13. Tucker LW, Robertson GG (1988) Architecture and applications of the connection machines. IEEE Comput 21(8):26–38
14. Becker J, Sterling D, Savarese T, Dorband JE, Ranawake UA, Packer CV (1995) Beowulf: a parallel workstation for scientific computation. In: Proceedings of ICPP workshop on challenges for parallel processing, CRC Press, Oconomowc, August 1995
15. Seitz CL, Athas W, Flaig C, Martin A, Seieovic J, Steele CS, Su WK (1988) The architecture and programming of the ametek series 2010 multicomputer. In: Proceedings of the third conference on hypercube concurrent computers and applications, Pasadena, 19–20 Jan 1988, pp 31–36

Hypergraph Partitioning

Ümit V. Çatalyürek[1], Bora Uçar[2], Cevdet Aykanat[3]
[1]The Ohio State University, Columbus, OH, USA
[2]ENS Lyon, Lyon, France
[3]Bilkent University, Ankara, Turkey

Definition

Hypergraphs are generalization of graphs where each edge (hyperedge) can connect more than two vertices. In simple terms, the hypergraph partitioning problem can be defined as the task of dividing a hypergraph into two or more roughly equal-sized parts such that a cost function on the hyperedges connecting vertices in different parts is minimized.

Discussion

Introduction

During the last decade, hypergraph-based models gained wide acceptance in the parallel computing community for modeling various problems. By providing natural way to represent multiway interactions and unsymmetric dependencies, hypergraph can be used to elegantly model complex computational structures in parallel computing. Here, some concrete applications will be presented to show how hypergraph models can be used to cast a suitable scientific problem as an hypergraph partitioning problem. Some insights and general guidelines for using hypergraph partitioning methods in some general classes of problems are also given.

Formal Definition of Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices (cells) $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. Every net $n \in \mathcal{N}$ is a subset of vertices, that is, $n \subseteq \mathcal{V}$. The vertices in a net n are called its *pins*. The *size* of a net is equal to the number of its pins. The *degree* of a vertex is equal to the number of nets it is connected to. Graph is a special instance of hypergraph such that each net has exactly two pins. Vertices can be associated with weights, denoted with $w[\cdot]$, and nets can be associated with costs, denoted with $c[\cdot]$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a *K-way partition* of $\mathcal{H}$ if the following conditions hold:

- Each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, that is, $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$
- Parts are pairwise disjoint, that is, $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$
- Union of K parts is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$

In a partition Π of $\mathcal{H}$, a net that has at least one pin (vertex) in a part is said to *connect* that part. *Connectivity* λ_n of a net n denotes the number of parts connected by n. A net n is said to be *cut* (*external*) if it connects more than one part (i.e., $\lambda_n > 1$), and *uncut* (*internal*) otherwise (i.e., $\lambda_n = 1$). A partition is said to be balanced if each part $\mathcal{V}_k$ satisfies the *balance criterion*:

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

In (1), weight W_k of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in that part (i.e., $W_k = \sum_{v \in \mathcal{V}_k} w[v]$), W_{avg} denotes the weight of each part under the perfect load balance condition (i.e., $W_{avg} = (\sum_{v \in \mathcal{V}} w[v])/K$), and ε represents the predetermined maximum imbalance ratio allowed.

The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. There are various [20] *cutsize* definitions for representing the cost $\chi(\Pi)$ of a partition Π. Two relevant definitions are:

$$\chi(\Pi) = \sum_{n \in \mathcal{N}_E} c[n] \tag{2}$$

$$\chi(\Pi) = \sum_{n \in \mathcal{N}_E} c[n](\lambda_n - 1). \tag{3}$$

In (2), the cutsize is equal to the sum of the costs of the cut nets. In (3), each cut net n contributes $c[n](\lambda_n - 1)$ to the cutsize. The cutsize metrics given in (2) and (3)

will be referred to here as *cut-net* and *connectivity* metrics, respectively. The hypergraph partitioning problem can be defined as the task of dividing a hypergraph into two or more parts such that the cutsize is minimized, while a given balance criterion (1) among part weights is maintained.

A recent variant of the above problem is the *multi-constraint hypergraph* partitioning [9, 18] in which each vertex has a vector of weights associated with it. The partitioning objective is the same as above, and the partitioning constraint is to satisfy a balancing constraint associated with each weight. Here, $w[v,i]$ denotes the C weights of a vertex v for $i = 1,\ldots,C$. Hence, the balance criterion (1) can be rewritten as

$$W_{k,i} \leq W_{avg,i}\,(1+\varepsilon) \text{ for } k = 1,\ldots,K \text{ and } i = 1,\ldots,C\,, \tag{4}$$

where the ith weight $W_{k,i}$ of a part $\mathcal{V}_k$ is defined as the sum of the ith weights of the vertices in that part (i.e., $W_{k,i} = \sum_{v\in\mathcal{V}_k} w[v,i]$), and $W_{avg,i}$ is the average part weight for the ith weight (i.e., $W_{avg,i} = (\sum_{v\in\mathcal{V}} w[v,i])/K$), and ε again represents the allowed imbalance ratio.

Another variant is the *hypergraph partitioning with fixed vertices*, in which some of the vertices are fixed in some parts before partitioning. In other words, in this problem, a *fixed-part* function is provided as an input to the problem. A vertex is said to be *free* if it is allowed to be in any part in the final partition, and it is said to be fixed in part k if it is required to be in $\mathcal{V}_k$ in the final partition Π.

Yet another variant is *multi-objective hypergraph partitioning* in which there are several objectives to be minimized [1, 21]. Specifically, a given net contributes different costs to different objectives.

Sparse Matrix Partitioning

One of the most elaborated applications of hypergraph partitioning (HP) method in the parallel scientific computing domain is the parallelization of sparse matrix-vector multiply (SpMxV) operation. Repeated matrix-vector and matrix-transpose-vector multiplies that involve the same large, sparse matrix are the kernel operations in various iterative algorithms involving sparse linear systems. Such iterative algorithms include solvers for linear systems, eigenvalues, and linear programs. The pervasive use of such solvers motivates the development of HP models and methods for efficient parallelization of SpMxV operations.

Before discussing the HP models and methods for parallelizing SpMxV operations, it is favorable to discuss parallel algorithms for SpMxV. Consider the matrix-vector multiply of the form $y \leftarrow \mathbf{A}x$, where the nonzeros of the sparse matrix $\mathbf{A}$ as well as the entries of the input and output vectors x and y are partitioned arbitrarily among the processors. Let $map(\cdot)$ denote the nonzero-to-processor and vector-entry-to-processor assignments induced by this partitioning. A parallel algorithm would execute the following steps at each processor P_k.

1. Send the local input-vector entries x_j, for all j with $map(x_j) = P_k$, to those processors that have at least one nonzero in column j.
2. Compute the scalar products $a_{ij}\,x_j$ for the local nonzeros, that is, the nonzeros for which $map(a_{ij}) = P_k$ and accumulate the results y_i^k for the same row index i.
3. Send local nonzero partial results y_i^k to the processor $map(y_i) \neq P_k$, for all nonzero y_i^k.
4. Add the partial y_i^ℓ results received to compute the final results $y_i = \sum y_i^\ell$ for each i with $map(y_i) = P_k$.

As seen in the algorithm, it is necessary to have partitions on the matrix $\mathbf{A}$ and the input- and output-vectors x and y of the matrix-vector multiply operation. Finding a partition on the vectors x and y is referred to as the vector partitioning operation, and it can be performed in three different ways: by decoding the partition given on $\mathbf{A}$; in a post-processing step using the partition on the matrix; or explicitly partitioning the vectors during partitioning the matrix. In any of these cases, the vector partitioning for matrix-vector operations is called *symmetric* if x and y have the same partition, and *non-symmetric* otherwise. A vector partitioning is said to be *consistent*, if each vector entry is assigned to a processor that has at least one nonzero in the respective row or column of the matrix. The consistency is easy to achieve for the nonsymmetric vector partitioning; x_j can be assigned to any of the processors that has a nonzero in the column j, and y_i can be assigned to any of the processors that has a nonzero in the row i. If a symmetric vector partitioning is sought, then special care must be taken to assign a pair of matching input- and output-vector entries, e.g., x_i and

y_i, to a processor having nonzeros in both row and column i. In order to have such a processor for all vector entry pairs, the sparsity pattern of the matrix $\mathbf{A}$ can be modified to have a zero-free diagonal. In such cases, a consistent vector partition is guaranteed to exist, because the processors that own the diagonal entries can also own the corresponding input- and output-vector entries; x_i and y_i can be assigned to the processor that holds the diagonal entry a_{ii}.

In order to achieve an efficient parallelism, the processors should have balanced computational load and the inter-processor communication cost should have been minimized. In order to have balanced computational load, it suffices to have almost equal number of nonzeros per processor so that each processor will perform almost equal number of scalar products, for example, $a_{ij}x_j$, in any given parallel system. The communication cost, however, has many components (the total volume of messages, the total number of messages, maximum volume/number of messages in a single processor, either in terms of sends or receives or both) each of which can be of utmost importance for a given matrix in a given parallel system. Although there are alternatives and more elaborate proposals, the most common communication cost metric addressed in hypergraph partitioning-based methods is the total volume of communication.

Loosely speaking, hypergraph partitioning-based methods for efficient parallelization of SpMxV model the data of the SpMxV (i.e., matrix and vector entries) with the vertices of a hypergraph. A partition on the vertices of the hypergraph is then interpreted in such a way that the data corresponding to a set of vertices in a part are assigned to a single processor. More accurately, there are two classes of hypergraph partitioning-based methods to parallelizing SpMxV. The methods in the first class build a hypergraph model representing the data and invoke a partitioning heuristic on the so-built hypergraph. The methods in this class can be said to be models rather than being algorithms. There are currently three main hypergraph models for representing sparse matrices, and hence there are three methods in this first class. These three main models are described below in the next section. Essential property of these models is that the cutsize (3) of any given partition is equal to the total communication volume to be incurred under a consistent vector partitioning when the matrix elements are distributed according to the vertex partition. The methods in the second class follow a mix-and-match approach and use the three main models, perhaps, along with multi-constraint and fixed-vertex variations in an algorithmic form. There are a number of methods in this second class, and one can develop many others according to application needs and matrix characteristics. Three common methods belonging to this class are described later, after the three main models. The main property of these algorithms is that the sum of the cutsizes of each application of hypergraph partitioning amounts to the total communication volume to be incurred under a consistent vector partitioning (currently these methods compute a vector partitioning after having found a matrix partitioning) when the matrix elements are distributed according to the vertex partitions found at the end.

Three Main Models for Matrix Partitioning

In the *column-net hypergraph model* [11] used for 1D rowwise partitioning, an $M \times N$ matrix $\mathbf{A}$ with Z nonzeros is represented as a unit-cost hypergraph $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}})$ with $|\mathcal{V}_R| = M$ vertices, $|\mathcal{N}_C| = N$ nets, and Z pins. In $\mathcal{H}_{\mathcal{R}}$, there exists one vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ for each row i of matrix $\mathbf{A}$. Weight $w[v_i]$ of a vertex v_i is equal to the number of nonzeros in row i. The name of the model comes from the fact that columns are represented as nets. That is, there exists one unit-cost net $n_j \in \mathcal{N}_{\mathcal{C}}$ for each column j of matrix $\mathbf{A}$. Net n_j connects the vertices corresponding to the rows that have a nonzero in column j. That is, $v_i \in n_j$ if and only if $a_{ij} \neq 0$.

In the *row-net hypergraph model* [11] used for 1D columnwise partitioning, an $M \times N$ matrix $\mathbf{A}$ with Z nonzeros is represented as a unit-cost hypergraph $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ with $|\mathcal{V}_C| = N$ vertices, $|\mathcal{N}_R| = M$ nets, and Z pins. In $\mathcal{H}_{\mathcal{C}}$, there exists one vertex $v_j \in \mathcal{V}_{\mathcal{C}}$ for each column j of matrix $\mathbf{A}$. Weight $w[v_j]$ of a vertex $v_j \in \mathcal{V}_{\mathcal{R}}$ is equal to the number of nonzeros in column j. The name of the model comes from the fact that rows are represented as nets. That is, there exists one unit-cost net $n_i \in \mathcal{N}_{\mathcal{R}}$ for each row i of matrix $\mathbf{A}$. Net $n_i \subseteq \mathcal{V}_{\mathcal{C}}$ connects the vertices corresponding to the columns that have a nonzero in row i. That is, $v_j \in n_i$ if and only if $a_{ij} \neq 0$.

In the *column-row-net hypergraph model*, otherwise known as the *fine-grain model* [13], used for 2D

nonzero-based fine-grain partitioning, an $M \times N$ matrix $\mathbf{A}$ with Z nonzeros is represented as a unit-weight and unit-cost hypergraph $\mathcal{H}_{\mathcal{Z}} = (\mathcal{V}_{\mathcal{Z}}, \mathcal{N}_{\mathcal{RC}})$ with $|\mathcal{V}_{\mathcal{Z}}| = Z$ vertices, $|\mathcal{N}_{\mathcal{RC}}| = M + N$ nets and $2Z$ pins. In $\mathcal{V}_Z$, there exists one unit-weight vertex v_{ij} for each nonzero a_{ij} of matrix $\mathbf{A}$. The name of the model comes from the fact that both rows and columns are represented as nets. That is, in $\mathcal{N}_{\mathcal{RC}}$, there exist one unit-cost row-net r_i for each row i of matrix $\mathbf{A}$ and one unit-cost column-net c_j for each column j of matrix A. The row-net r_i connects the vertices corresponding to the nonzeros in row i of matrix $\mathbf{A}$, and the column-net c_j connects the vertices corresponding to the nonzeros in column j of matrix $\mathbf{A}$. That is, $v_{ij} \in r_i$ and $v_{ij} \in c_j$ if and only if $a_{ij} \neq 0$. Note that each vertex v_{ij} is in exactly two nets.

Some Other Methods for Matrix Partitioning

The *jagged-like partitioning method* [16] uses the row-net and column-net hypergraph models. It is an algorithm with two steps, in which each step models either the *expand phase* (the 1st line) or the *fold phase* (the 3rd line) of the parallel SpMxV algorithm given above. Therefore, there are two alternative schemes for this partitioning method. The one which models the expands in the first step and the folds in the second step is described below.

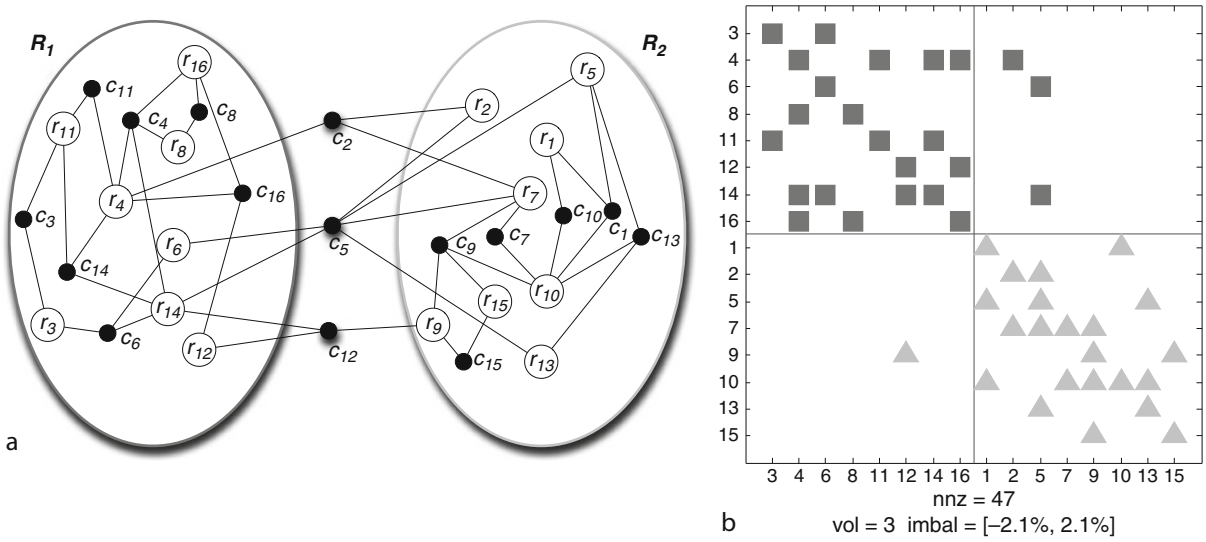
Given an $M \times N$ matrix $\mathbf{A}$ and the number K of processors organized as a $P \times Q$ mesh, the jagged-like partitioning model proceeds as shown in Fig. 1. The algorithm has two main steps. First, $\mathbf{A}$ is partitioned rowwise into P parts using the column-net hypergraph model $\mathcal{H}_{\mathcal{R}}$ (lines 1 and 2 of Fig. 1). Consider a P-way partition $\Pi_{\mathcal{R}}$ of $\mathcal{H}_{\mathcal{R}}$. From the partition $\Pi_{\mathcal{R}}$, one obtains P submatrices $\mathbf{A}_p$, for $p = 1, \ldots, P$ each having roughly equal number of nonzeros. For each p, the rows of the submatrix $\mathbf{A}_p$ correspond to the vertices in $\mathcal{R}_p$ (lines 6 and 7 of Fig. 1). The submatrix $\mathbf{A}_p$ is assigned to the pth row of the processor mesh. Second, each submatrix $\mathbf{A}_p$ for $1 \leq p \leq P$ is independently partitioned columnwise into Q parts using the row-net hypergraph $\mathcal{H}_p$ (lines 8 and 9 of Fig. 1). The nonzeros in the ith row of $\mathbf{A}$ are partitioned among the Q processors in a row of the processor mesh. In particular, if $v_i \in \mathcal{R}_p$ at the end of line 2 of the algorithm, then the nonzeros in the ith row of $\mathbf{A}$ are partitioned among the processors in the pth row of the processor mesh. After partitioning the submatrix $\mathbf{A}_p$ columnwise, the *map* array contains the partition information for the nonzeros residing in $\mathbf{A}_p$.

For each i, the volume of communication required to fold the vector entry y_i is accurately represented as a part of "foldVolume" in the algorithm. For each j, the volume of communication regarding the vector entry x_j

JAGGED-LIKE-PARTITIONING ($\mathbf{A}$, $K = P \times Q$, ε_1, ε_2)
Input : a matrix $\mathbf{A}$, the number of processors $K = P \times Q$, and the imbalance ratios ε_1, ε_2.
Output: $map(a_{ij})$ for all $a_{ij} \neq 0$ and total Volume.

1: $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}}) \leftarrow$ columnNet($\mathbf{A}$)
2: $\Pi_{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_P\} \leftarrow$ partition($\mathcal{H}_{\mathcal{R}}$, P, ε_1) ▷ rowwise partitioning of $\mathbf{A}$
3: expand Volume $\leftarrow$ *cutsize*($\Pi_{\mathcal{R}}$)
4: foldVolume $\leftarrow 0$
5: **for** $p = 1$ **to** P **do**
6: $R_p = \{r_i : v_i \in \mathcal{R}_p\}$
7: $\mathbf{A}_p \leftarrow \mathbf{A}(R_p, :)$ ▷ submatrix indexed by rows R_p
8: $\mathcal{H}_p = (\mathcal{V}_p, \mathcal{N}_p) \leftarrow$ rowNet($\mathbf{A}_p$)
9: $\Pi^{\mathcal{C}}_p = \{\mathcal{C}^1_p, \ldots, \mathcal{C}^Q_p\} \leftarrow$ partition($\mathcal{H}_p$, Q, ε_2) ▷ columnwise partitioning of $\mathbf{A}_p$
10: foldVolume $\leftarrow$ foldVolume + *cutsize*($\Pi^{\mathcal{C}}_p$)
11: **for all** $a_{ij} \neq 0$ of $\mathbf{A}_p$ **do**
12: $map(a_{ij}) = P_{p,q} \Leftrightarrow c_j \in \mathcal{C}^q_p$
13: **return** totalVolume$\leftarrow$expandVolume+foldVolume

Hypergraph Partitioning. Fig. 1 Jagged-like partitioning



Hypergraph Partitioning. Fig. 2 First step of four-way jagged-like partitioning of a matrix; (**a**) two-way partitioning $\Pi_{\mathcal{R}}$ of column-net hypergraph representation $\mathcal{H}_{\mathcal{R}}$ of **A**, (**b**) two-way rowwise partitioning of matrix $\mathbf{A}^{\Pi}$ obtained by permuting **A** according to the partitioning induced by Π; the nonzeros in the same partition are shown with the same shape and color; the deviation of the minimum and maximum numbers of nonzeros of a part from the average are displayed as an interval `imbal`; `vol` denotes the number of nonzeros and the total communication volume

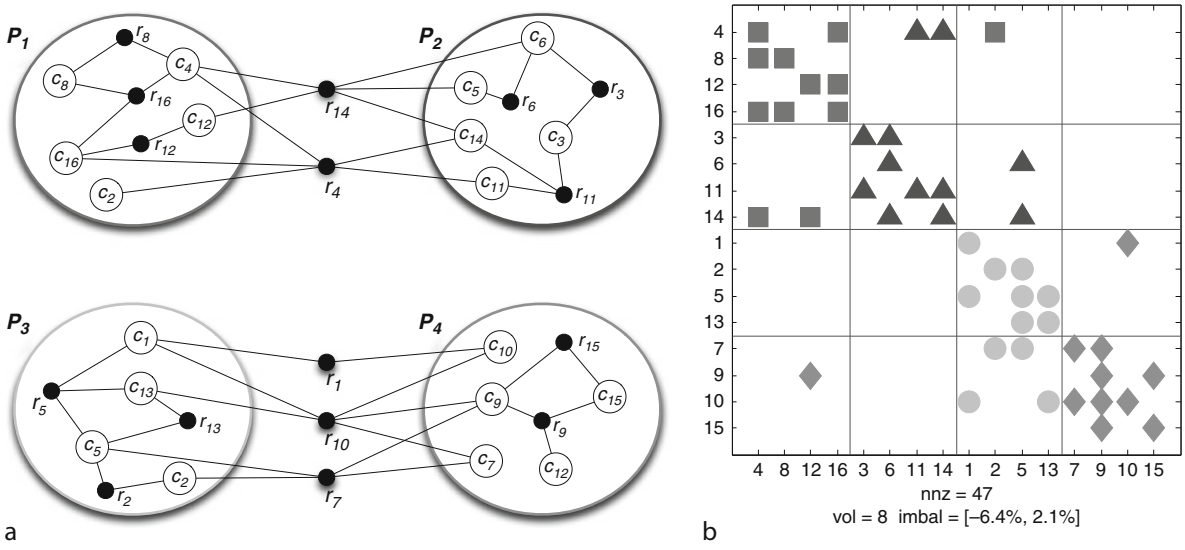
is accurately represented as a part of "expandVolume" in the algorithm.

Figure 2a illustrates the column-net representation of a sample matrix to be partitioned among the processors of a 2×2 mesh. For simplicity of the presentation, the vertices and the nets of the hypergraphs are labeled with letters "r" and "c" to denote the rows and columns of the matrix. The matrix is first partitioned rowwise into two parts, and each part is assigned to a row of the processor mesh, namely to processors $\{P_1, P_2\}$ and $\{P_3, P_4\}$. The resulting permuted matrix is displayed in Fig. 2b. Figure 3a displays the two row-net hypergraphs corresponding to each submatrix $\mathbf{A}_p$ for $p = 1, 2$. Each hypergraph is partitioned independently; sample partitions of these hypergraphs are also presented in this figure. As seen in the final symmetric permutation in Fig. 3b, the nonzeros of columns 2 and 5 are assigned to different parts, resulting P_3 to communicate with both P_1 and P_2 in the expand phase.

The *checkerboard partitioning method* [14] is also a two-step method, in which each step models either the expand phase or the fold phase of the parallel SpMxV. Similar to jagged-like partitioning, there are two alternative schemes for this partitioning method. The one which models the expands in the first step and the folds in the second step is presented below.

Given an $M \times N$ matrix **A** and the number K of processors organized as a $P \times Q$ mesh, the checkerboard partitioning method proceeds as shown in Fig. 4. First, **A** is partitioned rowwise into P parts using the column-net model (lines 1 and 2 of Fig. 4), producing $\Pi_{\mathcal{R}} = \{\mathcal{R}_1, \ldots, \mathcal{R}_P\}$. Note that this first step is exactly the same as that of the jagged-like partitioning. In the second step, the matrix **A** is partitioned columnwise into Q parts by using the multi-constraint partitioning to obtain $\Pi_{\mathcal{C}} = \{\mathcal{C}_1, \ldots, \mathcal{C}_Q\}$. In comparison to the jagged-like method, in this second step the whole matrix **A** is partitioned (lines 4 and 8 of Fig. 4), not the submatrices defined by $\Pi_{\mathcal{R}}$. The rowwise and columnwise partitions $\Pi_{\mathcal{R}}$ and $\Pi_{\mathcal{C}}$ together define a 2D partition on the matrix **A**, where $map(a_{ij}) = P_{p,q} \Leftrightarrow r_i \in \mathcal{R}_p$ and $c_j \in \mathcal{C}_q$.

In order to achieve a load balance among processors, a multi-constraint partitioning formulation is used (line 8 of the algorithm). Each vertex v_i of $\mathcal{H}_{\mathcal{C}}$ is assigned P weights: $w[i,p]$, for $p = 1, \ldots, P$. Here, $w[i,p]$ is equal to the number of nonzeros of column c_i in rows $\mathcal{R}_p$ (line 7 of Fig. 4). Consider a Q-way partitioning of $\mathcal{H}_{\mathcal{C}}$ with P constraints using the vertex weight definition



Hypergraph Partitioning. Fig. 3 Second step of four-way jagged-like partitioning: (**a**) Row-net representations of submatrices of **A** and two-way partitionings, (**b**) Final permuted matrix; the nonzeros in the same partition are shown with the same shape and color; the deviation of the minimum and maximum numbers of nonzeros of a part from the average are displayed as an interval `imbal`; `nnz` and `vol` denote, respectively, the number of nonzeros and the total communication volume

---

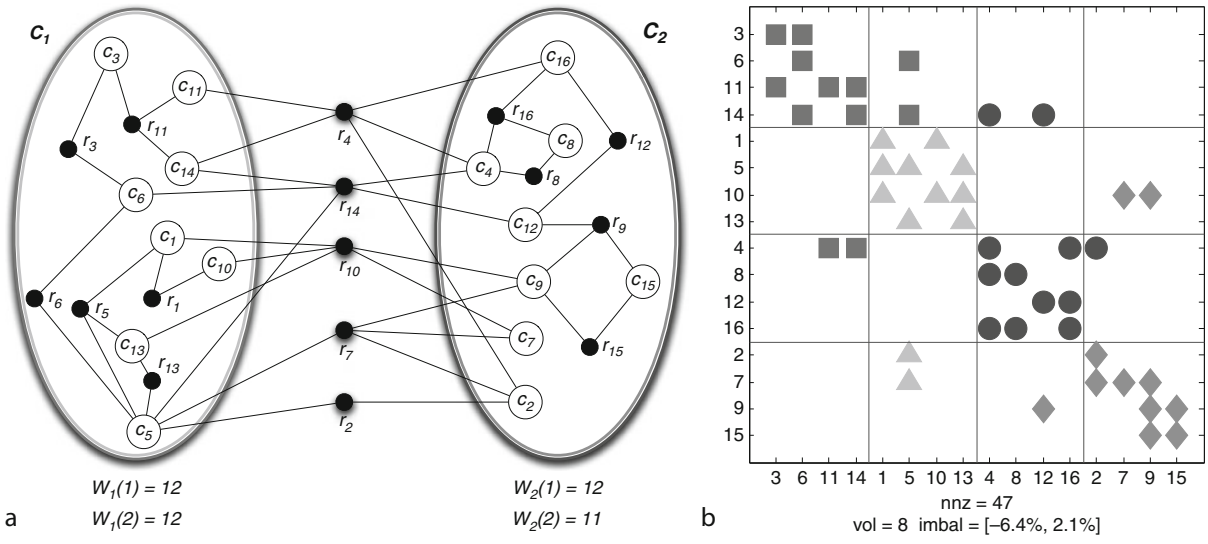
CHECKERBOARD-PARTITIONING(**A**, $K = P \times Q$, ε_1, ε_2)
Input: a matrix **A**, the number of processors $K = P \times Q$, and the imbalance ratios ε_1, ε_2.
Output: $map(a_{ij})$ for all $a_{ij} \neq 0$ and totalVolume.

1: $\mathcal{H}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{N}_{\mathcal{C}}) \leftarrow$ columnNet(**A**)
2: $\Pi_{\mathcal{R}} = \{\mathcal{R}_1,...,\mathcal{R}_P\} \leftarrow$ partition($\mathcal{H}_{\mathcal{R}}$, P, ε_1) ▷ rowwise partitioning of **A**
3: expand Volume $\leftarrow$ *cutsize*($\Pi_{\mathcal{R}}$)
4: $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}}) \leftarrow$ rowNet(**A**)
5: **for** $j = 1$ **to** $|\mathcal{V}_{\mathcal{C}}|$ **do**
6: **for** $p = 1$ **to** P **do**
7: $w_{j,p} = |\{n_j \cap \mathcal{R}_p\}|$
8: $\Pi_{\mathcal{C}} = \{\mathcal{C}_1,...,\mathcal{C}_Q\} \leftarrow$ MCPartition($\mathcal{H}_{\mathcal{C}}$, Q, ε_2) ▷ columnwise partitioning of **A**
9: foldVolume$\leftarrow$ $cutsize(\Pi_{\mathcal{C}})$
10: **for all** $a_{ij} \neq 0$ of **A** **do**
11: $map(a_{ij}) = P_{p,q} \Leftrightarrow r_i \in \mathcal{R}_p$ and $c_j \in \mathcal{C}_q$
12: totalVolume$\leftarrow$expandVolume+foldVolume

Hypergraph Partitioning. Fig. 4 Checkerboard partitioning

above. Maintaining the P balance constraints (4) corresponds to maintaining computational load balance on the processors of each row of the processor mesh.

Establishing the equivalence between the total communication volume and the sum of the cutsizes of the two partitions is fairly straightforward. The volume of communication for the fold operations corresponds exactly to the *cutsize*($\Pi_{\mathcal{C}}$). The volume of communication for the expand operations corresponds exactly to the *cutsize*($\Pi_{\mathcal{R}}$).



Hypergraph Partitioning. Fig. 5 Second step of four-way checkerboard partitioning: (**a**) two-way multi-constraint partitioning $\Pi_{\mathcal{C}}$ of row-net hypergraph representation $\mathcal{H}_{\mathcal{C}}$ of **A**, (**b**) Final checkerboard partitioning of **A** induced by $(\Pi_{\mathcal{R}}, \Pi_{\mathcal{C}})$; the nonzeros in the same partition are shown with the same shape and color; the deviation of the minimum and maximum numbers of nonzeros of a part from the average are displayed as an interval `imbal`; `nnz` and `vol` denote, respectively, the number of nonzeros and the total communication volume

Figure 5b displays the 2×2 checkerboard partition induced by $(\Pi_{\mathcal{R}}, \Pi_{\mathcal{C}})$. Here, $\Pi_{\mathcal{R}}$ is a rowwise two-way partition giving the same figure as shown in Fig. 2, and $\Pi_{\mathcal{C}}$ is a two-way multi-constraint partition $\Pi_{\mathcal{C}}$ of the row-net hypergraph model $\mathcal{H}_{\mathcal{C}}$ of **A** shown in Fig. 5a. In Fig. 5a, $w[9,1] = 0$ and $w[9,2] = 4$ for internal column c_9 of row stripe $\mathcal{R}_2$, whereas $w[5,1] = 2$ and $w[5,2] = 4$ for external column c_5.

Another common method of matrix partitioning is the *orthogonal recursive bisection* (ORB) [27]. In this approach, the matrix is first partitioned rowwise into two submatrices using the column-net hypergraph model, and then each part is further partitioned columnwise into two parts using the row-net hypergraph model. The process is continued recursively until the desired number of parts is obtained. The algorithm is shown in Fig. 6. In this algorithm, *dim* represents either rowwise or columnwise partitioning, where $-dim$ switches the partitioning dimension.

In the ORB method shown above, the step *bisect* $(\mathbf{A}, dim, \varepsilon)$ corresponds to partitioning the given matrix either along the rows or columns with, respectively, the column-net or the row-net hypergraph models into two. The total sum of the cutsizes (3) of each each bisection step corresponds to the total communication volume. It is possible to dynamically adjust the ε at each recursive call by allowing larger imbalance ratio for the recursive call on the submatrix $\mathbf{A}_1$ or $\mathbf{A}_2$.

Some Other Applications of Hypergraph Partitioning

As said before, the initial motivations for hypergraph models were accurate modeling of the nonzero structure of unsymmetric and rectangular sparse matrices to minimize the communication volume for iterative solvers. There are other applications that can make use of hypergraph partitioning formulation. Here, a brief overview of general classes of applications is given along with the names of some specific problems. Further application classes are given in bibliographic notes.

Parallel reduction or aggregation operations form a significant class of such applications, including the MapReduce model. The reduction operation consists of computing M output elements from N input elements. An output element may depend on multiple input elements, and an input element may contribute to multiple output elements. Assume that the operation on which

```
ORB-PARTITIONING(A, dim, K min, K max, ε)
Input: a matrix A, the part numbers K min (at initial call, it is equal to 1) and K max (at initial call it is
equal to K, the desired number of parts), and the imbalance ratio ε.
Output: map(a_ij) for all a_ij ≠ 0.
1:  if K max − K min > 0 then
2:      mid ← (K max − K min + 1)/2
3:      Π = ⟨A_1, A_2⟩←bisect(A, dim, ε)      ▷Partition A along dim into two, producing two submatrices
4:      totalVolume←totalVolume+cutsize(Π)
            ▷ Recursively partition each submatrix along the orthogonal direction
5:      map1(A_1)←ORB-PARTITIONING(A_1,−dim, Kmin, Kmin + mid−1, ε)
6:      map2(A_2)←ORB-PARTITIONING(A_2−dim, Kmin + mid, Kmax, ε)
7:      map(A)←map1(A_1) ∪ map2(A_2)
8:  else
9:      map(A)←Kmin
```

Hypergraph Partitioning. Fig. 6 Orthogonal recursive bisection (ORB)

reduction is performed is commutative and associative. Then, the inherent computational structure can be represented with an $M \times N$ dependency matrix, where each row and column of the matrix represents an output element and an input element, respectively. For an input element x_j and an output element y_i, if y_i depends on x_j, a_{ij} is set to 1 (otherwise zero). Using this representation, the problem of partitioning the workload for the reduction operation is equivalent to the problem of partitioning the dependency matrix for efficient SpMxV.

In some other reduction problems, the input and output elements may be preassigned to parts. The proposed hypergraph model can be accommodated to those problems by adding K *part* vertices and connecting those vertices to the nets which correspond to the preassigned input and output elements. Obviously, those part vertices must be fixed to the corresponding parts during the partitioning. Since the required property is already included in the existing hypergraph partitioners [6, 12, 19], this does not add extra complexity to the partitioning methods.

Iterative methods for solving linear systems usually employ *preconditioning* techniques. Roughly speaking, preconditioning techniques modify the given linear system to accelerate convergence. Applications of explicit preconditioners in the form of approximate inverses or factored approximate inverses are amenable to parallelization. Because, these techniques require SpMxV operations with the approximate inverse or factors of the approximate inverse at each step. In other words, preconditioned iterative methods perform SpMxV operations with both coefficient and preconditioner matrices in a step. Therefore, parallelizing a full step of these methods requires the coefficient and preconditioner matrices to be well partitioned, for example, processors' loads are balanced and communication costs are low in both multiply operations. To meet this requirement, the coefficient and preconditioner matrices should be partitioned simultaneously. One can accomplish such a simultaneous partitioning by building a single hypergraph and then partitioning that hypergraph. Roughly speaking, one follows a four-step approach: (i) build a hypergraph for each matrix, (ii) determine which vertices of the two hypergraphs need to be in the same part (according to the computations forming the iterative method), (iii) amalgamate those vertices coming from different hypergraphs, (iv) if the computations represented by the two hypergraphs of the first step are separated by synchronization points then assign multiple weights to vertices (the weights of the vertices of the hypergraphs of the first step are kept), otherwise assign a single weight to vertices (the weights of the vertices of the hypergraphs of the first step are summed up for each amalgamation).

The computational structure of the preconditioned iterative methods is similar to that of a more general class of scientific computations including multiphase, multiphysics, and multi-mesh simulations.

In *multiphase simulations*, there are a number of computational phases separated by global synchronization points. The existence of the global synchronizations

necessitates each phase to be load balanced individually. Multi-constraint formulation of hypergraph partitioning can be used to achieve this goal.

In *multi-physics simulations*, a variety of materials and processes are analyzed using different physics procedures. In these types of simulations, computational as well as the memory requirements are not uniform across the mesh. For scalability issues, processor loads should be balanced in terms of these two components. The multi-constraint partitioning framework also addresses these problems.

In *multi-mesh simulations*, a number of grids with different discretization schemes and with arbitrary overlaps are used. The existence of overlapping grid points necessitates a simultaneous partitioning of the grids. Such a simultaneous partitioning scheme should balance the computational loads of the processors and minimize the communication cost due to interactions within a grid as well as the interactions among different grids. With a particular transformation (the vertex amalgamation operation, also mentioned above), hypergraphs can be used to model the interactions between different grids. With the use of multi-constraint formulation, the partitioning problem in the multi-mesh computations can also be formulated as a hypergraph partitioning problem.

In obtaining partitions for two or more computation phases interleaved with synchronization points, the hypergraph models lead to the minimization of the overall sum of the total volume of communication in all phases (assuming that a single hypergraph is built as suggested in the previous paragraphs). In some sophisticated simulations, the magnitude of the interactions in one phase may be different than that of the interactions in another one. In such settings, minimizing the total volume of communication in each phase separately may be advantageous. This problem can be formulated as a multi-objective hypergraph partitioning problem on the so-built hypergraphs.

There are certain limitations in applying hypergraph partitioning to the multiphase, multiphysics, and multi-mesh-like computations. The dependencies must remain the same throughout the computations, otherwise the cutsize may not represent the communication volume requirements as precisely as before. The weights assigned to the vertices, for load balancing issues, should be static and available prior to the partitioning; the hypergraph models cannot be used as naturally for applications whose computational requirements vary drastically in time. If, however, the computational requirements change gradually in time, then the models can be used to re-partition the load at certain time intervals (while also minimizing the redistribution or migration costs associated with the new partition).

Ordering methods are quite common techniques to permute matrices in special forms in order to reduce the memory and running time requirements, as well as to achieve increased parallelism in direct methods (such as *LU* and Cholesky decompositions) used for solving systems of linear equations. Nested-dissection is a well-known ordering method that has been used quite efficiently and successfully. In the current state-of-the-art variations of the nested-dissection approach, a matrix is symmetrically permuted with a permutation matrix **P** into doubly bordered block diagonal form

$$\mathbf{A}_{DB} = \mathbf{PAP}^T \begin{bmatrix} A_{11} & & & & A_{1S} \\ & A_{22} & & & A_{2S} \\ & & \ddots & & \vdots \\ & & & A_{KK} & A_{KS} \\ A_{S1} & A_{S2} & \cdots & A_{SK} & A_{SS} \end{bmatrix},$$

where the nonzeros are only in the marked blocks (the blocks on the diagonal and the row and column borders). The aim in such a permutation is to have reduced numbers of rows/columns in the borders and to have equal-sized square blocks in the diagonal. One way to achieve such a permutation when **A** has symmetric pattern is as follows. Suppose a matrix **B** is given (if not, it is possible to find one) where the sparsity pattern of $\mathbf{B}^T\mathbf{B}$ equals to that of **A** (here arithmetic cancellations are ignored). Then, one can permute **B** nonsymmetrically into the singly bordered form

$$\mathbf{B}_{SB} = \mathbf{QBP}^T \begin{bmatrix} B_{11} & & & B_{1S} \\ & B_{22} & & B_{2S} \\ & & \ddots & \vdots \\ & & B_{KK} & B_{KS} \end{bmatrix},$$

so that $\mathbf{B}_{SB}^{T}\mathbf{B}_{SB} = \mathbf{PAP}^{T}$; that is, one can use the column permutation of $\mathbf{B}$ resulting in $\mathbf{B}_{SB}$ to obtain a symmetric permutation for $\mathbf{A}$ which results in $\mathbf{A}_{DB}$. Clearly, the column dimension of B_{kk} will be the size of the square matrix A_{kk} and the number of rows and columns in the border will be equal to the number of columns in the column border of $\mathbf{B}_{SB}$. One can achieve such a permutation of $\mathbf{B}$ by partitioning the column-net model of $\mathbf{B}$ while reducing the cutsize according to the cut-net metric (2), with unit net costs, to obtain the permutation $\mathbf{P}$ as follows. First, the permutation $\mathbf{Q}$ is defined to be able to define $\mathbf{P}$. Permute all rows corresponding to the vertices in part k before those in a part ℓ, for $1 \leq k < \ell \leq K$. Then, permute all columns corresponding to the nets that are internal to a part k before those that are internal to a part ℓ, for $1 \leq k < \ell \leq K$, yielding the diagonal blocks, and then permute all columns corresponding to the cut nets to the end, yielding the column border (the order of column defining a diagonal block). Clearly the correspondence between the size of the column border of $\mathbf{B}_{SB}$ and the doubly border of $\mathbf{A}_{DB}$ is exact, and hence the cutsize according to the cut-net metric is an exact measure. The requirement to have almost equal sized square blocks A_{kk} decoded as the requirement that each part should have an almost equal number of internal nets in the partition of the column-net model of $\mathbf{B}$. Although such a requirement is neither the objective nor the constraint of the hypergraph partitioning problem, the common hypergraph-partitioning heuristics easily accommodate such requirements.

Related Entries

►Data Distribution
►Graph Algorithms
►Graph Partitioning
►Linear Algebra, Numerical
►PaToH (Partitioning Tool for Hypergraphs)
►Preconditioners for Sparse Iterative Methods
►Sparse Direct Methods

Bibliographic Notes and Further Reading

The first use of the hypergraph partitioning methods for efficient parallel sparse matrix-vector multiply operations is seen in [10]. A more comprehensive study [11] describes the use of the row-net and column-net hypergraph models in 1D sparse matrix partitioning. For different views and alternatives on vector partitioning see [5, 22, 24].

A fair treatment of parallel sparse matrix-vector multiplication, analysis, and investigations on certain matrix types along with the use of hypergraph partitioning is given in [4, Chapter 4]. Further analysis of hypergraph partitioning on some model problems is given in [25].

Hypergraph partitioning schemes for preconditioned iterative methods are given in [23], where vertex amalgamation and multi-constraint weighting to represent different phases of computations are given. Discussions on application of such methodology for multiphase, multiphysics, and multi-mesh simulations are also discussed in the same paper.

Some different methods for sparse matrix partitioning using hypergraphs can be found in [26], including jagged-like and checkerboard partitioning methods, and in [27], the orthogonal recursive bisection approach. A recipe to choose a partitioning method for a given matrix is given in [16].

The use of hypergraph models for permuting matrices into special forms such as singly bordered block-diagonal form can be found in [3]. This permutation can be leveraged to develop hypergraph partitioning-based symmetric [9, 15] and nonsymmetric [17] nested-dissection orderings.

The standard hypergraph partitioning and the hypergraph partitioning with fixed vertices formulation, respectively, is used for static and dynamic load balancing for some scientific applications in [7, 8].

Some other applications of hypergraph partitioning are briefly summarized in [2]. These include image-space parallel direct volume rendering, parallel mixed integer linear programming, data declustering for multi-disk databases, scheduling file-sharing tasks in heterogeneous master-slave computing environments, and work-stealing scheduling, road network clustering methods for efficient query processing, pattern-based data clustering, reducing software development and maintenance costs, processing spatial join operations, and improving locality in memory or cache performance.

Bibliography

1. Ababei C, Selvakkumaran N, Bazargan K, Karypis G (2002) Multi-objective circuit partitioning for cutsize and path-based delay minimization. In: Proceedings of ICCAD 2002, San Jose, CA, November 2002
2. Aykanat C, Cambazoglu BB, Uçar B (2008) Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices. J Parallel Distr Comput 68(5): 609–625
3. Aykanat C, Pınar A, Çatalyürek UV (2004) Permuting sparse rectangular matrices into block-diagonal form. SIAM J Sci Comput 26(6):1860–1879
4. Bisseling RH (2004) Parallel scientific computation: a structured approach using BSP and MPI. Oxford University Press, Oxford, UK
5. Bisseling RH, Meesen W (2005) Communication balancing in parallel sparse matrix-vector multiplication. Electron Trans Numer Anal 21:47–65
6. Boman E, Devine K, Heaphy R, Hendrickson B, Leung V, Riesen LA, Vaughan C, Catalyurek U, Bozdag D, Mitchell W, Teresco J (2007) Zoltan 3.0: parallel partitioning, load balancing, and data-management services; user's guide. Sandia National Laboratories, Albuquerque, NM, 2007. Technical Report SAND2007-4748W http://www.cs.sandia.gov/Zoltan/ug_html/ug.html
7. Cambazoglu BB, Aykanat C (2007) Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids. IEEE Trans Parallel Distr Syst 18(1):3–16
8. Catalyurek U, Boman E, Devine K, Bozdag D, Heaphy R, Riesen L (2009) A repartitioning hypergraph model for dynamic load balancing. J Parallel Distr Comput 69(8):711–724
9. Çatalyürek UV (1999) Hypergraph models for sparse matrix partitioning and reordering. Ph.D. thesis, Computer Engineering and Information Science, Bilkent University. Available at http://www.cs.bilkent.edu.tr/tech-reports/1999/ABSTRACTS.1999.html
10. Çatalyürek UV, Aykanat C (1995) A hypergraph model for mapping repeated sparse matrixvector product computations onto multicomputers. In: Proceedings of International Conference on High Performance Computing (HiPC'95), Goa, India, December 1995
11. Çatalyürek UV, Aykanat C (1999) Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans Parallel Distr Syst 10(7):673–693
12. Çatalyürek UV, Aykanat C (1999) PaToH: a multilevel hypergraph partitioning tool, version 3.0. Department of Computer Engineering, Bilkent University, Ankara, 06533 Turkey. PaToH is available at http://bmi.osu.edu/~umit/software.htm
13. Çatalyürek UV, Aykanat C (2001) 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
14. Çatalyürek UV, Aykanat C (2001) A hypergraph-partitioning approach for coarse-grain decomposition. In: ACM/IEEE SC2001, Denver, CO, November 2001
15. Çatalyürek UV, Aykanat C, Kayaaslan E (2009) Hypergraph partitioning-based fill-reducing ordering. Technical Report OSUBMI-TR-2009-n02 and BU-CE-0904, Department of Biomedical Informatics, The Ohio State University and Computer Engineering Department, Bilkent University (Submitted)
16. Çatalyürek UV, Aykanat C, Uçar B (2010) On two-dimensional sparse matrix partitioning: models, methods, and a recipe. SIAM J Sci Comput 32(2):656–683
17. Grigori L, Boman E, Donfack S, Davis T (2008) Hypergraph unsymmetric nested dissection ordering for sparse LU factorization. Technical Report 2008-1290J, Sandia National Labs, Submitted to SIAM J Sci Comp
18. Karypis G, Kumar V (1998) Multilevel algorithms for multi-constraint hypergraph partitioning. Technical Report 99-034, Department of Computer Science, University of Minnesota/Army HPC Research Center, Minneapolis, MN 55455
19. Karypis G, Kumar V, Aggarwal R, Shekhar S (1998) hMeTiS a hypergraph partitioning package, version 1.0.1. Department of Computer Science, University of Minnesota/Army HPC Research Center, Minneapolis
20. Lengauer T (1990) Combinatorial algorithms for integrated circuit layout. Wiley–Teubner, Chichester
21. Selvakkumaran N, Karypis G (2003) Multi-objective hypergraph partitioning algorithms for cut and maximum subdomain degree minimization. In: Proceedings of ICCAD 2003, San Jose, CA, November 2003
22. Uçar B, Aykanat C (2004) Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. SIAM J Sci Comput 25(6):1837–1859
23. Uçar B, Aykanat C (2007) Partitioning sparse matrices for parallel preconditioned iterative methods. SIAM J Sci Comput 29(4):1683–1709
24. Uçar B, Aykanat C (2007) Revisiting hypergraph models for sparse matrix partitioning. SIAM Review 49(4):595–603
25. Uçar B, Çatalyürek UV (2010) On the scalability of hypergraph models for sparse matrix partitioning. In: Danelutto M, Bourgeois J, Gross T (eds), Proceedings of the 18th Euromicro Conference on Parallel, Distributed, and Network-based Processing, IEEE Computer Society, Conference Publishing Services, pp 593–600
26. Uçar B, Çatalyürek UV, Aykanat C (2010) A matrix partitioning interface to PaToH in MATLAB. Parallel Computing 36(5–6):254–272
27. Vastenhouw B, Bisseling RH (2005) A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. SIAM Review 47(1):67–95

Hyperplane Partitioning

►Tiling