HAL
open science

On the scalability of hypergraph models for sparse matrix partitioning

Bora Uçar, Umit V. Catalyurek

To cite this version:

Bora Uçar, Umit V. Catalyurek. On the scalability of hypergraph models for sparse matrix partitioning. 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2010), IEEE, Feb 2010, Pisa, Italy. pp.593-600, 10.1109/PDP.2010.92 . ensl-00529088

HAL Id: ensl-00529088

https://ens-lyon.hal.science/ensl-00529088v1

Submitted on 24 Oct 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the scalability of hypergraph models for sparse matrix partitioning

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 Cedex 7, France
Email: bora.ucar@ens-lyon.fr

Ümit V. Çatalyürek
Department of Biomedical Informatics and
Department of Electrical & Computer Engineering
The Ohio State University, Columbus, Ohio, USA
Email: catalyurek.1@osu.edu

***Abstract*—We investigate the scalability of the hypergraph-based sparse matrix partitioning methods with respect to the increasing sizes of matrices and number of nonzeros. We propose a method to rowwise partition the matrices that correspond to the discretization of two-dimensional domains with the five-point stencil. The proposed method obtains perfect load balance and achieves very good total communication volume. We investigate the behaviour of the hypergraph-based rowwise partitioning method with respect to the proposed method, in an attempt to understand how scalable the former method is. In another set of experiments, we work on general sparse matrices under different scenarios to understand the scalability of various hypergraph-based one- and two-dimensional matrix partitioning methods.**

I. Introduction

There are a number of hypergraph-based methods for sparse matrix partitioning methods. The row-net and column-net based models [1], [2] are used to obtain one-dimensional (1D) matrix partitions, along the columns or the rows. The fine-grain [3] method, the checkerboard model [4], the jagged-like method [5], and Mondriaan methods [6] are used to obtain two-dimensional (2D) matrix partitions. The principal objective of all these methods is to efficiently parallelize sparse matrix-vector multiply (SpMxV) operations by partitioning the matrix in such a way that the total volume of communication operations is reduced while achieving computational load balance among processors. Given a matrix, unless one of the specific partitioning method is required, one would like to use the best partitioning resulting from any of those. The notion of best is not very well defined. Considering the principal objective stated above, one can choose the partitioning method that gives the minimum total volume of communication as the best one. How can we know which method would that be without partitioning the given matrix with all of the methods? In [5], Çatalyürek et al. proposed a recipe that suggests a method among the alternatives by using simple statistical measures of the nonzero pattern of a given matrix. We try to go one step beyond and get a better insight into how communication volume scales with scaling of the input, either the matrix size, or the number of parts, or both. To achieve this goal, we examine the models under differing scenarios for which one can make an educated guess as to how the best algorithm would behave.

In order to understand the scalability of the hypergraph partitioning methods we run them on matrices arising from discretization of two-dimensional domains with five-point stencil, again with the objective of reducing the total communication volume in SpMxV operations. Initially, we have thought that the Cartesian partitioning of the mesh, which partitions the nodes of the mesh using vertical and horizontal lines each spanning the entire domain, would give good results. This corresponds to rowwise partitioning of the associated Laplacian matrices. We have observed that the hypergraph models for 1D partitioning yield smaller total communication volume than the Cartesian partitioning. Therefore, we tried to find a partitioning method that obtains better results than the Cartesian partitioning. We were able to find a mesh partitioning algorithm that obtains perfect load balance and smaller total volume of communication than the Cartesian partitioning and the 1D hypergraph models. This algorithm is described in Section III. Section IV presents both the comparison of different hypergraph based partitioning methods under different scaling scenarios as well as the comparison of the 1D hypergraph models with the proposed mesh partitioning method.

II. Background

In this section, we provide a brief summary of hypergraphs, hypergraph partitioning, and five-point stencil meshes. We also remind the reader the equivalence between the hypergraph partitioning problem and the partitioning of the finite-element meshes when the objective sought is the reduction of the total communication volume.

A. Hypergraphs and hypergraph partitioning

A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ consists of a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$. Every net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$; these vertices are called the *pins* of n_j. The size of a net is equal to the number of its pins. Weights can be associated with vertices and costs can be associated with nets. For our purposes in this paper each vertex has unit weight and each net has unique cost.

Given a hypergraph $\mathcal{H} = (\mathcal{V},\mathcal{N})$, $\Pi = \{\mathcal{V}_1,\ldots,\mathcal{V}_K\}$ is called a K-way partition of the vertex set $\mathcal{V}$ if each part is

non-empty, parts are pairwise disjoint, and the union of parts gives $\mathcal{V}$. The partitioning constraint is to maintain a balance criterion on part weights, i.e.,

$$|\mathcal{V}_k \ \leq \frac{|\mathcal{V}|}{K}(1+\varepsilon), \quad \text{for } k = 1, 2, \ldots, K \ . \tag{1}$$

In (1), ε represents the predetermined, maximum allowable imbalance ratio.

In a partition Π of $\mathcal{H}$, a net that connects at least one vertex in a part is said to connect that part. *Connectivity set* Λ_j of a net n_j is defined as the set of parts connected by n_j. *Connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. The partitioning objective is to minimize the cutsize defined over the cut nets. There are various cutsize definitions. The relevant cutsize definition for our purposes is:

$$cutsize(\Pi) \quad = \quad \sum_{n_j \in \mathcal{N}} \lambda_j - 1 \ . \tag{2}$$

The hypergraph partitioning problem is known to be NP-hard [7].

B. Five-point stencil meshes and their partitioning

Consider an $M \times N$ mesh corresponding to the discretization of a 2D domain with five-point stencil. Assume that the top leftmost node is denoted by $(1, 1)$ and the bottom rightmost node denoted by (M, N). In this mesh, each node (i, j) has up to four neighbors: one in the north $(i-1, j)$, one in the south $(i+1, j)$, one in the east $(i, j+1)$, and one in the west $(i, j-1)$. It is understood that if any of those neighboring index pair fall outside the range, then the node (i, j) does not have the corresponding neighbor. These meshes are used to obtain finite difference approximations to derivatives at the nodes of the mesh (see, for example, [8, pp.211–212]). In this context, the approximation at a node (i, j) is improved using the approximations at the node itself and the neighboring nodes.

For a node (i, j), let $adj(i, j)$ denote the set of neighboring nodes. Let $madj(i, j)$ denote the set of nodes $adj(i, j) \cup \{(i, j)\}$. Suppose we have partitioned the nodes of the mesh among K processors. We use $part(i, j)$ denotes the owner of the node (i, j). For a node (i, j) define $con(i, j) = \{p : (k, \ell) \in \ madj(i, j) \text{ and } part(k, \ell) = p\}$. Then the node (i, j) necessitates a communication volume of $|con(i, j)| - 1$, where the processor $part(i, j)$ sends messages to all processors in $con(i, j)$ except itself. Let us associate a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with the mesh such that each mesh node (i, j) corresponds to a unique vertex in $\mathcal{V}$, and $madj(i, j)$ corresponds to a unique net in $\mathcal{N}$. Then partitioning the nodes of the mesh among the processors in such a way that each processor gets almost equal number of nodes and the total communication volume is minimized corresponds to the partitioning of the hypergraph $\mathcal{H}$, where the balance criterion (1) is met and the objective (2) is minimized. We note that one can use graph models to partition the nodes of the mesh, but the graph edge-cut metric is not an exact measure of the total communication volume (see [2]).

III. MeshPart: An algorithm to partition the five-point stencil meshes

Although the partitioning of the five-point stencil matrices are very well studied to test the ordering heuristics for sparse matrix factorization [9], [10], they are not studied much for the total volume of communication metric. To the best of our knowledge, only Bisseling [8, Section 4.8] studies the partitioning of these meshes. The objective in that work is to reduce the BSP cost which includes metrics such as maximum volume of messages sent and received by a processor, and hence does not addresses our mesh partitioning problem.

We consider partitioning of the nodes of an $M \times N$, five-point stencil mesh among $K = P \times Q$ processor in such a way that each processor gets the same number of mesh nodes, and the total communication volume is reduced. We propose an algorithm, referred to as MeshPart, for the case $\frac{M}{P} = \frac{N}{Q}$, both ratios are integral, $4 \times \frac{M \times N}{P \times Q}$ is a square number, and $2 \times \frac{M \times N}{P \times Q} \geq 16$. We find the simplest way of implementing the algorithm as follows: first partition a square mesh of $4 \times \frac{M \times N}{P \times Q}$ nodes into four partitions (quadrisection), and then extend the quadrisection to the rest of the mesh by tearing apart and sliding the partitions.

Even though our MeshPart is not generic to partition all possible mesh size dimensions and the number of parts, we believe that it provides very useful insights for achieving good partitioning of the five-point stencil meshes. Furthermore, as we will show in the experimental result section (Section IV), it achieves better results than the existing methods, therefore it becomes a good base case for evaluating other methods.

As an intuitive alternative to the proposed MeshPart algorithm, consider partitioning the nodes of the mesh using a Cartesian partitioning, that is partitioning with only vertical and horizontal lines. In a Cartesian partitioning of an $M \times N$ mesh into $K = P \times Q$ parts, there are $P - 1$ horizontal lines, each of length N, and $Q - 1$ vertical lines, each of length M, where each part gets $(M \times N)/K$ nodes. Let $p(i) = \lfloor (i-1)/P \rfloor + 1$ and $q(j) = \lfloor (j-1)/Q \rfloor + 1$ for $i = 1, \ldots, M$ and $j = 1, \ldots, N$. Then the mesh node (i, j) is assigned to the part $(p(i) - 1) \times Q + q(j)$. It is easy to establish that the total volume of communication resulting from $P \times Q$-Cartesian partitioning of $M \times N$ mesh is given by the formula

$$vol_+(M, N, P, Q) = 2 \times (P-1) \times N + 2 \times (Q-1) \times M \ . \tag{3}$$

In particular when $M = N$, $P = Q = 2$, i.e., in the quadrisection of a square mesh of size $M \times M$ with the Cartesian partitioning, the volume is $4 \times M$.

A. Quadrisection of a square mesh

Assume that we are going to partition an $M \times M$ mesh into four. We note that due to our assumptions $M \geq 16$. The proposed quadrisection algorithm uses slanted lines to partition the mesh, instead of vertical and horizontal ones used in the Cartesian partitioning. At this point, we invite the reader to have a look at the quadrisection of the 16×16 mesh shown in

Fig. 1 to see what we intend to achieve with the quadrisection algorithm described below. We are going to partition the mesh into four in such a way that the four corners of the mesh will be assigned to different parts as follows: $part(1,1) = 1$, $part(1,M) = 2$, $part(M,1) = 3$ and $part(M,M) = 4$. This restriction reveals one of the properties of the quadrisection algorithm we propose: the symmetric (with respect to diagonal and anti-diagonal) mesh nodes of part 1's will be in part 4, and those of part 2's will be in part 3. That is, if $part(i,j) = 1$, then $part(M+1-j, M+1-i) = 4$; similarly, if $part(i,j) = 2$, then $part(M+1-j, M+1-i) = 3$. We note that this holds for the Cartesian quadrisection as well.

The proposed quadrisection algorithm is shown in Algorithm 1. In this algorithm, we first define the nodes of the parts 1 and 4 that are going to be neighbors of some nodes of 2 and 3, then define some nodes of 2 and 3 that are going to be neighbors of some nodes of each other. In our design, this defines all the boundaries. Then we start partitioning all the remaining nodes with a subroutine, bfsColor, we have written. The subroutine bfsColor, given a starting node and a part number, assigns the starting node to the given part, and adds all the neighboring nodes, if not assigned to a part yet, into a queue. Then, bfsColor picks a node from the queue and repeats the process until the queue is empty.

We now discuss the parts 1 and 2. In the 3rd line, we assign the node $(M/2-M/8, M/2-M/8)$ to the part 1. The square block from $(1,1)$ to this node will be assigned to the same part. After this decision, we need to find *target* many nodes to be assigned to the same processor. We achieve this in the while loop of the lines 8–14. During this while loop, the rightmost nodes in some rows are assigned to the part 1, implying that the nodes in the same mesh row, up until that rightmost node are going to be assigned to the same part. The nodes that are symmetric to those nodes are also marked with the same intention. The nodes in the last such row are all marked to be assigned to the part 1 (for loop of lines 15–17). Note that the nodes marked for the part 1 form a symmetric structure along the main diagonal of the mesh. If *target* is odd, we cannot achieve this. We therefore need the assumption that $M \times M$ is divisible by 128. The square block mentioned above has $9 \times M \times M/64$ nodes. Therefore, *target* is $7 \times M \times M/64$. We will have half of this amount below the diagonal and other half above the diagonal. As long as $M \geq 12$, we can find the set of nodes found in the lines 8–17 of Algorithm 1. Since we assume M to be divisible by 8, the smallest such M is 16. After those nodes of the parts 1 and 4, we then define $M/8$ nodes along the main diagonal for each of the parts 2 and 3 in lines 18–19.

Figure 1 displays the quadrisection of the 16×16 mesh obtained by the proposed quadrisection algorithm. In the figure, nodes in different parts are shown with different symbols (and colors). The total volume of communication (vol), the number of nodes (i,j) with $con(i,j) - 1 = 1$ (referred to as boundary-1), and the number of nodes (i,j) with $con(i,j) - 1 = 2$ (referred to as boundary-2) are also shown. The algorithm marks the node $(6,6)$ to be in part 1, which sets *target* as 28.

Algorithm 1 Quadrisection of an $M \times M$ mesh

1: $M_{1/8} \leftarrow \frac{M}{8}$; $M_{1/2} \leftarrow \frac{M}{2}$; $M_{3/8} \leftarrow M_{1/2} - M_{1/8}$
2: $M_{+1} \leftarrow M + 1$
3: $part(M_{3/8}, M_{3/8}) \leftarrow 1$
4: $part(M_{+1} - M_{3/8}, M_{+1} - M_{3/8}) \leftarrow 4$
5: target $\leftarrow M \times M/4 - M_{3/8} \times M_{3/8}$
6: $i \leftarrow M_{3/8}$
7: $j \leftarrow M_{3/8}$
8: **while** target> 0 **do**
9: $\quad i \leftarrow i + 1$; $j \leftarrow j - 1$
10: $\quad$ target$\leftarrow$ target $-2 \times j$
11: $\quad$ **if** target< 0 **then**
12: $\quad\quad j \leftarrow j +$ target$/2$; target $\leftarrow 0$
13: $\quad part(i,j) \leftarrow 1$; $part(M_{+1} - i, M_{+1} - j) \leftarrow 4$
14: $\quad part(j,i) \leftarrow 1$; $part(M_{+1} - j, M_{+1} - i) \leftarrow 4$
15: **for** $k = 1$ **to** j **do**
16: $\quad part(i,k) \leftarrow 1$; $part(M_{+1} - i, M_{+1} - k) \leftarrow 4$
17: $\quad part(k,i) \leftarrow 1$; $part(M_{+1} - k, M_{+1} - i) \leftarrow 4$
18: **for** $k = M_{3/8} + 1$ **to** $M_{1/2}$ **do**
19: $\quad part(k,k) \leftarrow 2$; $part(M_{+1} - k, M_{+1} - k) \leftarrow 3$
20: bfsColor(1, 1, 1); bfsColor(1, M, 2)
21: bfsColor(M, 1, 3); bfsColor(M, M, 4)

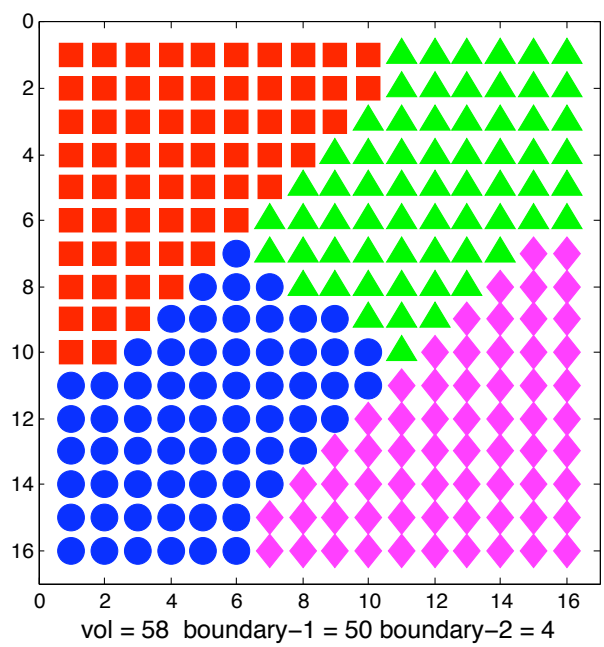


Fig. 1. Quadrisection of the 16×16 mesh

Then the nodes $(7,5)$, $(8,4)$, $(9,3)$, and $(10,2)$ are marked as the rightmost nodes to be assigned to the part 1 in the respective rows of the mesh (the transposes of these nodes are marked to be the bottom most nodes in the respective columns to be assigned to 4). The nodes $(7,7)$ and $(8,8)$ are assigned to the part 2; whereas the associated nodes $(17-7,17-7)$ and $(17-8,17-8)$ are assigned to the part 3. Then the remaining nodes are partitioned using the subroutine bfsColor.

Note that the quadrisection of a square mesh of size $M \times M$ by the proposed method results in a total communication volume of $\frac{7}{2} \times M + 2$. Although we could not prove it yet, we suspect that this might be the best possible. Although there is an equivalence between the mesh partitioning problem (see Section II-B) and the NP-complete hypergraph partitioning problem, the hypergraphs corresponding to the five-point stencil meshes are very special, and hence optimal results might be found in polynomial time. For example, any two vertices corresponding to two nodes which are not neighbors in mesh have at most two nets in common.

B. Extending the quadrisection

We achieve a $P \times Q$ way partitioning of an $M \times N$ mesh by first applying the quadrisection algorithm to a mesh of size $\sqrt{4 \times M/P \times N/Q}$. Then we keep the nodes that belong to parts 1 and 3 (corresponding to the two parts in the first column of the mesh) intact and push the others to the right by an amount of $M/P = N/Q$. This operation opens up a space for two parts, each will have $M/P \times N/Q$ nodes, and one of them (say the upper part) will contain the new nodes at the mesh boundary $i = 1$ and the other (say the lower part) will contain the new nodes at $2 \times M/P$. We define the boundary between these two parts and then assign the new nodes to parts by again using the subroutine bfsColor. The boundary between the two newly added parts is defined as follows. Let $n = 2 \times M/P$, and consider the nodes in the line joining the nodes $(n/2+n/8, n/2+n/8+1)$ and $(n+1-n/2-n/8, n+1-(n/2+n/8+1)+n/2)$. There are a total of $n/4$ nodes (including the two nodes as defined). We mark the first half of these, i.e., $n/8$ of them, to be assigned to the upper part, and the second half to be assigned to the lower part. Then calling bfsColor on the node $(n/2+n/8-1, n/2+n/8+1)$ with color "upper", and on the node $(n+1-n/2-n/8+1, n+1-(n/2+n/8+1)+n/2)$ with color "lower" results in the partitioning of the $(2 \times M/P) \times (3 \times N/Q)$ mesh into 6 parts. This is seen in Fig. 2(a). We repeat this process until we obtain $2 \times Q$-way partitioning of the mesh of size $(2 \times M/P) \times N$.

A similar procedure is run to extend the $2 \times Q$-way partitioning into $3 \times Q$ partitioning; which is then repeatedly used to obtain $P \times Q$-way partitioning of the given original mesh. In this rowwise extension process, the Q parts that have nodes in the mesh boundary $i = 1$ are kept intact and the rest are pushed down by an amount of $M/P = N/Q$, and $Q-1$ boundaries among the Q new coming parts are defined. This is best seen again in the example of Fig. 2. After the 2×4-way partitioning of the 16×32 mesh (not displayed), we extend the mesh to 24×32 by pushing the nodes in the parts containing nodes at $i = 16$ downwards by 8. Then the boundaries between the nodes are defined, yielding 3×4-way partitioning of the 24×32 mesh as shown in Fig. 2(b). Then the nodes in the parts containing nodes at $i = 1$ are kept intact but others are pushed downwards by again 8 to open up space for the last four parts. Again, the boundaries between the new coming parts are defined, yielding the $16 = 4 \times 4$-way partitioning of the 32×32 mesh as shown in Fig. 2(c). We note that the four parts in the four corners of the partition shown in Fig. 2(c) are the ones that we obtained by the quadrisection algorithm at the very beginning.

C. Analysis

As stated before, the volume of communication resulting from the quadrisection of the $M \times M$ mesh with the proposed algorithm is given by the formula

$$vol(M, M, 2, 2) = \frac{7 \times M}{2} + 2 \ . \tag{4}$$

This can be derived by tracing the algorithm. We have found that the total volume of communication of the $P \times Q$-way partitioning of the $M \times N$ mesh with the proposed MeshPart method behaves according to the formula

$$\begin{aligned} vol(M, N, P, Q) = {} & (3 \times P \times Q - (P+Q) - 1) \times n \\ & + (P-1) \times (3 \times Q - 5) \\ & + (Q-1) \times (3 \times P - 5) \ , \end{aligned} \tag{5}$$

where $n = M/P = N/Q$. Notice that with $M = N$ and $P = Q = 2$, this checks with (4). We have observed this outcome experimentally (see Table 3) but have not proved it at the time of writing.

The number of nodes (i,j) with $|con(i,j)| - 1 = 2$ is given by the formula

$$boundary_2(P, Q) = 4 \times (P-1) \times (Q-1) \ . \tag{6}$$

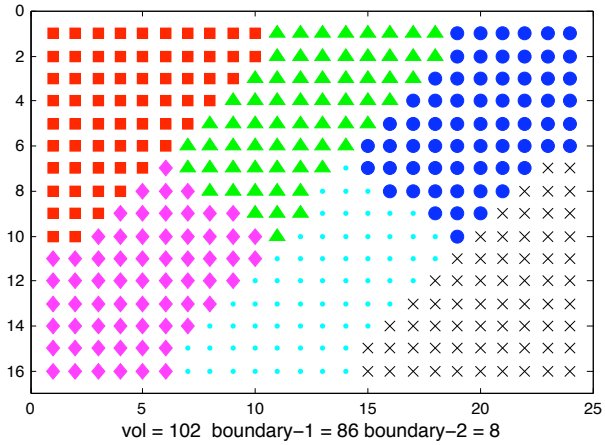
Note that each of these nodes necessitates a total communication volume of 2. Hence, with the assumption that the relation (5) holds, the number of boundary nodes (i,j) with $|con(i,j)| - 1 = 1$ (that is the number of nodes which necessitate a communication volume of 1) is given by the formula

$$\begin{aligned} boundary_1(M, N, P, Q) = {} & vol(M, N, P, Q) \\ & - 2 \times boundary_2(P, Q) \ . \end{aligned} \tag{7}$$

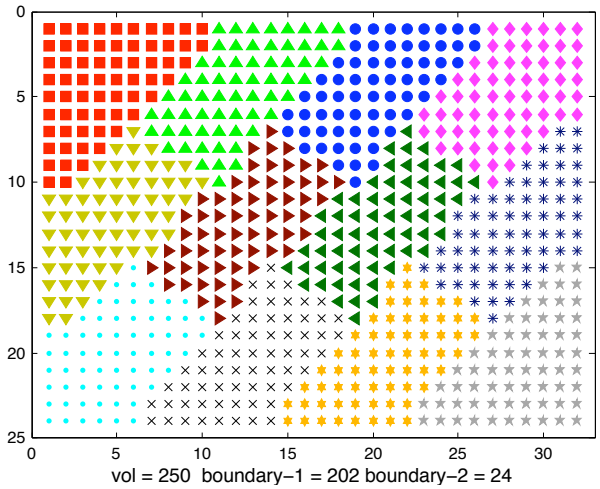
IV. Experimental Results

We ran our tests using PaToH Matlab Matrix-Partitioning Interface [11], [12] on a dual quad-core 2.26 GHz Intel Xeon desktop with 24 GB of memory using Matlab v7.8 (R2009a).

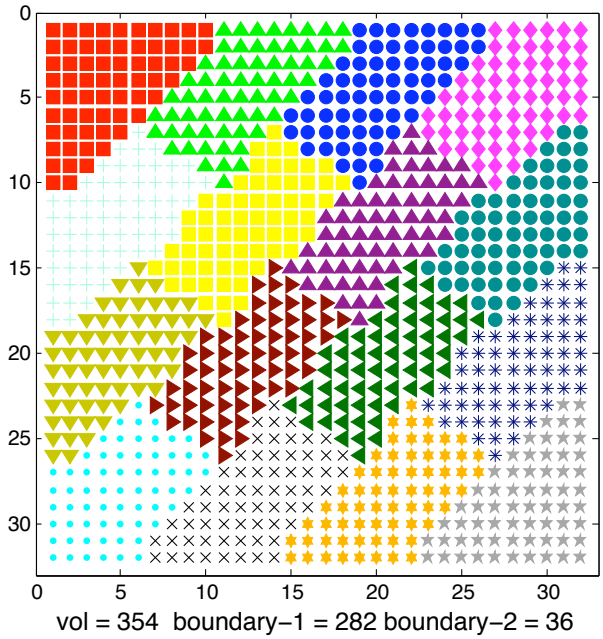
In our experiments, we have used five different hypergraph partitioning methods: two 1D matrix partitioning methods, namely Rowwise (RW) and Columnwise (CW) partitionings [1], [2], and three 2D partitioning methods, namely Fine-grain (FG) [3], Checkerboard (CH) [4] and Jagged-like (JL) [5].



(a) 2×3-way partitioning of the 16×24 mesh



(b) 3×4-way partitioning of the 24×32 mesh



(c) 16-way partitioning of the 32×32 mesh

Fig. 2. Steps for 16-way partitioning of the 32×32 mesh

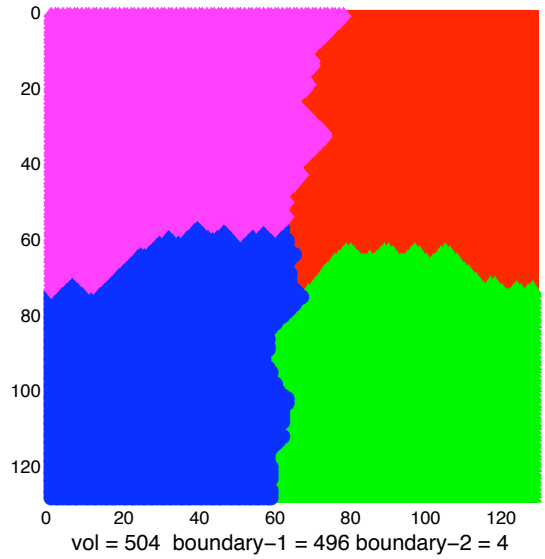


Fig. 3. A sample 4-way partitioning of the 128×128 mesh with hypergraph partitioning.

In the first set of scalability experiments, we evaluated our new MeshPart algorithm comparing its results to Cartesian partitioning as well as hypergraph partitioning. In order to perform a fair comparison, we have only used 1D hypergraph partitioning which produces partitioning of mesh nodes. One could also partition the mesh matrices with 2D partitioning methods (this would be equivalent to partitioning the edges of the mesh). Note that since these matrices are symmetric, 1D RW and CW partitioning methods are equivalent. Table I displays the partitioning results for meshes of sizes from 64×64 to 2048×2048 with varying number of parts K. In this table, we only display partitionings that would yield at least 100 vertices in each part. As seen in the results, the Cartesian partitioning produced results that are on the average 21% worse than those of the proposed MeshPart method. 1D hypergraph partitioning produces better results than the Cartesian partitioning, but it is, too, worse than MeshPart. Figure 3 displays a sample 4-way partitioning of 128×128 mesh using 1D hypergraph partitioning. As seen in the figure, the partitioning result looks somewhat in between Cartesian partitioning result and MeshPart result as one would expect. An interesting trend is that for a given mesh size, generally, the relative total communication volume of Cartesian and 1D hypergraph partitionings first increases with the number of parts, then decreases. The fact that this last trend holds for the Cartesian partitioning can be verified by looking at the total communication volume formulas. That, the same trend holds for the hypergraph partitioning is observed on meshes of sizes 960 and 1920 as well with $K = 4, 16, 25, 36, 64, 100, 144, 576, 900, 1600, 3600$ for the smaller one, and in addition to those eleven Ks with 256 and 2304 for the larger one.

The second set of experiments is designed to evaluate performance of hypergraph models with the increasing problem

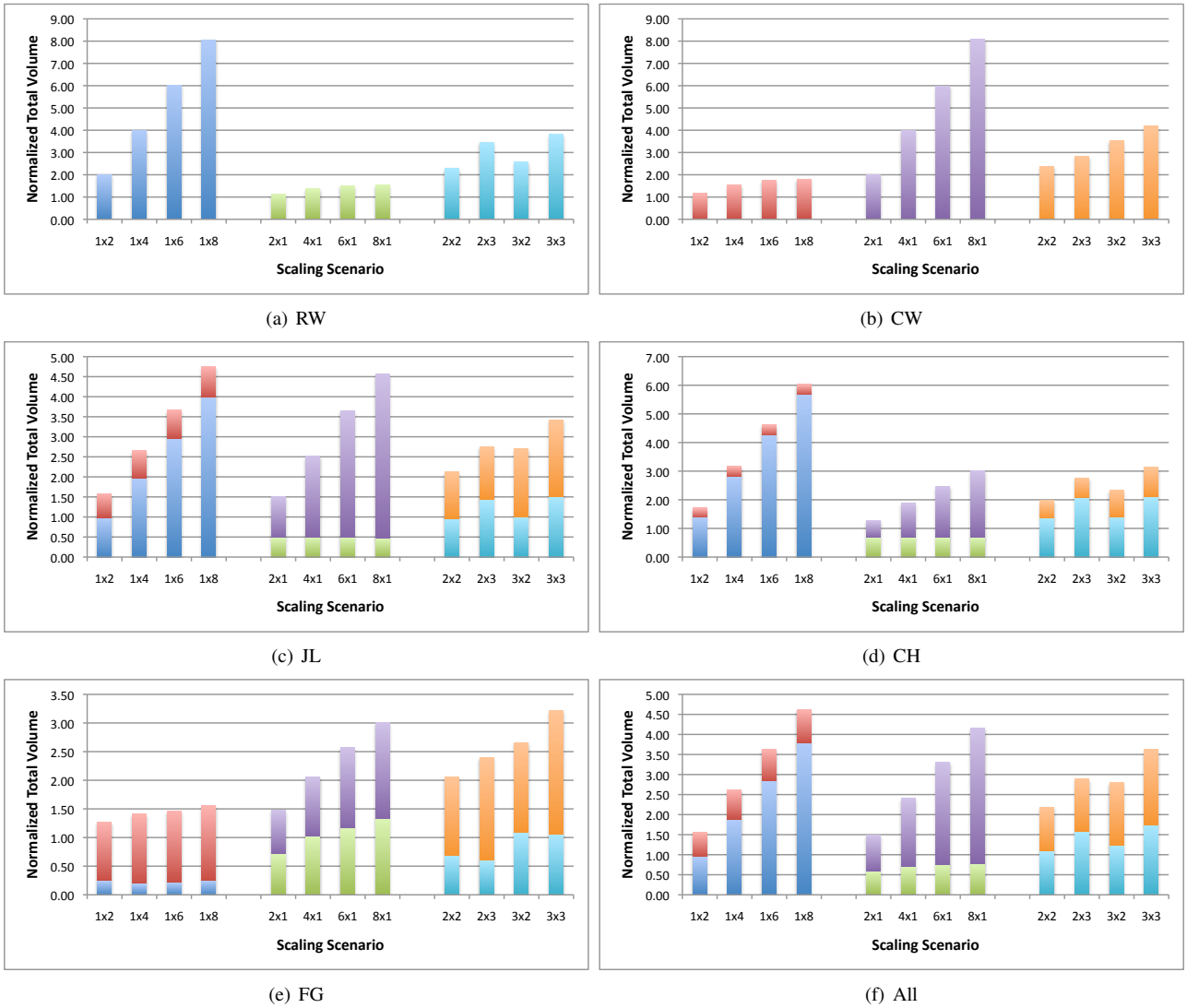


Fig. 4. Comparison of hypergraph based partitioning methods. For all partitioning instances and scaling scenarios, we first normalized the total communication volume with respect to the total communication of the original matrix, then present average results over all matrices and number of parts.

size (both matrix size and the number of parts). For this second set of experiments, we picked six matrices (Table II) from University of Florida Matrix Collection [13], then systematically replicated its rows and/or columns to create larger problem instances. The advantage of this approach is that under ideal conditions (such as if the solution is not strictly restricted by the balance constraints), we could easily formulate total communication volume for the larger problem instances, hence we can discuss how the methods scale with the increasing problem size. In this section, a *scaling scenario* $R \times C$ means that each row of the matrix is replicated R times and each column of the matrix is replicated C times.

We tested with $K \in \{2, 4, 8, 16, 32, 64, 128, 256\}$. For a specific K value, K-way partitioning of a test matrix constitutes a partitioning instance. The partitioning instances in which $\min\{M, N\} < 100 \times K$ are discarded, as the parts would become too small to be meaningful. Averages over all K values for five hypergraph models, together with an overall average of all methods, is displayed in Figure 4. For 2D methods (i.e., JL, CH, FG) and for average of all methods (Fig. 4(f)), the normalized total communication volume is displayed as a stacked bar to illustrate normalized expand and fold communications. In these charts, bottom of the bar represents the total volume of expand communications and top represents volume of fold communications.

In this experiment, when only columns (rows) of the matrix is replicated, we expect the total communication volume of RW (CW) partitioning method to increase linearly with the number of replication due to the linear increase in the number of data elements that needs to be communicated. Our results displayed in Figs. 4(a) and 4(b) (including the last four bars showing scenarios with replication in both rows and columns) confirm this expectation. Similarly, when only rows (columns) of the matrix are replicated, we expect the total communication

TABLE I
COMPARISON OF TOTAL COMMUNICATION VOLUME FOUND BY MESHPART ALGORITHM WITH RESPECT TO THOSE OF CARTESIAN PARTITIONING AND 1D HYPERGRAPH PARTITIONING. NUMBERS IN PARENTHESIS ARE NORMALIZED TOTAL COMMUNICATION VOLUME WITH RESPECT TO THOSE OF MESHPART.

Mesh Size	K	MeshPart	Cartesian Part		1D Hypergraph	
64x64	4	226	256	(1.13)	252	(1.11)
64x64	16	666	768	(1.15)	739	(1.11)
128x128	4	450	512	(1.14)	504	(1.12)
128x128	16	1290	1536	(1.19)	1475	(1.14)
128x128	64	3066	3584	(1.17)	3353	(1.09)
256x256	4	898	1024	(1.14)	1015	(1.13)
256x256	16	2538	3072	(1.21)	2979	(1.17)
256x256	64	5866	7168	(1.22)	6736	(1.15)
256x256	256	13050	15360	(1.18)	13893	(1.06)
512x512	4	1794	2048	(1.14)	2051	(1.14)
512x512	16	5034	6144	(1.22)	6272	(1.25)
512x512	64	11466	14336	(1.25)	13648	(1.19)
512x512	256	24810	30720	(1.24)	28135	(1.13)
512x512	1024	53754	63488	(1.18)	56306	(1.05)
1024x1024	4	3586	4096	(1.14)	4194	(1.17)
1024x1024	16	10026	12288	(1.23)	12251	(1.22)
1024x1024	64	22666	28672	(1.26)	28279	(1.25)
1024x1024	256	48330	61440	(1.27)	58598	(1.21)
1024x1024	1024	101866	126976	(1.25)	114223	(1.12)
2048x2048	4	7170	8192	(1.14)	8463	(1.18)
2048x2048	16	20010	24576	(1.23)	24382	(1.22)
2048x2048	64	45066	57344	(1.27)	56890	(1.26)
2048x2048	256	95370	122880	(1.29)	117996	(1.24)
2048x2048	1024	198090	253952	(1.28)	234477	(1.18)
average				(1.21)		(1.16)

TABLE II
PROPERTIES OF THE TEST MATRICES.

	Number of		
name	rows	columns	nonzeros
lp_dfl001	6,071	12,230	35,632
shermanACb	18,510	18,510	145,149
mult_dcop_01	25,187	25,187	193,276
lp_cre_b	9,648	77,137	260,785
lp_nug30	52,260	379,350	1,567,800
Stanford	281,903	281,903	2,312,497

volume of RW (CW) partitioning method to remain about the same, because one can achieve this volume simply by assigning replica rows (columns) to the same part with the respective row (column) of the original matrix. However, since we are using an heuristic method which does not use concepts like supernodes [10]—vertices that have identical net sets—, the solutions can be a little different than the expected outcomes. It is seen that the normalized total volume for RW increases up to 1.55 times, whereas for CW it increases up to 1.81 times. We believe that this discrepancy is due to the shapes of the matrices we used in our experiments. Three out of six matrices are rectangular matrices with substantially more columns than rows. In these cases, column replication increases the number of vertices of the hypergraph in CW partitioning and makes the number of nets to number of vertices ratio substantially smaller. This result suggests that, similar to identical net elimination techniques, hypergraph partitioning tools should consider implementing identical vertex elimination (supernode detection).

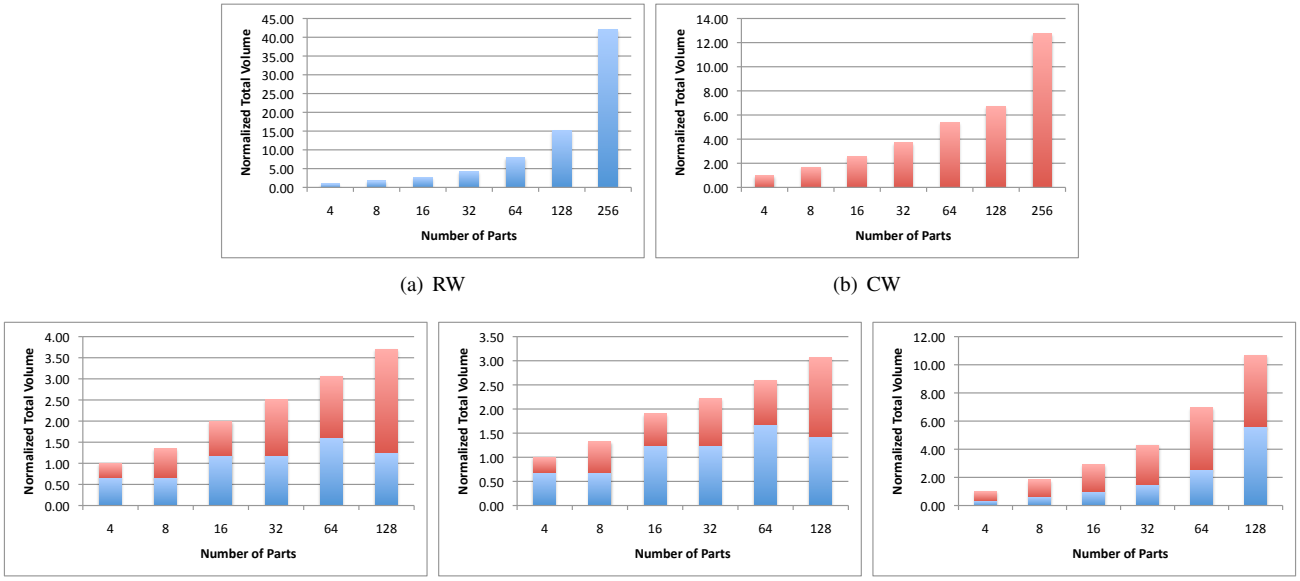
As explained in [4], [5], JL and CH methods can be applied by using rowwise partitioning first followed by columnwise partitioning, and vice versa. In our experiments, without any particular reason, we only test the former approach. This inherently effected the results, especially for CH and FG, because of the properties of our test matrices. For CH where second partitioning is a multi-constraint columnwise partitioning, column replication makes the multi-constraint partitioning harder, in comparison to row replication. Hence the different scaling behavior in CH with row versus column replication. For FG, this means that FG will find solutions "closer" to CW in wide rectangular matrices, and when we compare the trends in Fig. 4(e) with Fig. 4(b) we notice that they are similar but, as expected, FG scales better. One general and expected observation is that the 2D partitioning methods scale better than the 1D partitioning methods. That is, their normalized total communication volume increases less with the increased number of replication.

The last figure (Fig. 5) displays the performance of partitioning methods with the increasing number of parts. We expect a more steep increase in the normalized communication volume with the 1D partitioning methods, where the results displayed in Figs. 5(a) and 5(b) confirm this expectation. Unfortunately, the shape of our test matrices also shows its effects in this experiment. Due to the wide rectangular matrices, partitioning into larger K values does not scale well in the RW method in comparison to the CW method. When compared to FG, the normalized total communication volume of the JL and CH methods increase much slower with the increasing number of parts, and hence creating an illusion of JL and CH scaling better than FG in these results. However, we need to note that absolute value of the total communication volume for JL and CH methods are noticeably higher than those of FG—in some instances even 256-way FG partitioning produces smaller total communication than 4-way those found by JL and CH. Nevertheless, these results once more confirms that 2D methods scale better than 1D partitioning methods. One final note about this result is that FG shows a similar scaling pattern with our MeshPart method.

V. CONCLUSION

We investigated the scalability of the hypergraph partitioning methods under different scenarios for which an educated guess as to how the best algorithm would behave can be made. Experimental results showed that the 2D partitioning methods scale better than the 1D partitioning methods. The increase in the total communication volume of the 2D methods is smaller than the increase in that of the 1D methods for the increasing problem size and/or number of parts. In both the 1D and 2D methods, the increases in the total communication volume are smaller than the increase in the number of parts. Results also suggest that hypergraph partitioning tools should consider implementing identical net and vertex elimination (supernode detection) to improve the solution quality.

For another scalability study, we investigated the performance of the hypergraph partitioning methods on the matrices



(a) RW (b) CW

(c) JL (d) CH (e) FG

Fig. 5. Comparison of hypergraph based partitioning methods. For all partitioning instances and scaling scenarios we first normalized the total communication volume with respective the total communication of the corresponding partitioning for $K = 4$ then here present average results over all matrices and all scenarios.

arising from the discretization of two-dimensional domains with the five-point stencil. For such matrices, we proposed a partitioning method, MeshPart, which uses the domain specific knowledge to partition the nodes of the mesh (and hence the rows of the matrices). The proposed MeshPart method assigns equal number of mesh nodes to processors and yields better total volume of communication than the hypergraph models. Although we were not able to prove yet, we think that it might yield the best possible total communication volume for four way partitioning of the two-dimensional meshes in question. We used this method to investigate the scalability of the 1D hypergraph partitioning methods. Results showed that 1D hypergraph partitioning method scales reasonably well but the results are still far from optimum—this calls for better partitioning heuristics.

We think that the current work sheds light into the scalability of the hypergraph models. However, there is still much to do in order to address the question: which method would yield the best results for a given matrix and the number of processors, and what would the resulting total communication volume be?

A few research directions arise regarding the proposed mesh partitioning method. First, we are investigating the use of the proposed mesh partitioning routine for developing fill-reducing ordering methods for sparse matrices, again corresponding to the finite element meshes. Second, we are trying to answer the following questions. How can one partition the meshes that are built using the 7-point and 9-point stencils? How can one generalize these methods to the meshes arising from the discretization of 3D domains? Can we generalize the these methods to address partitioning of irregular meshes? In other words, can we infer heuristic approaches for the hypergraph partitioning problem?

References

[1] U. V. Çatalyürek and C. Aykanat, "A hypergraph model for mapping repeated sparse matrix-vector product computations onto multicomputers," in *Proceedings of International Conference on High Performance Computing*, Dec. 1995.

[2] Ü. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions Parallel and Distributed Systems*, vol. 10, no. 7, pp. 673–693, 1999.

[3] ——, "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," The Ohio State University, Department of Biomedical Informatics, Tech. Rep. TR_2008_n04, 2008, to appear in SIAM J. Sci. Comput.

[6] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Review*, vol. 47, no. 1, pp. 67–95, 2005.

[7] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, U.K.: Wiley–Teubner, 1990.

[8] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach Using BSP and MPI*. Oxford University Press, 2004.

[9] J. A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, Apr 1973.

[10] J. A. George and J. W. H. Liu, *Computer solution of large sparse positive definite systems*. Prentice-Hall, 1981.

[11] Ü. V. Çatalyürek and C. Aykanat, "PaToH: A multilevel hypergraph partitioning tool, version 3.0," Computer Engineering Department, Bilkent University, Tech. Rep. BU-CE-9915, 1999.

[12] B. Uçar, C. Aykanat, and U. V. Catalyurek, "A matrix partitioning interface to PaToH in matlab," Presented at the 5th International Workshop on Parallel Matrix Algorithms and Applications (PMAA'08), Neuchâtel, Switzerland, June 2008.

[13] T. Davis, "The University of Florida sparse matrix collection," CISE Department, University of Florida, Gainesville, FL, USA, Tech. Rep. REP-2007-298, 2007.