] = PaToHMatrixPart(west0479, 4, 'CWU');
[nnzpvfgu, outpvfgu, inpvfgu] = PaToHMatrixPart(west0479, 4, 'FGU');
[nnzpvjlu, outpvjlu, inpvjlu] = PaToHMatrixPart(west0479, 4, 'JLU');
[nnzpvchu, outpvchu, inpvchu] = PaToHMatrixPart(west0479, 4, 'CHU');
```

Then, we visualize the different partitions

```
PaToHSpy(nnzpvcwu, outpvcwu, inpvcwu, 4, '.', 'Columnwise partitioning');
PaToHSpy(nnzpvfgu, outpvfgu, inpvfgu, 4, '.', 'Fine-grain partitioning');
PaToHSpy(nnzpvjlu, outpvjlu, inpvjlu, 4, '.', 'Jagged-like partitioning');
PaToHSpy(nnzpvchu, outpvchu, inpvchu, 4, '.', 'Checkerboard partitioning');
```

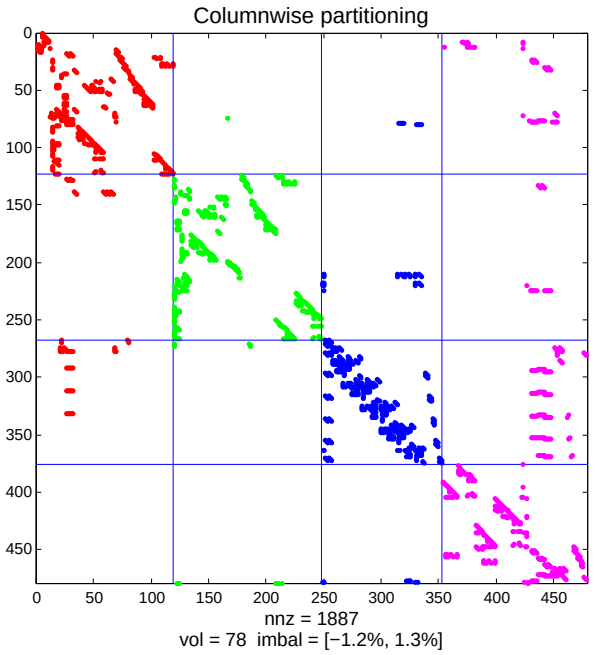
The plots are seen in Fig. 1. We see that the total volume of communication in CWU is 78, in FGU it is 72, in JLU it is 81, and in CHU it is 89. We also see the lower and upper imbalance ratios in the figures. Please note that since PaToH uses random seeds, it is likely that the resulting partitions will differ at each run on the same input set.

We are wondering what happens when we request a symmetric vector partitioning

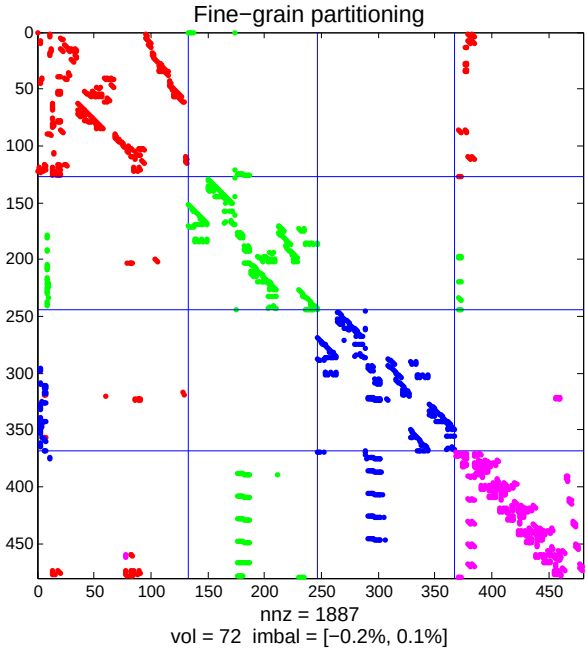
```
[nnzpvcws, outpvcws, inpvcws] = PaToHMatrixPart(west0479, 4, 'CWS');
[nnzpvfgs, outpvfgs, inpvfgs] = PaToHMatrixPart(west0479, 4, 'FGS');
[nnzpvjls, outpvjls, inpvjls] = PaToHMatrixPart(west0479, 4, 'JLS');
[nnzpvchs, outpvchs, inpvchs] = PaToHMatrixPart(west0479, 4, 'CHS');
```

and compare the total volume of messages with respect to the unsymmetric vector partitioning case using the commands below.

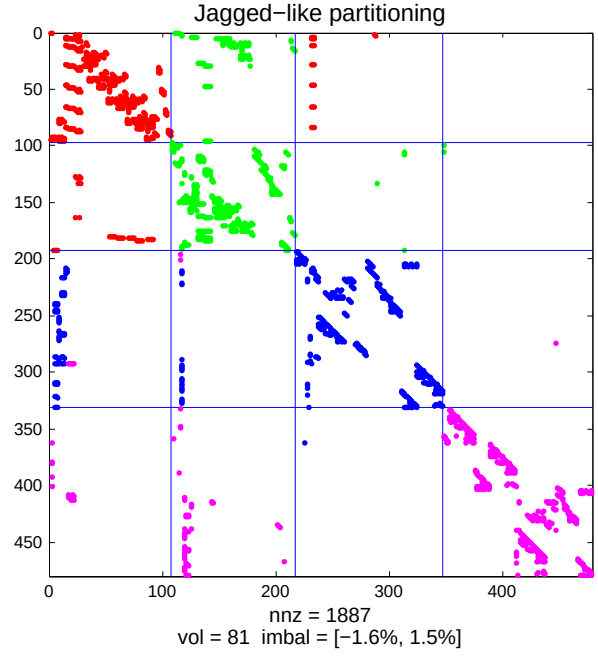
```
totvolcws = PaToHComputeVolume(nnzpvcws, outpvcws, inpvcws, 4)
totvolcws =

   261

totvolfgs = PaToHComputeVolume(nnzpvfgs, outpvfgs, inpvfgs, 4)
totvolfgs =

   238
```



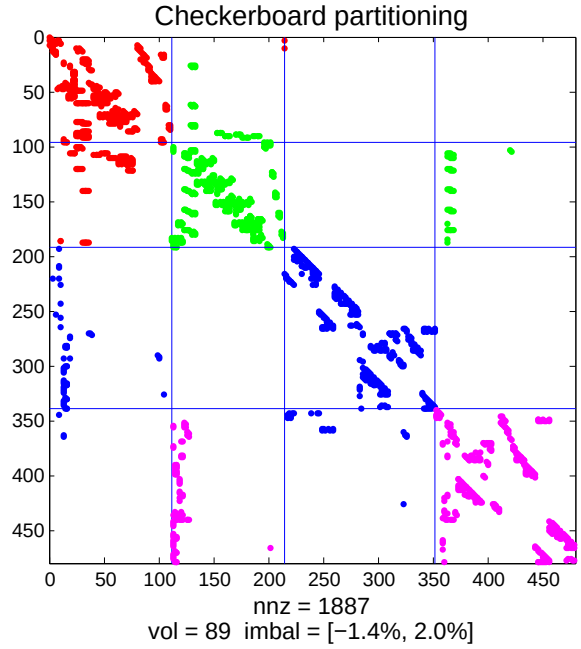
(a) Columnwise, unsymmetric partitioning



(b) Fine-grain, unsymmetric partitioning



(c) Jagged-like, unsymmetric partitioning



(d) Checkerboard, unsymmetric partitioning

Figure 1: Visualizing different partitioning methods on the matrix `west0479`.

```
totvoljls = PaToHComputeVolume(nnzpvjls, outpvjls, inpvjls, 4)
totvoljls =

   240

totvolchs = PaToHComputeVolume(nnzpvchs, outpvchs, inpvchs, 4)
totvolchs =

   259
```

Not much of a surprise, in this case we have a lot more total communication volume.

6 License

PaToH Matlab Interface is released under the GNU Lesser General Public License. A copy of the license is included in this distribution in COPYING.LESSER file.

References

[1] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7):673–693, 1999.

[2] Ü. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.0*. Bilkent University, Department of Computer Engineering, Ankara, 06533 Turkey. PaToH is available at http://cc.gatech.edu/~umit/software.html, 1999.

[3] Ü. V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2D decomposition of sparse matrices. In *Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, CA, April 2001.

[4] Ü. V. Çatalyürek and C. Aykanat. A hypergraph-partitioning approach for coarse-grain decomposition. In *ACM/IEEE SC2001*, Denver, CO, November 2001.

[5] Ü. V. Çatalyürek, C. Aykanat, and B. Uçar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM Journal on Scientific Computing*, 32(2):656–683, 2010.

[6] B. Uçar, Ü. V. Çatalyürek, and C. Aykanat. A matrix partitioning interface to PaToH in Matlab. *Parallel Computing*, 36(5-6):254–272, 2010.

[7] B. Vastenhouw and R. H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.