HYPERGRAPH-BASED DATA PARTITIONING

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Enver Kayaaslan
September, 2013

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Prof. Dr. Cevdet Aykanat (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Hakan Ferhatosmanoğlu

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Hande Yaman

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Uğur Güdükbay

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Assoc. Prof. Dr. Murat Manguoğlu

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

HYPERGRAPH-BASED DATA PARTITIONING

Enver Kayaaslan
Ph.D. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
September, 2013

A hypergraph is a general version of graph where the edges may connect any number of vertices. By this flexibility, hypergraphs has a larger modeling power that may allow accurate formulaion of many problems of combinatorial scientific computing. This thesis discusses the use of hypergraph-based approaches to solve problems that require data partitioning. The thesis is composed of three parts. In the first part, we show how to implement hypergraph partitioning efficiently using recursive graph bipartitioning. The remaining two parts show how to formulate two important data partitioning problems in parallel computing as hypergraph partitioning. The first problem is global inverted index partitioning for parallel query processing and the second one is row-columnwise sparse matrix partitioning for parallel matrix vector multiplication, where both multiplication and sparse matrix partitioning schemes has novelty. In this thesis, we show that hypergraph models achieve partitions with better quality.

Keywords: hypergraph, data partitioning, combinatorial algorithms.

ÖZET

HİPERÇİZGE TABANLI VERİ BÖLÜMLEME

Enver Kayaaslan
Bilgisayar Mühendisliği, Doktora
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Eylül, 2013

Hiperçizgeler, bir kenarın herhangi bir sayıda düğümü bağlayabilme özelliği oldugu, çizgelerin genelleştirilmiş bir versiyonudur. Bu genelleme ile hiperçizgeler yüksek bir modelleme gücüne sahiptir öyle ki kombinatöriyel bilimsel hesaplama alanında birçok önemli problem hiperçizgeler ile güçlü bir şekilde modellenebilmektedir. Bu tez ise hiperçizge tabanlı yöntemler kullanılarak veri bölümleme problemlerinin çözülmesini araştırmaktadır. Bu tez üç ana bölümden oluşmaktadır. Birinci bölümde, özyinelemeli çizge ikiye-bölümleme kullanarak, verimli bir hiperçizge bölümleme aracının nasıl oluşturulduğu gösterilmektedir. İkinci ve üçüncü bölümlerde, paralel hesaplamadaki iki önemli veri bölümleme probleminin hiperçizge bölümleme ile nasıl modellendiği gösterilmektedir. Birinci problem paralel sorgu hesaplama için indeksin terim-tabanlı bölümlenmesi problemidir. İkincisi ise yeni önerilen bir paralel matriks vektör carpiminda kullanılmak üzere yine yeni önerilen bir seyrek matriks bölümleme problemidir. Bu tezde, hiperçizge tabanlı modelleri ile daha kaliteli veri bölümleme elde edildiği gösterilmektedir.

Anahtar sözcükler: hiperçizge, veri bölümleme, kombinatöriyel algoritmalar.

to my family ...

Acknowledgement

I would like to express my highest gratitude to my advisor Cevdet Aykanat for his guidance, suggestions, and encouragement to my research. I am thankful to Hakan Ferhatosmanoğlu and Hande Yaman for their help and guidance on the progress of this thesis study. I also thank to my juri members Uğur Güdükbay and Murat Manguoğlu for their valuable comments and suggestions.

I am grateful to my friends and my relatives for their infinite moral support. I still owe special thanks to Abdullah Bülbül and Erkan Okuyan. I am very grateful to Barla Cambazoğlu and Bora Uçar for their very kind attitudes that encouraged me well in both personal and academical life.

I thank to Ali Pınar and Ümit Çatalyürek for their intellectual contribution in Chapter 3. I thank to B. Barla. Cambazoğlu for drawing Figure 5.5 and his contributions for the improvement of the textual material of Chapter 4. Finally, I would like to thank to Bora Uçar for his intellectual contributions in Chapter 5.

I would thank to Scientific and Technological Research Council (TÜBİTAK) for supporting my PhD program.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A hypergraph is a generalization of a graph, since it replaces edges that connect only two vertices, with hyperedges (nets) that can connect multiple vertices. This generalization provides a critical modeling flexibility that allows accurate formulation of many important problems in combinatorial scientific computing. After their introduction in [1, 2], the modeling power of hypergraphs appealed to many researchers and they were applied to a wide variety of many applications in scientific computing [3–23]. Hypergraphs and hypergraph partitioning are now standard tools of combinatorial scientific computing. Increasing popularity of hypergraphs has been accompanied with the development of effective hypergraph partitioning (HP) tools: wide applicability of hypergraphs motivated development of fast HP tools, and availability of effective HP tools motivated further applications. This virtuous cycle produced sequential HP tools such as hMeTiS [24], PaToH [25] and Mondriaan [21], and parallel HP tools such as Parkway [26] and Zoltan [27], all of which adopt the multilevel framework successfully.

While the hypergraph partitioning tools provide good performances both in terms of solution quality and processing times, they are hindered by the inherent complexity of dealing with hypergraphs. Algorithms on hypergraphs are more difficult both in terms of computational complexity and runtime performance, since operations on nets are performed on sets of vertices as opposed to pairs of vertices as in graphs. The wide interest over the last decade has proven the

modeling flexibility of hypergraphs to be essential, but the runtime efficiency of graph algorithms cannot be overlooked, either. Therefore, we believe that the new research thrust should be how to cleverly trade-off between the modeling flexibility of hypergraphs and the practicality of graphs.

In Chapter 3, we investigate solving the HP problem by finding vertex separators on the *net intersection graph* (NIG) of the hypergraph. In the NIG of a hypergraph, each net is represented by a vertex and each vertex of the hypergraph is replaced with a clique of the nets connecting that vertex. A vertex separator on this graph defines a net separator for the hypergraph. This model has been initially studied for circuit partitioning [34]. While faster algorithms can be designed to find vertex separators on graphs, the NIG model has the drawback of attaining unbalanced partitions. Once vertices of the hypergraphs are replaced with cliques, it will be impossible to preserve the vertex weight information accurately. Therefore, we can view the NIG model as a way to trade off computational efficiency with exact modeling power.

As we will show in the experiments, the NIG model can effectively be employed for these applications to achieve high quality solutions in a shorter time. We show that it is easy to enforce a balance criterion on the internal nets of hypergraph partitioning by enforcing vertex balancing during the partitioning of the NIG. However, the NIG model cannot completely preserve the vertex balancing information of the hypergraph. We propose a weighting scheme in NIG, which is quite effective in attaining fairly vertex-balanced partitions of the hypergraph. The proposed vertex balancing scheme for the NIG partitioning can be easily enhanced to improve the balancing quality of the hypergraph partitions in a simple post-processing phase. The recursive bipartitioning (RB) paradigm is widely used for multiway HP and known to produce good solution qualities [24, 25]. At each RB step, cutnet removal and cutnet splitting techniques [6] are adopted to optimize the cutsize according to the *cutnet* and *connectivity* metrics, respectively, which are the most commonly used cutsize metrics in scientific and parallel computing [6, 35] as well as VLSI layout design [33, 34]. In this work, we propose separator-vertex removal and separator-vertex splitting techniques for RB-based partitioning of the NIG, which exactly correspond to the cutnet removal and

cutnet splitting techniques, respectively. We also propose an implementation for our GPVS-based HP formulations by adopting and modifying a state-of-the-art GPVS tool used in fill-reducing sparse matrix ordering.

In Chapters 4 and 5, we respectively show how to model a data partitioning problem as a hypergraph partitioning problems, on parallel query processing and parallel sparse matrix vector multiplication. The large-scale search engines has to process queries in a reasonable amount of time. Parallelism is the remedy of this requirement. To process queries efficiently, an inverted index on the document collection is built [47], where an inverted index contains a list of document ids for each term in the vocabulary. For each term-document pair, some other auxiliary information, such as the frequency of the term in the document, can be held. There are two common approaches for parallel query processing: doc-parallel and term-parallel. Term-parallel query processing has an advantage in number of disk accesses. The quest is to distribute the terms to processors such that query processing load is evenly shared and the total inter-processor communication is low in a batch-mode processing scenario. We formulate the term partitioning problem with a hypergraph partitioning problem where the vertices are terms and the nets are queries.

Chapter 5 investigates sparse matrix vector multiplication (SpMxV), which is a kernel operation repeatedly performed in iterative linear system solvers. There are mainly three types of parallel SpMxV algorithms used in the scientific community: row-parallel, column-parallel and row-column-parallel. The row-parallel algorithm involves expand-type point-to-point communication operations on the local input vector entries before the local SpMxV operations, whereas column-parallel algorithm involves fold-type point-to-point communication operations on the local output vector results after the local SpMxV operations. The row-column-parallel algorithm necesitates two-phase communication: expand operation before local SpMxVs and fold operation after the local SpMxVs. 1D rowwise and columnwise partitioning of the coefficiant matrix are used for row-parallel and column-parallel SpMxV algorithms, respectively, whereas 2D-nonzero partitioning of the coefficiant matrix is used for row-column-parallel SpMxV algorithms. Several hypergraph partitioning models and methods have been successfully used

for sparse matrix partitioning for efficient row-parallel, column-parallel and row-column-parallel SpMxV operations. In all these models the partitioning objective is to minimize the total volume of communication whereas the partitioning constraint is to minimize the computational load balance. 2D nonzero based partitioning models are both more scalable and perform considerably better than the 1D partitioning models in terms of communication volume metric. However, 1D models perform considerably better than 2D models in terms of speedup values due to the increased number of messages in the row-column-parallel SpMxV algorithm.

In Chapter 5, we propose a single-phase row-column-parallel SpMxV algorithm to address this bottleneck of the row-column-parallel SpMxV operation. This new parallel multiplication scheme introduced row-columnwise partitioning of sparse matrices where a nonzero is assigned to either the receiver or the sender processor associated with the related input- or output-vector entries. We model this partitioning with hypergraph partitioning problem where cooccurrence relations are introduced, which in turn causes a restriction of the solution space but providing larger modeling flexibility. Unfortunately, there is currently no tool implementing this new version of hypergraph partitioning. Thus, we solved the row-columnwise partitioning problem resorting on the one-dimensional partitioning methods. After obtaining a rowwise partitioning, we relax the assignments the nonzeros of the off-diagonal blocks using Dulmage-Mendhelson decomposition on those blocks, separately. Using this decomposition, we obtain assignment of nonzeros that accurately minimizes the communication volume in this framework.

Chapter 2

Background

In this chapter, we give some combinatorial background that is required for the rest of the thesis. Specifically, we define graph and hypergraph partitioning problems, and give the definition of net intersection graph of a hypergraph.

2.1 Graph Partitioning

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. We use the notation $Adj(v_i)$ to denote the set of vertices adjacent to vertex v_i. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}' \subset \mathcal{V}$, i.e., $Adj(\mathcal{V}') = \{v_j \in \mathcal{V} - \mathcal{V}' : v_j \in Adj(v_i) \text{ for some } v_i \in \mathcal{V}'\}$. Two disjoint vertex subsets $\mathcal{V}_k$ and $\mathcal{V}_\ell$ are said to be adjacent if $Adj(\mathcal{V}_k) \cap \mathcal{V}_\ell \neq \emptyset$ (equivalently $Adj(\mathcal{V}_\ell) \cap \mathcal{V}_k \neq \emptyset$) and non-adjacent otherwise. The degree $d(v_i)$ of a vertex v_i is equal to the number of edges incident to v_i, i.e., $d(v_i) = |Adj(v_i)|$. A weight $w(v_i) \geq 0$ is associated with each vertex v_i.

An edge subset $\mathcal{E}_S$ is a K-way *edge separator* if its removal disconnects the graph into at least K connected components. That is, $\Pi_{ES}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way vertex partition of $\mathcal{G}$ by edge separator $\mathcal{E}_S \subset \mathcal{E}$ if each part $\mathcal{V}_k$ is

non-empty; parts are pairwise disjoint; and the union of parts gives $\mathcal{V}$. Edges between the vertices of different parts belong to $\mathcal{E}_S$, and are called *cut (external)* edges and all other edges are called *uncut (internal)* edges.

A vertex subset $\mathcal{V}_S$ is a K-way *vertex separator* if the subgraph induced by the vertices in $\mathcal{V}-\mathcal{V}_S$ has at least K connected components. That is, $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ is a K-way vertex partition of $\mathcal{G}$ by vertex separator $\mathcal{V}_S \subset \mathcal{V}$ if each part $\mathcal{V}_k$ is non-empty; all parts and the separator are pairwise disjoint; parts are pairwise non-adjacent; and the union of parts and the separator gives $\mathcal{V}$. The non-adjacency of the parts implies that $Adj(\mathcal{V}_k) \subseteq \mathcal{V}_S$ for each $\mathcal{V}_k$. The *connectivity* $\lambda(v_i)$ of a vertex v_i denotes the number of parts connected by v_i, where a vertex that is adjacent to any vertex in a part is said to *connect* that part. A vertex $v_i \in \mathcal{V}_k$ is said to be a boundary vertex of part $\mathcal{V}_k$ if it is adjacent to any vertex in $\mathcal{V}_S$. A vertex separator is said to be *narrow* if no subset of it forms a separator, and *wide*, otherwise.

The objective of graph partitioning is finding a separator of smallest size subject to a given balance criterion on the weights of the K parts. The weight $W(\mathcal{V}_k)$ of a part V_k is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, i.e.,

$$W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i) \tag{2.1}$$

and the balance criterion is defined as

$$\begin{aligned} \max_{1 \leq k \leq K} W(\mathcal{V}_k) &\leq (1+\epsilon) W_{avg} \ , \quad \text{where} \\ W_{avg} &= \frac{\sum_{k=1}^{K} W(\mathcal{V}_k)}{K}. \end{aligned} \tag{2.2}$$

Here, W_{avg} is the weight each part must have in the case of perfect balance, and ϵ is the maximum imbalance ratio allowed. We proceed with formal definitions for the GPES and GPVS problems, both of which are known to be NP-hard [31].

Definition 1 (Problem GPES) *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, an integer K, and a maximum allowable imbalance ratio ϵ, GPES problem is finding a K-way vertex partition $\Pi_{ES}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{G}$ by edge separator $\mathcal{E}_S$ that satisfies the*

balance criterion given in Equation 2.2 *while minimizing the cutsize, which is defined as*

$$cutsize(\Pi_{ES}) = \sum_{e_{ij} \in \mathcal{E}_S} c(e_{ij}), \tag{2.3}$$

where $c(e_{ij}) \geq 0$ *is the cost of edge* $e_{ij} = (v_i, v_j)$.

Definition 2 (Problem GPVS) *Given a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, an integer* K*, and a maximum allowable imbalance ratio* ϵ*, GPVS problem is finding a* K*-way vertex partition* $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ *of* $\mathcal{G}$ *by vertex separator* $\mathcal{V}_S$ *that satisfies the balance criterion given in* Equation 2.2 *while minimizing the cutsize, which is defined as one of*

$$a)\ cutsize(\Pi_{VS}) = \sum_{v_i \in \mathcal{V}_S} c(v_i) \tag{2.4}$$

$$b)\ cutsize(\Pi_{VS}) = \sum_{v_i \in \mathcal{V}_S} c(v_i)(\lambda(v_i) - 1) \tag{2.5}$$

where $c(v_i) \geq 0$ *is the cost of vertex* v_i.

In the cutsize definition given in Equation 2.4, each separator vertex incurs its cost to the cutsize, whereas in Equation 2.5, the connectivity of a vertex is considered while incurring its cost to the cutsize. In the general GPVS definition given above, both a weight and a cost are associated with each vertex. The weights are used in computing loads of parts for balancing, whereas the costs are utilized in computing the cutsize.

The techniques for solving GPES and GPVS problems are closely related. An *indirect* approach to solve the GPVS problem is to first find an edge separator through GPES, and then translate it to any vertex separator. After finding an edge separator, this approach takes vertices adjacent to separator edges as a wide separator to be refined to a narrow separator, with the assumption that a small edge separator is likely to yield a small vertex separator. The wide-to-narrow refinement problem [32] is described as a minimum vertex cover problem on the

bipartite graph induced by the cut edges. A minimum vertex cover can be taken as a narrow separator for the whole graph, because each cut edge will be adjacent to a vertex in the vertex cover.

2.2 Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ is defined as a set $\mathcal{U}$ of nodes (vertices) and a set $\mathcal{N}$ of nets among those vertices. We refer to the vertices of $\mathcal{H}$ as nodes, to avoid the confusion between graphs and hypergraphs. Every net $n_i \in \mathcal{N}$ connects a subset of nodes, i.e., $n_i \subseteq \mathcal{U}$. The nodes connected by a net n_i are called *pins* of n_i and denoted as $Pins(n_i)$. We extend this operator to include the pin list of a net subset $\mathcal{N}' \subset \mathcal{N}$, i.e., $Pins(\mathcal{N}') = \bigcup_{n_i \in \mathcal{N}'} Pins(n_i)$. The size $s(n_i)$ of a net n_i is equal to the number of its pins, i.e., $s(n_i) = |Pins(n_i)|$. The set of nets that connect a node u_j is denoted as $Nets(u_j)$. We also extend this operator to include the net list of a node subset $\mathcal{U}' \subset \mathcal{U}$, i.e., $Nets(\mathcal{U}') = \bigcup_{u_j \in \mathcal{U}'} Nets(u_j)$. The degree $d(u_j)$ of a node u_j is equal to the number of nets that connect u_j, i.e., $d(u_j) = |Nets(u_j)|$. The total number of pins, p, denotes the size of $\mathcal{H}$ where $p = \sum_{n_i \in \mathcal{N}} s(n_i) = \sum_{u_j \in \mathcal{U}} d(u_j)$. A graph is a special hypergraph such that each net has exactly two pins. A weight $w(u_j)$ is associated with each node u_j, whereas a cost $c(n_i)$ is associated with each net n_i. A weight $w(n_i)$ can also be associated with each net n_i as we will discuss later in this section.

A net subset $\mathcal{N}_S$ is a K-way *net separator* if its removal disconnects the hypergraph into at least K connected components. That is, $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ is a K-way node partition of $\mathcal{H}$ by net separator $\mathcal{N}_S \subset \mathcal{N}$ if each part $\mathcal{U}_k$ is non-empty; parts are pairwise disjoint; and the union of parts gives $\mathcal{U}$. In a partition $\Pi_{\mathcal{U}}(\mathcal{H})$, a net that connects any node in a part is said to *connect* that part. The *connectivity* $\lambda(n_i)$ of a net n_i denotes the number of parts connected by n_i. Nets connecting multiple parts belong to $\mathcal{N}_S$, and are called *cut (external)* (i.e., $\lambda(n_i) > 1$), and *uncut (internal)* otherwise (i.e., $\lambda(n_i) = 1$). The set of internal nets of a part $\mathcal{U}_k$ is denoted as $\mathcal{N}_k$, for $k = 1, \ldots, K$. So, although $\Pi_{\mathcal{U}}(\mathcal{H})$ is defined as a K-way partition on the node set of $\mathcal{H}$, it can also be considered as

inducing a $(K+1)$-way partition $\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$ on the net set.

As in the GPES and GPVS problems, the objective of the hypergraph partitioning (HP) problem is finding a net separator of smallest size subject to a given balance criterion on the weights of the K parts. The weight $W(\mathcal{U}_k)$ of a part $\mathcal{U}_k$ is defined either as the sum of the weights of nodes in $\mathcal{U}_k$, i.e.,

$$W(\mathcal{U}_k) = \sum_{u_j \in \mathcal{U}_k} w(u_j) \tag{2.6}$$

or as the sum of weights of internal nets of part $\mathcal{U}_k$, i.e.,

$$W(\mathcal{U}_k) = \sum_{n_i \in \mathcal{N}_k} w(n_i). \tag{2.7}$$

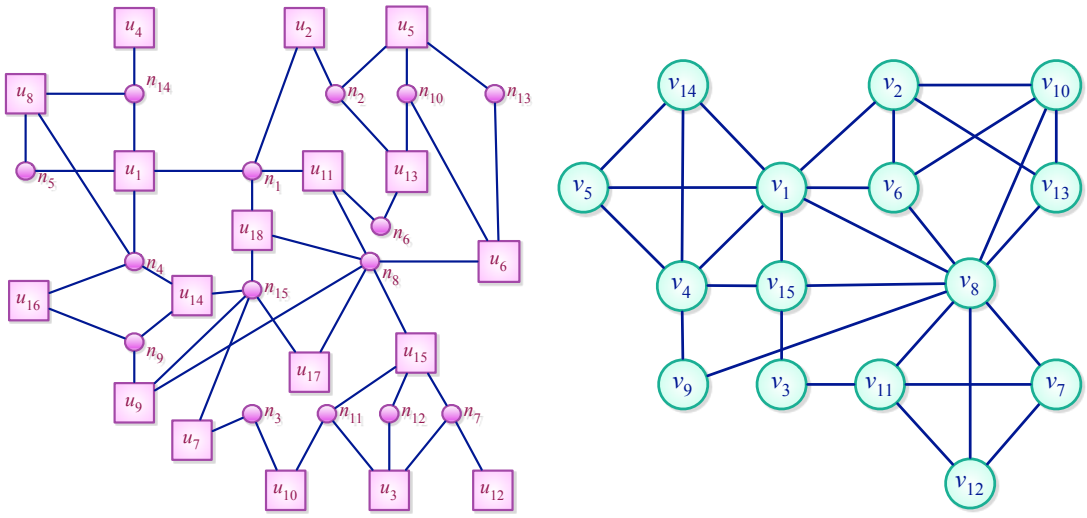
The former and latter part-weight computation schemes together with the load balancing criterion given in Equation 2.2 will be referred to here as node and net balancing, respectively. We proceed with a formal definition for the HP problem, which is also known to be NP-hard [33].

Definition 3 (Problem HP) *Given a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, an integer K, and a maximum allowable imbalance ratio ϵ, HP problem is finding a K-way node partition $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ that satisfies the balance criterion given in* Equation 2.2 *while minimizing the cutsize, which is defined as one of*

$$a)\ cutsize(\Pi_{\mathcal{U}}) = \sum_{n_i \in \mathcal{N}_S} c(n_i) \tag{2.8}$$

$$b)\ cutsize(\Pi_{\mathcal{U}}) = \sum_{n_i \in \mathcal{N}_S} c(n_i)(\lambda(n_i) - 1). \tag{2.9}$$

The cutsize metrics given in Equation 2.8 and Equation 2.9 are referred to as the *cut-net* and *connectivity* metrics, respectively, [6, 9, 33].



(a) (b)

Figure 2.1: (a) A sample hypergraph $\mathcal{H}$ and (b) the corresponding NIG representation $\mathcal{G}$.

2.3 Net Intersection Graph

In the NIG representation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of a given hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, each vertex v_i of $\mathcal{G}$ corresponds to net n_i of $\mathcal{H}$, and we will use notation $v_i \equiv n_i$ to represent this correspondence. Two vertices $v_i, v_j \in \mathcal{V}$ of $\mathcal{G}$ are adjacent if and only if respective nets $n_i, n_j \in \mathcal{N}$ of $\mathcal{H}$ share at least one pin, i.e., $e_{ij} \in \mathcal{E}$ if and only if $Pins(n_i) \cap Pins(n_j) \neq \emptyset$. So,

$$Adj(v_i) = \{v_j \equiv n_j \mid n_j \in \mathcal{N} \text{ and } Pins(n_i) \cap Pins(n_j) \neq \emptyset\}. \tag{2.10}$$

Note that for a given hypergraph $\mathcal{H}$, NIG $\mathcal{G}$ is well-defined, however there is no unique reverse construction [34]. Figures 2.1(a) and 2.1(b), respectively, display a sample hypergraph $\mathcal{H}$ and the corresponding NIG representation $\mathcal{G}$. In the figure, the sample hypergraph $\mathcal{H}$ contains 18 nodes and 15 nets, whereas the corresponding NIG $\mathcal{G}$ contains 15 vertices and 30 edges.

Chapter 3

Fast Hypergraph Partitioning based on Recursive Graph Bipartitioning

How can we solve problems that are most accurately modeled with hypergraphs using graph algorithms without sacrificing too much from what is really important for the application? This question has been asked before, and the motivation was either theoretical [28] or practical [29, 30] when the absence of HP tools behest these attempts. This earlier body of work investigated the relation between HP and graph partitioning by edge separator (GPES), and achieved little success. Today, we are facing a more difficult task, as effectiveness of available HP tools sets high standards for novel approaches. On the other hand, we can draw upon the progress on related problems, in particular the advances in tools for graph partitioning by vertex separator (GPVS). In this chapter, we present how the hypergraph partitioning problem can be implemented using recursive two-way GPVS efficiently and support our discussion with a detailed emprical study.

3.1 Background

In [39] the authors propose a net-partitioning based K-way HP algorithm that avoids the module contention problem (which we will also refer to as *contention-free*) by describing the HP problem as a GPVS problem through the NIG model. The following theorem lays down the basis for the proposed GPVS-based HP formulation. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote the NIG of a given hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$. The cost of each net n_i of $\mathcal{H}$ is assigned as the cost of the respective vertex v_i of $\mathcal{G}$, i.e., $c(v_i) = c(n_i)$. For brevity of the presentation we assume unit net costs here, but all proposed models and methods generalize to hypergraphs with non-unit net costs.

Theorem 1 [39] *A K-way vertex partition $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of $\mathcal{G}$ by a narrow vertex separator $\mathcal{V}_S$ induces a K-way contention-free net partition $\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1 \equiv \mathcal{V}_1, \mathcal{N}_2 \equiv \mathcal{V}_2, \ldots, \mathcal{N}_K \equiv \mathcal{V}_K; \mathcal{N}_S \equiv \mathcal{V}_S\}$ of $\mathcal{H}$ by a net separator $\mathcal{N}_S$.*

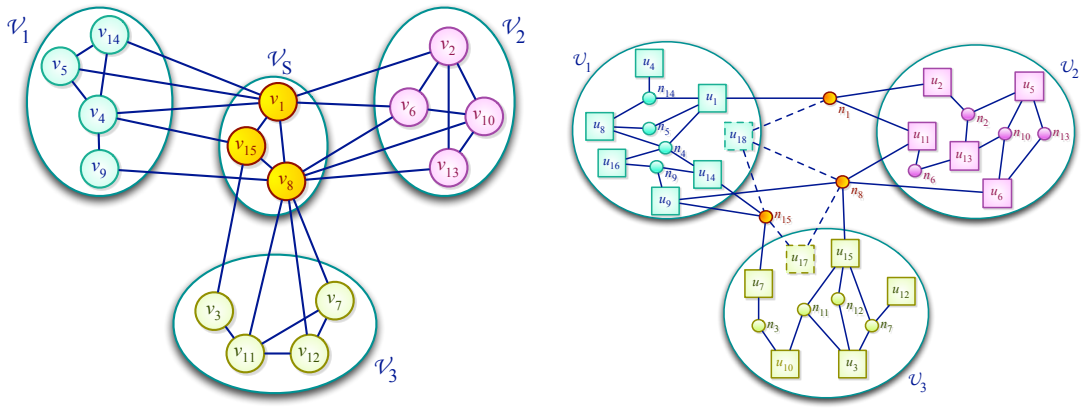
A K-way contention-free net partition of $\mathcal{H}$ by a net separator $\mathcal{N}_S$

$$\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1 \equiv \mathcal{V}_1, \ldots, \mathcal{N}_K \equiv \mathcal{V}_K; \mathcal{N}_S \equiv \mathcal{V}_S\} \tag{3.1}$$

induces a K-way partial node partition

$$\Pi'_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}'_1 = Pins(\mathcal{N}_1), \ldots, \mathcal{U}'_K = Pins(\mathcal{N}_K)\}. \tag{3.2}$$

Figure 3.1(a) shows a 3-way GPVS $\Pi_{VS}(\mathcal{G})$ of the sample NIG $\mathcal{G}$ given in Figure 2.1(b). Figure 3.1(b) shows the 3-way partial and complete node partition $\Pi'_{\mathcal{U}}(\mathcal{H})$ of the sample $\mathcal{H}$, which is induced by $\Pi_{VS}(\mathcal{G})$. Partial node partition is displayed with nodes drawn with solid lines, and complete node partition is achieved by adding 2 free nodes (drawn with dashed lines). The sample $\mathcal{H}$ given in Figure 2.1(a) contains only 2 free nodes, which are u_{17} and u_{18}. Comparison of Figures 3.1(a) and 3.1(b) illustrates that the separator vertices v_1, v_8 and v_{15} of $\Pi_{VS}(\mathcal{G})$ induce the cut nets n_1, n_8, and n_{15} of $\Pi'_{\mathcal{U}}(\mathcal{H})$, respectively.



(a) (b)

Figure 3.1: (a) A 3-way GPVS of the sample NIG given in Figure 2.1(b) and (b) corresponding partitioning of the hypergraph.

We can construct a complete node partition in the following form,

$$\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1 \supseteq \mathcal{U}'_1, \mathcal{U}_2 \supseteq \mathcal{U}'_2, \ldots, \mathcal{U}_K \supseteq \mathcal{U}'_K\}. \tag{3.3}$$

Note that any K-way node partition of $\mathcal{H}$ inducing the $(K+1)$-way net partition $\Pi_{\mathcal{N}}(\mathcal{H})$ has to be in the form above.

Theorem 2 [39] *Given a K-way vertex partition $\Pi_{VS}(\mathcal{G})$ of $\mathcal{G}$ by a narrow vertex separator $\mathcal{V}_S$, any node partition $\Pi_{\mathcal{U}}(\mathcal{H})$ of $\mathcal{H}$ as constructed according to Equation 3.3 induces the $(K+1)$-way net partition $\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1 \equiv \mathcal{V}_1, \ldots, \mathcal{N}_K \equiv \mathcal{V}_K; \mathcal{N}_S \equiv \mathcal{V}_S\}$ such that the connectivity of each cut net in $\mathcal{N}_S$ is greater than or equal to the connectivity of the corresponding separator vertex in $\mathcal{V}_S$.*

Corollary 1 [39] *Given a K-way vertex partition $\Pi_{VS}(\mathcal{G})$ of $\mathcal{G}$ by a narrow vertex separator $\mathcal{V}_S$, the separator size of $\Pi_{VS}(\mathcal{G})$ is equal to the cutsize of node partition $\Pi_{\mathcal{U}}(\mathcal{H})$ induced by $\Pi_{VS}(\mathcal{G})$ according to the cutnet metric, whereas the separator size of $\Pi_{VS}(\mathcal{G})$ approximates the cutsize of node partition $\Pi_{\mathcal{U}}(\mathcal{H})$ induced by $\Pi_{VS}(\mathcal{G})$ according to the connectivity metric.*

Comparison of Figures 3.1(a) and 3.1(b) illustrates that the connectivities of separator vertices in Π_{VS} are exactly equal to those of the cut nets of induced partial node partition $\Pi'_{\mathcal{U}}(\mathcal{H})$. Figure 3.1(b) shows a 3-way complete node partition $\Pi_{\mathcal{U}}(\mathcal{H})$ obtained by assigning the free nodes (shown with dashed lines) u_{17} and u_{18} to parts $\mathcal{U}_3$ and $\mathcal{U}_1$, respectively. This free node assignment does not increase the connectivities of the cut nets. However a different free node assignment might increase the connectivities of the cut nets. For example, assigning free node u_{17} to part $\mathcal{U}_2$ instead of $\mathcal{U}_3$ will increase the connectivity of net n_{15} by 1.

3.2 Recursive-bipartitioning-based Partitioning

In the recursive bipartitioning (RB) paradigm, a hypergraph is first partitioned into 2 parts. Then, each part of the bipartition is further bipartitioned recursively until the desired number of parts, K is achieved.

3.2.1 Separator-vertex Removal and Splitting

The following corollary forms the basis for the use of RB-based GPVS for RB-based HP according to the connectivity and the cut-net metrics.

Corollary 2 *Let $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$ be a partition of $\mathcal{G}$ by a vertex separator $\mathcal{V}_S$, and let $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2\}$ be a node partition of $\mathcal{H}$ that induces the net partition $\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1 \equiv \mathcal{V}_1, \mathcal{N}_2 \equiv \mathcal{V}_2; \mathcal{N}_S \equiv \mathcal{V}_S\}$. The connectivity of a net n_i in $\Pi_{\mathcal{U}}(\mathcal{H})$ is equal to the connectivity of the corresponding vertex v_i in $\Pi_{VS}(\mathcal{G})$.*

3.2.1.1 Separator-vertex Removal

In RB-based multiway HP, the cut-net metric is formulated by cut-net removal after each RB step. In this method, after each hypergraph bipartitioning step, each cut net is discarded from further RB steps. That is, a node bipartition

$\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2\}$ of the current hypergraph $\mathcal{H}$, which induces the net bipartition $\Pi_{\mathcal{N}}(\mathcal{H}) = \{\mathcal{N}_1, \mathcal{N}_2; \mathcal{N}_S\}$, is decoded as generating two sub-hypergraphs $\mathcal{H}_1 = (\mathcal{U}_1, \mathcal{N}_1)$ and $\mathcal{H}_2 = (\mathcal{U}_2, \mathcal{N}_2)$ for further RB steps. Hence, the total cutsize of the resulting multiway partition of $\mathcal{H}$ according to the cut-net metric will be equal to the sum of the number of cut nets of the bipartition obtained at each RB step.

The cut-net metric can be formulated in the RB-GPVS-based multiway HP by separator-vertex removal so that each separator vertex is discarded from further RB steps. That is, at each RB step, a 2-way vertex separator $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$ of $\mathcal{G}$ is decoded as generating two sub-graphs $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$, where $\mathcal{E}_1$ and $\mathcal{E}_2$ denote the internal edges of vertex parts $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. In other words, $\mathcal{G}_1$ and $\mathcal{G}_2$ are the sub-graphs of $\mathcal{G}$ induced by the vertex parts $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. $\mathcal{G}_1$ and $\mathcal{G}_2$ constructed in this way become the NIG representations of hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. Hence, the sum of the number of separator vertices of the 2-way GPVS obtained at each RB step will be equal to the total cutsize of the resulting multiway partition of $\mathcal{H}$ according to the cut-net metric.

3.2.1.2 Separator-vertex Splitting

In RB-based multiway HP, the connectivity metric is formulated by adapting the cut-net splitting method after each RB step. In this method, each RB step, $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2\}$ is decoded as generating two sub-hypergraphs $\mathcal{H}_1 = (\mathcal{U}_1, \mathcal{N}_1)$ and $\mathcal{H}_2 = (\mathcal{U}_2, \mathcal{N}_2)$ as in the cut-net removal method. Then, each cut net n_s of $\Pi_{\mathcal{U}}(\mathcal{H})$ is split into two pin-wise disjoint nets n_s^1 and n_s^2 with $Pins(n_s^1) = Pins(n_s) \cap \mathcal{U}_1$ and $Pins(n_s^2) = Pins(n_s) \cap \mathcal{U}_2$, where n_s^1 and n_s^2 are added to the net lists of $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. In this way, the total cutsize of the resulting multiway partition according to the connectivity metric will be equal to the sum of the number of cut nets of the bipartition obtained at each RB step [6].

The connectivity metric can be formulated in the RB-GPVS-based multiway HP by separator-vertex splitting, which is not as easy as the separator-vertex

removal method and it needs special attention. In a straightforward implementation of this method, a 2-way vertex separator $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$ is decoded as generating two subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ which are the sub-graphs of $\mathcal{G}$ induced by the vertex sets $\mathcal{V}_1 \cup \mathcal{V}_S$ and $\mathcal{V}_2 \cup \mathcal{V}_S$, respectively. That is, each separator vertex $v_s \in \mathcal{V}_S$ is split into two vertices v_s^1 and v_s^2 with $Adj(v_s^1) = Adj(v_s) \cap (\mathcal{V}_1 \cup \mathcal{V}_S)$ and $Adj(v_s^2) = Adj(v_s) \cap (\mathcal{V}_2 \cup \mathcal{V}_S)$. Then, the split vertices v_s^1 and v_s^2 are added to the subgraphs $(\mathcal{V}_1, \mathcal{E}_1)$ and $(\mathcal{V}_2, \mathcal{E}_2)$ to form $\mathcal{G}_1$ and $\mathcal{G}_2$, respectively.

This straightforward implementation of separator-vertex splitting method can be overcautious because of the unnecessary replication of separator edges in both subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. Here an edge is said to be a separator edge if two vertices connected by the edge are both in the separator $\mathcal{V}_S$. Consider a separator edge $(v_{s_1}, v_{s_2}) \in \mathcal{E}$ in a given bipartition $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$ of $\mathcal{G}$, where $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2\}$ is a bipartition of $\mathcal{H}$ induced by $\Pi_{VS}(\mathcal{G})$ according to construction given in Equation 3.3. If both $\mathcal{U}_1$ and $\mathcal{U}_2$ contain at least one node that induces the separator edge (v_{s_1}, v_{s_2}) of $\mathcal{G}$ then the replication of (v_{s_1}, v_{s_2}) in both subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ is necessary. If, however, all hypergraph nodes that induce the edge (v_{s_1}, v_{s_2}) of $\mathcal{G}$ remain in only one part of $\Pi_{\mathcal{U}}(\mathcal{H})$ then the replication of (v_{s_1}, v_{s_2}) on the graph corresponding to the other part is unnecessary. For example, if all nodes connected by both nets n_{s_1} and n_{s_2} of $\mathcal{H}$ remain in $\mathcal{U}_1$ of $\Pi_{\mathcal{U}}(\mathcal{H})$ then the edge (v_{s_1}, v_{s_2}) should be replicated in only $\mathcal{G}_1$. $\mathcal{G}_1$ and $\mathcal{G}_2$ constructed in this way become the NIG representations of hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively. Hence, the sum of the number of separator vertices of the 2-way GPVS obtained at each RB step will be equal to the total cutsize of the resulting multiway partition of $\mathcal{H}$ according to the connectivity metric.

Figure 3.2 illustrates three separator vertices v_{s_1}, v_{s_2} and v_{s_3} in a 2-way vertex separator and their splits into vertices $v_{s_1}^1$,$v_{s_2}^1$,$v_{s_3}^1$ and $v_{s_1}^2$,$v_{s_2}^2$,$v_{s_3}^2$. The three separator vertices v_{s_1}, v_{s_2} and v_{s_3} are connected with each other by three separator edges (v_{s_1}, v_{s_2}), (v_{s_1}, v_{s_3}) and (v_{s_2}, v_{s_3}) in order to show three distinct cases of separator edge replication in the accurate implementation. The figure also shows four hypergraph nodes u_x,u_y,u_z and u_t which induce the three separator edges, where u_x,u_z are assigned to part $\mathcal{U}_1$ and u_y,u_t are assigned to part $\mathcal{U}_2$. Since only u_x induces the separator edge (v_{s_1}, v_{s_2}) and u_x is assigned to $\mathcal{U}_1$, it is sufficient

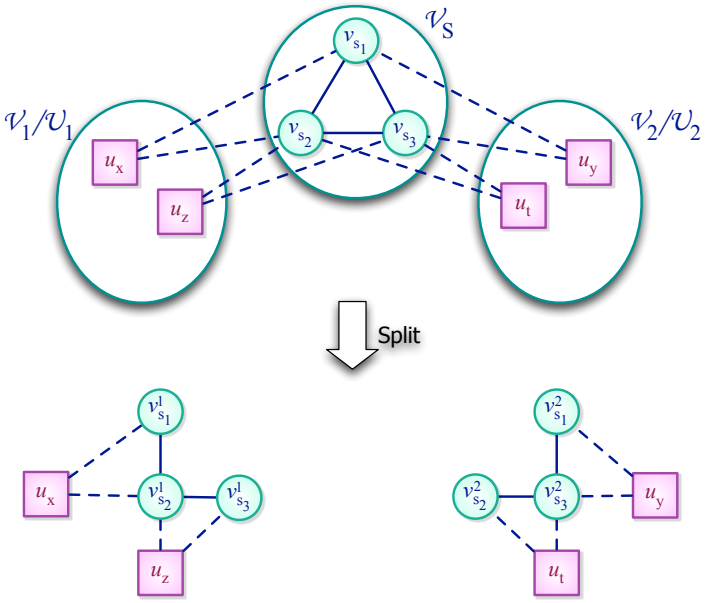


Figure 3.2: Separator-vertex splitting.

to replicate the separator edge (v_{s_1}, v_{s_2}) in only $\mathcal{V}_1$. Symmetrically, since only u_y induces the separator edge (v_{s_1}, v_{s_3}) and u_y is assigned to $\mathcal{U}_2$, it is sufficient to replicate the separator edge (v_{s_1}, v_{s_3}) in only $\mathcal{V}_2$. However, since u_z and u_t both induce the separator edge (v_{s_2}, v_{s_3}) and u_z and u_t are respectively assigned to $\mathcal{U}_1$ and $\mathcal{U}_2$, it necessary to replicate the separator edge (v_{s_2}, v_{s_3}) in both $\mathcal{V}_1$ and $\mathcal{V}_2$.

This accurate implementation of the separator-vertex splitting method depends on the availability of both $\mathcal{H}$ and its NIG representation $\mathcal{G}$ at the beginning of each RB step. Hence, after each RB step, the sub-hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ should be constructed as well as the subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$. We briefly summarize the details of the proposed implementation method performed at each RB step. A 2-way GPVS is performed on $\mathcal{G}$ to obtain a vertex separator $\Pi_{VS}(\mathcal{G})$. Then, a node bipartition $\Pi_{\mathcal{U}}(\mathcal{H})$ of $\mathcal{H}$ is constructed according to Equation 3.3 by decoding the vertex separator $\Pi_{VS}(\mathcal{G})$ of $\mathcal{G}$. Then, the 2-way vertex separator $\Pi_{VS}(\mathcal{G})$ is used together with the node bipartition $\Pi_{\mathcal{U}}(\mathcal{H})$ to generate subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ as described above. The sub-hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ are also constructed for use in subsequent RB steps. An alternative implementation could

be first generating sub-hypergraphs $\mathcal{H}_1$ and $\mathcal{H}_2$ from $\Pi_{\mathcal{U}}(\mathcal{H})$ and then constructing subgraphs $\mathcal{G}_1$ and $\mathcal{G}_2$ from $\mathcal{H}_1$ and $\mathcal{H}_2$, respectively, using NIG construction. However, this alternative implementation method is quite inefficient compared to the proposed implementation, since construction of the NIG representation from a given hypergraph is computationally expensive.

3.2.2 Vertex Weighting Scheme

Consider a node partition $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ constructed from the vertex partitioning $\Pi_{VS}(\mathcal{G}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ of NIG $\mathcal{G}$ according to Equation 3.3. Since the vertices of $\mathcal{G}$ correspond to the nets of the given hypergraph $\mathcal{H}$, it is easy to enforce a balance criterion on the nets of $\mathcal{H}$ by setting $w(v_i) = w(n_i)$. For example, assuming unit net weights, the partitioning constraint of balancing on the vertex counts of parts of $\Pi_{VS}(\mathcal{G})$ infers balance among the internal net counts of node parts of $\Pi_{\mathcal{U}}(\mathcal{H})$.

However, balance on the nodes of $\mathcal{H}$ can not be directly enforced during the GPVS of $\mathcal{G}$, because the NIG model suffers from information loss on hypergraph nodes. Here, we propose a vertex-weighting model for estimating the cumulative weight of hypergraph nodes in each vertex part $\mathcal{V}_k$ of the vertex separator $\Pi_{VS}(\mathcal{G})$. In this model, the objective is to find appropriate weights for the vertices of $\mathcal{G}$ so that vertex-part weight $W(\mathcal{V}_k)$ computed according to Equation 2.1 approximates the node-part weight $W(\mathcal{U}_k)$ computed according to Equation 2.6.

The NIG model can also be viewed as a clique-node model since each node u_h of the hypergraph induces an edge between each pair of vertices corresponding to the nets that connect u_h. So, the edges of $\mathcal{G}$ implicitly represent the nodes of $\mathcal{H}$. Each hypergraph node u_h of degree d_h induces $\binom{d_h}{2}$ clique edges among which the weight $w(u_h)$ is distributed evenly. That is, every clique edge induced by node u_h can be considered as having a uniform weight of $w(u_h)/\binom{d_h}{2}$. Multiple edges between the same pair of vertices are collapsed into a single edge whose weight is equal to the sum of the weights of its constituent edges. Hence, the weight $w(e_{ij})$

of each edge e_{ij} of $\mathcal{G}$ becomes,

$$w(e_{ij}) = \sum_{u_h \in Pins(n_i) \cap Pins(n_j)} \frac{w(u_h)}{\binom{d_h}{2}}. \tag{3.4}$$

Then, the weight of each edge is uniformly distributed between the pair of vertices connected by that edge. That is, edge e_{ij} contributes $w(e_{ij})/2$ to both v_i and v_j. Hence, in the proposed model, the weight $w(v_i)$ of vertex v_i becomes,

$$\begin{aligned} w(v_i) &= \frac{1}{2} \sum_{v_j \in Adj(v_i)} w(e_{ij}) \\ &= \sum_{u_h \in Pins(n_i)} \frac{w(u_h)}{d_h}. \end{aligned} \tag{3.5}$$

Consider an internal hypergraph node u_h of part $\mathcal{U}_k$ of $\Pi_{\mathcal{U}}(\mathcal{H})$. Since all graph vertices corresponding to the nets that connect u_h are in part $\mathcal{V}_k$ of $\Pi_{VS}(\mathcal{G})$, u_h will contribute $w(u_h)$ to $W(\mathcal{V}_k)$. Consider a boundary hypergraph node u_h of part $\mathcal{U}_k$ with an external degree $\delta_h < d_h$, i.e., u_h is connected by δ_h cut nets. Thus, u_h will contribute by an amount of $(1 - \delta_h/d_h)w(u_h)$ to $W(\mathcal{V}_k)$ instead of $w(u_h)$. So, vertex-part weight $W(\mathcal{V}_k)$ of $\mathcal{V}_k$ in $\Pi_{VS}(\mathcal{G})$ will be less than the actual node-part weight $W(\mathcal{U}_k)$ of $\mathcal{U}_k$ in $\Pi_{\mathcal{U}}(\mathcal{H})$. As the vertex-part weights of different parts of $\Pi_{VS}(\mathcal{G})$ will involve similar errors, the proposed method can be expected to produce a sufficiently good balance on the node-part weights of $\Pi_{\mathcal{U}}(\mathcal{H})$.

The free nodes can easily be exploited to improve the balance during the completion of partial node partition. For the cut-net metric in Equation 2.8, we perform free-node-to-part assignment after obtaining K-way GPVS, since arbitrary assignments of free nodes do not disturb the cutsize by Corollary 2. However, for the connectivity metric in Equation 2.9, free-node-to-part assignment needs special attention if it is performed after obtaining a K-way GPVS. According to Theorem 2, arbitrary assignments of free nodes may increase the connectivity of cut nets. So, for the connectivity cutsize metric, we perform free-node-to-part assignment after each RB step to improve the balance. Note that free-node-to-part assignment performed in this way does not increase the connectivity of cut nets in the RB-GPVS-based by Corollary 2. For both cutsize metrics, the best-fit-decreasing heuristic [40] used in solving the bin-packing problem is adapted to

obtain a complete node partition/bipartition. Free nodes are assigned to parts in decreasing weight, where the best-fit criterion corresponds to assigning a free node to a part that currently has the minimum weight. Initial part weights are taken as the weights of the two parts in partial node bipartition.

3.3 Adapted Multilevel Implementation of GPVS

The state-of-the-art graph and hypergraph partitioning tools that adopt the multilevel framework, consist of three phases: *coarsening*, *initial partitioning*, and *uncoarsening*. In the first phase, a multilevel clustering is applied starting from the original graph/hypergraph by adopting various matching heuristics until the number of vertices in the coarsened graph/hypergraph reduces below a predetermined threshold value. Clustering corresponds to coalescing highly interacting vertices to supernodes. In the second phase, a partition is obtained on the coarsest graph/hypergraph using various heuristics including FM, which is an iterative refinement heuristic proposed for graph/hypergraph partitioning by Fiduccia and Mattheyses [41] as a faster implementation of the KL algorithm proposed by Kernighan and Lin [42]. In the third phase, the partition found in the second phase is successively projected back towards the original graph/hypergraph by refining the projected partitions on the intermediate level uncoarserned graphs/hypergraphs using various heuristics including FM.

One of the most important applications of GPVS is George's *nested–dissection* algorithm [43,44], which has been widely used for reordering of the rows/columns of a symmetric, sparse, and positive definite matrix to reduce *fill* in the factor matrices. Here, GPVS is defined on the standard graph model of the given symmetric matrix. The basic idea in the nested dissection algorithm is to reorder a symmetric matrix into a 2-way DB form so that no fill can occur in the off-diagonal blocks. The DB form of the given matrix is obtained through a symmetric row/column permutation induced by a 2-way GPVS. Then, both diagonal blocks are reordered by applying the dissection strategy recursively. The performance of the nested-dissection reordering algorithm depends on finding small

vertex separators at each dissection step.

In this work, we adapted and modified the *onmetis* ordering code of *MeTiS* [45] for implementing our GPVS-based HP formulation. *onmetis* utilizes the RB paradigm for obtaining multiway GPVS. Since K is not known in advance for ordering applications, recursive bipartitioning operations continue until the weight of a part becomes sufficiently small. In our implementation, we terminate the recursive bipartitioning process whenever the number of parts becomes K.

The separator refinement scheme used in the uncoarsening phase of *onmetis* considers vertex moves from vertex separator $\Pi_{VS}(\mathcal{G})$ to both $\mathcal{V}_1$ and $\mathcal{V}_2$ in $\Pi_{VS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$. During these moves, *onmetis* uses the following feasibility constraint, which incorporates the size of the separator in balancing, i.e.,

$$\max\{W(\mathcal{V}_1), W(\mathcal{V}_2)\} \leq (1+\epsilon)\frac{W(\mathcal{V}_1)+W(\mathcal{V}_2)+W(\mathcal{V}_S)}{2} = W_{max}. \tag{3.6}$$

However, this may become a loose balancing constraint compared to Equation 2.2 for relatively large separator sizes which is typical during refinements of coarser graphs. This loose balancing constraint is not an important concern in *onmetis*, because it is targeted for fill-reducing sparse matrix ordering which is not very sensitive to the imbalance between part sizes. Nevertheless, this scheme degrades the load balancing quality of our GPVS-based HP implementation, where load balancing is more important in the applications for which HP is utilized. We modified *onmetis* by computing the maximum part weight constraint as

$$W_{max} = (1+\epsilon)\frac{W(\mathcal{V}_1) + W(\mathcal{V}_2)}{2}. \tag{3.7}$$

at the beginning of each FM pass, whereas *onmetis* computes W_{max} according to Equation 3.6 once for all FM passes, in a level. Furthermore, *onmetis* maintains only one value for each vertex which denotes both the weight and the cost of the vertex. We added a second field for each vertex to hold the weight and the cost of the vertex separately. The weights and the costs of vertices are accumulated independently during vertex coalescings performed by matchings at the coarsening phases. Recall that weight values are used for maintaining the load balancing criteria, whereas cost values are used for computing the size of the separator.

That is, FM gains of the separator vertices are computed using the cost values of those vertices.

The GPVS-based HP implementation obtained by adapting *onmetis* as described in this subsection will be referred to as *onmetisHP*.

3.4 Experimental Results

We test the performance of our GPVS-based HP formulation by partitioning matrices from the linear-programming (LP) and the positive definite (PD) matrix collections of the University of Florida matrix collection [46]. Matrices in the latter collection are square and symmetric, whereas the matrices in the former collection are rectangular. The row-net hypergraph models [6, 9] of the test matrices constitute our test set. In these hypergraphs, nets are associated with unit cost. To show the validity of our GPVS-based HP formulation, test hypergraphs are partitioned by both *PaToH* and *onmetisHP* and default parameters are utilized in both tools. In general, the maximum imbalance ratio ϵ was set to be 10%.

We excluded small matrices that have less than 1000 rows or 1000 columns. In the LP matrix collection, there were 190 large matrices out of 342 matrices. Out of these 190 large matrices, 5 duplicates, 1 extremely large matrix and 5 matrices, for which NIG representations are extremely large were excluded. We also excluded 26 outlier matrices which yield large separators[1] to avoid skewing the results. Thus, 153 test hypergraphs are used from the LP matrix collection. In the PD matrix collection, there were 170 such large matrices out of 223 matrices. Out of these 170 large matrices, 2 duplicates, 2 matrices, for which NIG representations are extremely large and 7 matrices with large separators were excluded. Thus, 159 test hypergraphs are used from the PD matrix collection. We experimented with K-way partitioning of test hypergraphs for $K = 2, 4, 8, 16, 32, 64$, and 128. For a specific K value, K-way partitioning of a

[1]Here, a separator is said to be large if it includes more than 33% of all nets.

test hypergraph constitutes a partitioning instance. For the LP collection, instances in which $min\{|\mathcal{U}|, |\mathcal{N}|\} < 50K$ are discarded as the parts would become too small. So, 153, 153, 153, 153, 135, 100, and 65 hypergraphs are partitioned for $K = 2, 4, 8, 16, 32, 64$, and 128, respectively, for the LP collection. Similarly for the PD collection, instances in which $|\mathcal{U}| < 50K$ are discarded. So, 159, 159, 159, 159, 145, 131, and 109 hypergraphs are partitioned for $K = 2, 4, 8, 16, 32, 64$, and 128, respectively for the PD collection. In this section, we summarize our findings in these experiments.

In our first set of experiments, the hypergraphs obtained from the LP matrix collection are used for permuting the matrices into singly-bordered (SB) block-angular-form for coarse-grain parallelization of linear-programming applications [35]. Here, minimizing the cutsize according to the cut-net metric Equation 2.4 corresponds to minimizing the size of the row border in the induced SB form. In these applications, nets are either have unit weights or weights that are equal to the nonzeros in the respective rows. In the former case, net balancing corresponds to balancing the row counts of the diagonal blocks, whereas in the latter case, net balancing corresponds to balancing the nonzero counts of the diagonal blocks. Experimental comparisons are provided only for the former case, because *PaToH* does not support different cost and weight associations to nets.

In our second set of experiments, the hypergraphs obtained from the PD matrix collection are used for minimizing communication overhead in a column-parallel matrix-vector multiply algorithm in iterative solvers. Here, minimizing the cutsize according to the connectivity metric Equation 2.5 corresponds to minimizing the total communication volume when the point-to-point inter-processor communication scheme is used [6]. Minimizing the cutsize according to the cut-net metric Equation 2.4 corresponds to minimizing the total communication volume when the collective communication scheme is used [9]. In these applications, nodes have weights that are equal to the number of nonzeros in the respective columns. So, balancing part weights corresponds to computational load balancing. All experiments are conducted sequentially on a 24-core PC equipped with quad 2.1Ghz 6-core AMD Opteron processors with 6 128 KB L1, and 512 KB L2 caches, and a single 6MB L3 cache. The system is 128 GB memory and runs

Debian Linux v5.0.5.

In the following tables, the performance figures are computed and displayed as follows. Since both *PaToH* and *onmetisHP* tools involve randomized heuristics, 10 different partitions are obtained for each partitioning instance and the geometric average of the 10 resultant partitions are computed as the representative results for both HP tools on the particular partitioning instance. For each partitioning instance, the cutsize value is normalized with respect to the total number of nets in the respective hypergraph. Recall that all test hypergraphs have unit-cost nets. So, for the cut-net metric, these normalized cutsize values show the fraction of the cut nets. For the connectivity metric, these normalized cutsize values show the average net connectivity. For each partitioning instance, the running time of *PaToH* is normalized with respect to that of *onmetisHP*, thus showing the speedup obtained by *onmetisHP* for that partitioning instance. These normalized cutsize values and speedup values as well as percent load imbalance values are summarized in the tables by taking the geometric averages for each K value.

Table 3.1: Performance averages on the LP matrix collection for the cut-net metric with net balancing.

	PaToH		*onmetisHP*		
K	cutsize	$\%LI$	cutsize	$\%LI$	speedup
2	0.02	1.2	0.03	0.3	2.04
4	0.02	1.9	0.05	2.6	2.45
8	0.07	3.1	0.09	6.9	2.64
16	0.09	5.2	0.14	13.0	2.78
32	0.13	8.8	0.18	23.1	2.83
64	0.15	11.5	0.21	27.8	2.83
128	0.16	13.5	0.21	31.3	2.76

Table 3.1 displays overall performance averages of *onmetisHP* compared to those of *PaToH* for the cut-net metric (see Equation 2.8) with net balancing on the LP matrix collection. As seen in Table 3.1, *onmetisHP* obtains hypergraph partitions of comparable cutsize quality with those of *PaToH*. However, load balancing quality of partitions produced by *onmetisHP* is worse than those of *PaToH*, especially with increasing K. As seen in the table, *onmetisHP* runs

significantly faster than *PaToH* for each K. For example, *onmetisHP* runs 2.83 times faster than *PaToH* for 32-way partitionings on the average.

Table 3.2: Performance averages on the PD matrix collection for the cut-net metric with node balancing.

	PaToH		*onmetisHP*				
K	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
2	0.01	0.1	0.01	0.2	0.2	0.1	1.40
4	0.03	0.3	0.03	0.9	1.5	1.1	1.75
8	0.05	0.4	0.05	2.8	3.7	2.7	1.96
16	0.08	0.6	0.08	6.7	7.4	5.4	1.98
32	0.12	0.9	0.12	13.4	12.8	9.2	2.17
64	0.17	1.2	0.16	22.1	19.8	13.5	2.27
128	0.25	1.6	0.24	32.5	28.8	17.9	2.25

Table 3.2 displays overall performance averages of *onmetisHP* compared to those of *PaToH* for the cut-net metric with node balancing on the PD matrix collection. In the table, $exp\%LI_p$ and $act\%LI_p$ respectively denote the expected and actual percent load imbalance values for the partial node partitions of the hypergraphs induced by K-way GPVS. $act\%LI_c$ denotes the actual load imbalance values for the complete node partitions obtained after free-node-to-part assignment. The small discrepancies between the $exp\%LI_p$ and $act\%LI_p$ values show the validity of the approximate weighting scheme proposed in Section 3.1 for the vertices of the NIG. As seen in the table, for each K, the $act\%LI_c$ value is considerably smaller than the $act\%LI_p$ value. This experimental finding confirms the effectiveness of the free-node-to-part assignment scheme mentioned in Section 3.1. As seen in Table 3.2, *onmetisHP* obtains hypergraph partitions of comparable cutsize quality with those of *PaToH*. However, load balancing quality of partitions produced by *onmetisHP* is considerably worse than those of *PaToH*. As seen in the table, *onmetisHP* runs considerably faster than *PaToH* for each K.

Table 3.3 is constructed based on the PD matrix collection to show the validity of the accurate vertex splitting formulation proposed in Section 3.2.1 for the connectivity cutsize metric (see Equation 2.9). In this table, speedup, cutsize and load imbalance values of *onmetisHP* that uses the straightforward (overcautious)

Table 3.3: Comparison of accurate and overcautious separator-vertex splitting implementations with averages on the PD matrix collection for the connectivity metric with node balancing.

	overcautious / *accurate*		
K	cutsize	$\%LI$	speedup
2	1.00	0.63	1.07
4	1.02	0.79	1.13
8	1.10	0.79	1.16
16	1.29	0.70	1.19
32	1.56	0.64	1.21
64	1.84	0.69	1.22
128	2.09	0.60	1.21

separator-vertex splitting implementation are normalized with respect to those of *onmetisHP* that uses the accurate implementation. In the straightforward implementation, free-node-to-part assignment is performed after obtaining a K-way GPVS, since hypergraphs are not carried through the RB process. Free nodes are assigned to parts in decreasing weight, where the best-fit criterion corresponds to assigning a free node to a part which increases connectivity cutsize by the smallest amount with ties broken in favor of the part with minimum weight. As seen in the table, the overcautious implementation leads to slightly better load balance than accurate implementation, because overcautious implementation performs free-node-to-part assignment on the K-way partial node partition induced by the K-way GPVS. As also seen in the table, the overcautious implementation, as expected, leads to slightly better speedup than the accurate implementation. However, the accurate implementation leads to significantly less cutsize values.

Table 3.4 displays overall performance averages of *onmetisHP* compared to those of *PaToH* for the connectivity metric with node balancing on the PD matrix collection. In contrast to Table 3.2, load imbalance values are not displayed for partial node partitions in Table 3.4, because free-node-to-part assignments are performed after each 2-way GPVS operation for the sake of accurate implementation of the separator-vertex splitting method as mentioned in Section 3.1. So, $\%LI$ values displayed in Table 3.4 show the actual percent imbalance values for the K-way node partitions obtained. As seen in Table 3.4, similar to results of Table 3.2, *onmetisHP* obtains hypergraph partitions of comparable cutsize

Table 3.4: Performance averages on the PD matrix collection for the connectivity metric with node balancing.

	PaToH		*onmetisHP*		
K	cutsize	$\%LI$	cutsize	$\%LI$	speedup
2	1.03	0.1	1.03	0.2	1.29
4	1.08	0.3	1.08	0.8	1.50
8	1.15	0.5	1.15	1.7	1.61
16	1.26	0.7	1.25	4.1	1.63
32	1.37	1.0	1.36	7.9	1.61
64	1.49	1.5	1.47	11.8	1.60
128	1.63	1.9	1.60	16.5	1.54

quality with those of *PaToH*, whereas load balancing quality of partitions produced by *onmetisHP* is considerably worse than those of *PaToH*. As seen in Table 4, *onmetisHP* still runs considerably faster than *PaToH* for each K for the connectivity metric. However, the speedup values in Table 3.4, are considerable smaller compared to those displayed in Table 3.2, which is due to the fact that *onmetisHP* carries hypergraphs during the RB process for the sake of accurate implementation of the separator-vertex splitting method as mentioned in Section 3.1.

A common observation about Tables 3.1, 3.2, and 3.4, is the increasing speedup of *onmetisHP* compared to *PaToH* with increasing K values. This experimental finding stems from the fact that the initial NIG construction overhead amortizes with increasing K. Another common observation is that *onmetisHP* runs significantly faster than *PaToH*, while producing partitions of comparable cutsize quality with, however, worse load balancing quality. These experimental findings justify our GPVS-based hypergraph partitioning formulation for effective parallelization of applications in which computational balance definition is not very precise and preprocessing overhead due to partitioning overhead is important.

Table 3.5: Hypergraph and NIG properties for matrices of LP and PD matrix collections.

LP Collection				PD Collection		
name	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$	name	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$
lp_truss	1000	8806	25122	msc01050	1050	136594
rosen2	1032	3080	31800	bcsstm08	1074	0
lp_ship12s	1042	2869	10690	bcsstm09	1083	0
lp_ship12l	1042	5533	21346	bcsstk09	1083	70206
lp_sctap2	1090	2500	11010	bcsstk10	1086	53418
lp_woodw	1098	8418	40842	1138_bus	1138	10004
lp_osa_07	1118	25067	104932	bcsstk27	1224	136882
qiulp	1192	1900	12144	mhd1280b	1280	26362
lp_sierra	1227	2735	9872	plbuckle	1282	79330
lp_ganges	1309	1706	15312	msc01440	1440	149808
model4	1337	4962	87914	bcsstk11	1473	92714
lp_pilot	1441	4860	123076	bcsstm11	1473	0
lp_sctap3	1480	3340	14772	bcsstm12	1473	52142
lp_degen3	1503	2604	100356	bcsstk12	1473	92714
fxm2-6	1520	2845	26656	ex33	1733	59050
cep1	1521	4769	196152	bcsstk14	1806	193848
primagaz	1554	10836	21658	ex3	1821	193498
pcb1000	1565	2820	32902	nasa1824	1824	140442
model3	1609	4565	43084	plat1919	1919	98990
progas	1650	1900	26210	bcsstm26	1922	0
model5	1744	11802	173646	bcsstk26	1922	90608
scrs8-2b	1820	3499	203016	bcsstk13	2003	394770
lp_cycle	1890	3371	55428	nasa2146	2146	189396
deter0	1923	5468	12466	ex10	2410	191524
lp_pilot87	2030	6680	236594	Chem97ZtZ	2541	88824
rosen10	2056	6152	47160	ex10hs	2548	202682
model6	2094	5289	62046	ex13	2568	277316
p6000	2095	7947	8964	nasa2910	2910	887840
lp_stocfor2	2157	3045	25476	bcsstk23	3134	217498
lp_d2q06c	2171	5831	53982	bcsstm23	3134	0
lp_80bau3b	2262	11934	20148	mhd3200b	3200	30944
nemspmm2	2301	8734	101804	bibd_81_2	3240	0
lp_bnl2	2324	4486	26914	ex9	3363	370452
lp_osa_14	2337	54797	227686	bcsstm24	3562	0

Continued on next page

Table 3.5 – continued from previous page

LP Collection				PD Collection		
name	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$	name	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$
nemspmm1	2362	8903	111902	bcsstk24	3562	442912
lp_greenbea	2389	5598	67682	bcsstk21	3600	89472
lpi_greenbea	2390	5596	67694	bcsstm21	3600	0
lp_ken_07	2426	3602	11956	bcsstk15	3948	523740
scagr7-2c	2447	3479	257282	sts4098	4098	1085428
lpi_gran	2604	2525	194708	t2dal_e	4257	0
lpi_bgindy	2671	10880	124076	bcsstk28	4410	591662
l30	2701	16281	53428	msc04515	4515	265404
model9	2787	10939	101082	nasa4704	4704	356788
model8	2896	6464	53908	mhd4800b	4800	46552
lp_pds_02	2953	7716	20328	crystm01	4875	395322
lp22	2958	16392	221064	bcsstk16	4884	1033898
lp_cre_c	2986	6411	37810	s3rmt3m3	5357	540084
lpi_cplex1	3005	5224	2262516	s3rmt3m1	5489	573546
plddb	3069	5049	19586	s2rmq4m1	5489	749964
rat	3136	9408	1245198	s1rmt3m1	5489	573546
lp_maros_r7	3136	9408	660944	s1rmq4m1	5489	749964
delf	3170	5598	30338	s2rmt3m1	5489	573546
stat96v4	3173	63076	51540	s3rmq4m1	5489	749964
deter4	3235	9133	86758	ex15	6867	259938
lpl2	3294	10881	36762	Kuu	7102	1555534
model7	3358	9560	94080	Muu	7102	774216
sctap1-2c	3390	7458	273912	bcsstk38	8032	1660234
lp_cre_a	3428	7248	41496	aft01	8205	426542
lpi_ceria3d	3576	4400	1959730	fv1	9604	224652
ch	3700	8291	50464	fv3	9801	229320
aircraft	3754	7517	2834250	fv2	9801	229320
lpi_gosh	3790	13455	202218	bundle1	10581	24062342
deter8	3831	10905	34624	ted_B	10605	479178
fxm2-16	3900	7335	70906	ted_B_unscaled	10605	479178
nemsemm1	3945	75310	393474	msc10848	10848	6174798
pcb3000	3960	7732	84924	bcsstk17	10974	1395962
pgp2	4034	13254	1347842	t2dah_e	11445	602052
rlfddd	4050	61521	376536	bcsstk18	11948	701260
deter6	4255	12113	40868	cbuckle	13681	2255450
large	4282	7297	46414	crystm02	13965	1294602

Continued on next page

Table 3.5 – continued from previous page

LP Collection				PD Collection		
name	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$	name	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$
lp_osa_30	4350	104374	432388	Pres_Poisson	14822	2235694
stormg2-8	4393	11322	50684	bcsstm25	15439	0
model10	4400	16819	288860	bcsstk25	15439	1153480
nsir	4453	10057	469684	Dubcova1	16129	981872
seymourl	4944	6316	1208040	olafu	16146	3372106
cq5	5048	11748	112872	gyro_m	17361	1908612
p05	5090	9590	219438	gyro	17361	5760558
deter5	5103	14529	54796	bodyy4	17546	314540
scsd8-2b	5130	35910	1408030	bodyy5	18589	333146
r05	5190	9690	400968	bodyy6	19366	346860
bas1lp	5411	9825	2591680	raefsky4	19779	5322790
deter1	5527	15737	62480	LFAT5000	19994	129928
co5	5774	12325	125918	LF10000	19998	179956
stat96v1	5995	197472	69024	t3dl_e	20360	0
lp_dfl001	6071	12230	76196	msc23052	23052	3623204
deter2	6095	17313	120428	bcsstk36	23052	3611816
fxm3_6	6200	12625	105616	crystm03	24696	2388726
deter7	6375	18153	79288	smt	25710	19418850
lp_cre_d	6476	73948	363340	thread	29736	24648426
ulevimin	6590	46937	198008	wathen100	30401	1627220
nemswrld	6647	28550	354774	ship_001	34920	25565618
nemsemm2	6943	48878	138470	nd12k	36000	90870894
nl	7039	15325	98050	wathen120	36441	1953940
lp_cre_b	7240	77137	389158	obstclae	40000	472820
deter3	7647	21777	108100	jnlbrng1	40000	476004
rlfdual	8052	74970	714646	minsurfo	40806	486844
scsd8-2r	8650	60550	3896670	bcsstm39	46772	0
cq9	9278	21534	212312	vanbody	47072	8006490
pf2177	9728	9908	715416	gridgena	48962	1638710
scagr7-2b	9743	13847	3928898	cvxbqp1	50000	1049432
lp_pds_06	9881	29351	78122	ct20stif	52329	9964622
p010	10090	19090	438228	crankseg_1	52804	75044100
ge	10099	16369	102030	nasasrb	54870	8279516
lp_osa_60	10280	243246	1006074	Andrews	60000	5451632
co9	10789	22924	238416	crankseg_2	63838	104526330
lpl3	10828	33686	116590	Dubcova2	65025	4027504

Continued on next page

Table 3.5 – continued from previous page

LP Collection				PD Collection		
name	$\lvert\mathcal{N}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$	name	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{E}\rvert$
fome11	12142	24460	152392	qa8fm	66127	7285062
scrs8-2r	14364	27691	12404296	cfd1	70656	8088220
stormg2-27	14387	37485	205610	nd24k	72000	189118604
lp_ken_11	14694	21349	67760	oilpan	73752	11536112
sctap1-2b	15390	33858	5245512	finan512	74752	4522496
car4	16384	33052	182624	apache1	80800	1776124
lp_pds_10	16558	49932	133100	shallow_water1	81920	737280
lp_stocfor3	16675	23541	218144	shallow_water2	81920	737280
ex3sta1	17443	17516	662414	thermal1	82654	1519688
testbig	17613	31223	3274430	denormal	89400	3540180
dbir1	18804	45775	2419194	s3dkt3m2	90449	10025526
dbir2	18906	45877	2618552	s3dkq4m2	90449	13192104
scfxm1-2b	19036	33047	519242	m_t1	97578	36435564
route	20894	43019	1273910	2cubes_sphere	101492	8873034
ts-palko	22002	47235	8149338	thermomech_TK	102158	1866380
fxm4_6	22400	47185	504136	thermomech_TC	102158	1866380
fome12	24284	48920	304784	x104	108384	38593344
e18	24617	38601	780314	shipsec8	114919	22608304
pltexpa	26894	70364	242842	ship_003	121728	32654210
baxter	27441	30733	1196786	cfd2	123440	13295204
lp_ken_13	28632	42659	133172	boneS01	127224	25388478
stat96v2	29089	957432	323660	shipsec1	140874	23945538
lp_pds_20	33798	108175	286322	bmw7st_1	141347	23432912
stat96v3	33841	1113780	375972	Dubcova3	146689	17334072
world	34506	67147	547558	bmwcra_1	148770	49534938
mod2	34774	66409	570136	G2_circuit	150102	1852894
sc205-2r	35213	62423	12948830	shipsec5	179860	32159300
scfxm1-2r	37980	65943	1593802	thermomech_dM	204316	3732760
fxm3_16	41340	85575	724186	pwtk	217918	32554318
dbic1	43200	226317	2721302	hood	220542	34021638
fome13	48568	97840	609568	BenElechi1	245874	36015470
pds-30	49788	158489	418478	offshore	259789	23096456
rlfprim	58866	62712	9060730	F1	343791	224140612
stormg2-125	65935	172431	1887584	msdoor	415863	62406596
pds-40	66641	217531	571226	af_2_k101	503625	46968400
fome21	67596	216350	572644	af_5_k101	503625	46968400

Continued on next page

Table 3.5 – continued from previous page

LP Collection				PD Collection		
name	$\|\mathcal{N}\|$	$\|\mathcal{U}\|$	$\|\mathcal{E}\|$	name	$\|\mathcal{U}\|$	$\|\mathcal{E}\|$
pds-50	82837	275814	719666	af_1_k101	503625	46968400
pds-60	99204	336421	873016	af_4_k101	503625	46968400
pds-70	114717	390005	1008932	af_3_k101	503625	46968400
pds-80	128954	434580	1120120	af_0_k101	503625	46968400
pds-90	142596	475448	1221102	inline_1	503712	252580926
pds-100	156016	514577	1314672	af_shell8	504855	47055520
watson_1	201155	386992	1736008	af_shell3	504855	47055520
sgpf5y6	246077	312540	2530568	af_shell4	504855	47055520
watson_2	352013	677224	3038266	af_shell7	504855	47055520
stormG2_1000	526185	1377306	82461084	parabolic_fem	525825	9434110
cont11_l	1468599	1961394	16595662	apache2	715176	15848148
				tmt_sym	726713	13776468
				boneS10	914898	222646668
				ldoor	952203	144470732
				ecology2	999999	11979976
				thermal2	1228045	22790012
				G3_circuit	1585478	19681656

Table 3.6: 2-way partitioning performance of the LP matrix collection for cut-net metric with net balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
lp_truss	0.05	9.9%	0.04	2.2%	4.81
rosen2	0.01	0.0%	0.01	0.0%	1.50
lp_ship12s	0.02	0.1%	0.01	0.0%	2.47
lp_ship12l	0.01	0.1%	0.01	0.0%	3.63
lp_sctap2	0.04	1.0%	0.04	1.6%	2.17
lp_woodw	0.05	0.6%	0.06	1.6%	8.81
lp_osa_07	0.07	0.1%	0.06	1.1%	2.85
qiulp	0.11	7.2%	0.13	0.0%	2.63
lp_sierra	0.04	2.1%	0.03	0.1%	2.26
lp_ganges	0.02	0.1%	0.02	0.0%	2.74

Continued on next page

Table 3.6 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
model4	0.08	4.6%	0.07	3.2%	4.36
lp_pilot	0.16	7.9%	0.18	0.0%	2.91
lp_sctap3	0.03	1.4%	0.03	1.3%	2.45
lp_degen3	0.12	5.6%	0.16	4.7%	2.85
fxm2-6	0.03	6.9%	0.03	0.0%	1.76
cep1	0.28	0.9%	0.55	0.5%	0.35
primagaz	0.00	0.1%	0.00	99.0%	0.45
pcb1000	0.03	0.1%	0.03	0.0%	5.16
model3	0.02	9.8%	0.05	0.0%	2.11
progas	0.02	2.0%	0.02	1.8%	1.65
model5	0.00	0.1%	0.00	0.1%	13.15
scrs8-2b	0.13	11.8%	0.14	5.7%	0.41
lp_cycle	0.02	5.3%	0.03	2.9%	1.74
deter0	0.07	8.4%	0.07	0.2%	1.97
lp_pilot87	0.19	6.3%	0.31	2.1%	3.05
rosen10	0.00	0.0%	0.00	40.7%	1.06
model6	0.02	2.0%	0.04	3.2%	3.27
p6000	0.00	0.0%	0.00	47.4%	0.71
lp_stocfor2	0.00	0.9%	0.00	1.5%	1.17
lp_d2q06c	0.05	3.0%	0.06	0.0%	2.82
lp_80bau3b	0.04	9.8%	0.03	0.3%	4.14
nemspmm2	0.05	2.0%	0.03	3.1%	10.49
lp_bnl2	0.05	3.8%	0.05	2.1%	2.25
lp_osa_14	0.03	1.5%	0.03	0.0%	5.95
nemspmm1	0.07	3.2%	0.03	4.2%	5.69
lp_greenbea	0.03	0.0%	0.04	0.0%	2.81
lpi_greenbea	0.04	1.3%	0.04	0.0%	3.07
lp_ken_07	0.01	2.0%	0.01	2.0%	1.63
scagr7-2c	0.11	9.5%	0.45	6.0%	0.21
lpi_gran	0.00	6.4%	0.09	1.1%	0.43
lpi_bgindy	0.08	7.9%	0.07	1.1%	17.59
l30	0.04	5.6%	0.03	1.7%	4.06
model9	0.01	0.0%	0.03	3.6%	3.70
model8	0.05	4.7%	0.05	0.0%	3.34
lp_pds_02	0.03	1.4%	0.03	0.0%	2.29
lp22	0.32	6.7%	0.50	3.6%	2.75

Continued on next page

Table 3.6 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
lp_cre_c	0.02	0.0%	0.02	1.2%	3.06
lpi_cplex1	0.31	9.8%	0.70	9.7%	0.25
plddb	0.00	6.9%	0.00	0.0%	1.27
rat	0.22	9.2%	0.18	0.0%	1.67
lp_maros_r7	0.09	3.5%	0.09	0.0%	1.32
delf	0.01	9.0%	0.01	0.6%	1.77
stat96v4	0.01	8.3%	0.01	0.0%	20.74
deter4	0.15	9.5%	0.10	0.0%	1.01
lpl2	0.04	3.0%	0.07	3.2%	3.62
model7	0.02	6.2%	0.00	0.1%	2.55
sctap1-2c	0.07	0.0%	0.17	0.0%	0.35
lp_cre_a	0.02	0.0%	0.02	0.5%	2.76
lpi_ceria3d	0.29	1.9%	0.65	11.8%	0.20
ch	0.06	0.0%	0.15	1.9%	2.33
aircraft	0.20	0.0%	0.11	6.5%	0.06
lpi_gosh	0.05	3.2%	0.06	0.0%	4.64
deter8	0.07	9.9%	0.07	0.0%	1.87
fxm2-16	0.02	9.8%	0.02	2.4%	1.48
nemsemm1	0.00	5.0%	0.02	4.5%	44.06
pcb3000	0.02	5.8%	0.02	0.0%	7.17
pgp2	0.29	3.0%	0.65	0.5%	0.21
rlfddd	0.04	6.9%	0.06	0.9%	12.82
deter6	0.07	9.5%	0.07	0.0%	1.72
large	0.01	0.7%	0.01	0.0%	1.75
lp_osa_30	0.02	0.0%	0.02	2.0%	0.98
stormg2-8	0.01	0.6%	0.01	2.0%	1.83
model10	0.05	4.2%	0.09	6.7%	4.42
nsir	0.09	3.6%	0.13	7.9%	3.91
seymourl	0.28	9.8%	0.42	3.8%	0.30
cq5	0.02	1.0%	0.04	1.3%	6.40
p05	0.02	7.2%	0.07	0.0%	15.81
deter5	0.07	9.8%	0.07	0.0%	1.61
scsd8-2b	0.25	6.7%	0.84	0.9%	0.51
r05	0.04	7.0%	0.10	0.0%	17.39
bas1lp	0.07	5.4%	0.12	0.0%	1.78
deter1	0.07	9.6%	0.07	0.0%	1.53

Continued on next page

Table 3.6 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
co5	0.02	5.6%	0.05	0.7%	6.21
stat96v1	0.01	2.1%	0.01	0.0%	21.38
lp_dfl001	0.09	9.6%	0.20	1.8%	2.52
deter2	0.10	9.4%	0.09	0.0%	1.24
fxm3_6	0.00	1.9%	0.00	1.9%	1.74
deter7	0.07	9.4%	0.07	0.0%	1.44
lp_cre_d	0.05	0.0%	0.15	0.8%	18.10
ulevimin	0.04	3.8%	0.11	1.7%	3.28
nemswrld	0.08	0.0%	0.05	1.8%	7.93
nemsemm2	0.00	0.5%	0.00	3.6%	7.90
nl	0.04	0.0%	0.07	1.3%	3.48
lp_cre_b	0.05	0.2%	0.09	1.3%	17.46
deter3	0.07	9.9%	0.08	0.0%	1.49
rlfdual	0.05	8.3%	0.06	0.0%	5.85
scsd8-2r	0.25	8.0%	0.91	0.0%	0.37
cq9	0.02	8.2%	0.05	0.0%	6.73
pf2177	0.23	0.1%	0.70	0.5%	0.37
scagr7-2b	0.11	9.8%	0.47	4.0%	0.07
lp_pds_06	0.03	5.8%	0.04	2.7%	2.49
p010	0.01	0.0%	0.02	0.0%	0.36
ge	0.02	0.4%	0.02	0.0%	1.52
lp_osa_60	0.01	0.0%	0.01	7.3%	0.85
co9	0.02	6.8%	0.14	0.0%	6.58
lpl3	0.01	1.1%	0.07	2.2%	7.28
fome11	0.00	0.0%	0.00	0.0%	2.99
scrs8-2r	0.30	40.4%	0.14	10.3%	0.13
stormg2-27	0.01	0.5%	0.01	3.5%	1.81
lp_ken_11	0.00	0.8%	0.00	3.9%	2.44
sctap1-2b	0.07	0.0%	0.54	5.3%	0.13
car4	0.00	0.0%	0.00	0.0%	0.13
lp_pds_10	0.03	8.8%	0.05	0.0%	2.80
lp_stocfor3	0.00	0.2%	0.00	0.2%	0.95
ex3sta1	0.07	8.4%	0.07	1.4%	1.12
testbig	0.09	9.9%	0.09	3.2%	0.12
dbir1	0.06	10.0%	0.30	22.6%	5.11
dbir2	0.06	8.0%	0.37	6.8%	4.82

Continued on next page

Table 3.6 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
scfxm1-2b	0.03	9.2%	0.04	0.0%	0.67
route	0.01	0.0%	0.01	0.0%	2.21
ts-palko	0.12	9.9%	0.94	18.6%	2.08
fxm4_6	0.00	0.5%	0.00	0.5%	1.78
fome12	0.00	0.0%	0.00	0.0%	2.91
e18	0.23	9.8%	0.68	1.2%	1.18
pltexpa	0.00	0.7%	0.00	0.7%	1.90
baxter	0.01	4.3%	0.00	1.6%	0.66
lp_ken_13	0.00	0.6%	0.00	2.7%	3.35
stat96v2	0.00	1.1%	0.00	0.0%	22.45
lp_pds_20	0.02	0.9%	0.03	0.0%	3.04
stat96v3	0.00	6.3%	0.00	0.0%	11.82
world	0.00	6.3%	0.02	4.9%	2.11
mod2	0.00	6.9%	0.01	4.9%	1.97
sc205-2r	0.06	9.2%	0.20	3.6%	0.07
scfxm1-2r	0.03	9.8%	0.11	0.3%	0.56
fxm3_16	0.00	0.4%	0.00	0.5%	1.72
dbic1	0.01	0.0%	0.01	0.5%	9.51
fome13	0.00	0.0%	0.00	0.0%	2.74
pds-30	0.03	9.0%	0.02	0.0%	3.05
rlfprim	0.06	4.8%	0.08	2.2%	0.25
stormg2-125	0.02	8.9%	0.01	0.7%	1.25
pds-40	0.02	1.1%	0.02	0.9%	3.35
fome21	0.00	0.0%	0.00	0.0%	3.18
pds-50	0.03	0.3%	0.01	1.2%	3.28
pds-60	0.02	1.8%	0.01	0.4%	3.37
pds-70	0.02	6.9%	0.01	0.0%	3.44
pds-80	0.02	0.1%	0.01	0.4%	3.43
pds-90	0.02	6.2%	0.01	0.0%	3.19
pds-100	0.01	0.6%	0.01	0.0%	3.20
watson_1	0.00	0.0%	0.00	0.0%	2.91
sgpf5y6	0.00	4.1%	0.00	3.7%	1.67
watson_2	0.00	4.7%	0.00	4.8%	2.86
stormG2_1000	0.01	4.4%	0.03	3.3%	0.36
cont11_l	0.00	0.0%	0.00	0.0%	1.06

Table 3.7: 2-way partitioning performance of the PD matrix collection for cut-net metric with node balancing.

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
msc01050	0.21	2.3%	0.30	1.7%	3.2%	3.1%	1.27
bcsstm08	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.45
bcsstm09	0.00	0.1%	0.00	0.1%	0.1%	0.1%	2.39
bcsstk09	0.11	3.7%	0.12	0.2%	0.2%	0.2%	1.94
bcsstk10	0.03	1.3%	0.03	1.3%	1.3%	1.3%	1.66
1138_bus	0.02	0.0%	0.02	0.9%	1.0%	0.9%	1.91
bcsstk27	0.06	2.8%	0.06	3.0%	2.8%	2.8%	1.97
mhd1280b	0.01	0.4%	0.01	0.1%	0.4%	0.2%	1.14
plbuckle	0.00	2.6%	0.00	2.6%	2.6%	2.6%	1.65
msc01440	0.11	0.3%	0.11	0.3%	0.2%	0.2%	1.74
bcsstk11	0.04	3.6%	0.04	4.0%	3.8%	3.8%	1.64
bcsstm11	0.00	0.1%	0.00	0.1%	0.1%	0.1%	2.56
bcsstm12	0.05	0.0%	0.05	0.1%	0.1%	0.1%	1.64
bcsstk12	0.04	3.6%	0.04	4.0%	3.8%	3.8%	1.64
ex33	0.04	6.0%	0.05	2.0%	2.1%	2.1%	1.64
bcsstk14	0.11	0.0%	0.12	0.2%	0.2%	0.1%	1.49
ex3	0.06	0.0%	0.06	0.8%	0.8%	0.0%	1.59
nasa1824	0.11	0.2%	0.12	0.1%	0.1%	0.1%	1.31
plat1919	0.04	2.1%	0.04	1.4%	1.8%	1.4%	1.42
bcsstm26	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.49
bcsstk26	0.07	1.0%	0.08	0.3%	0.8%	0.8%	1.42
bcsstk13	0.22	1.6%	0.26	1.4%	2.7%	2.7%	1.39
nasa2146	0.07	2.3%	0.07	3.3%	3.2%	3.2%	1.70
ex10	0.02	0.9%	0.02	1.4%	1.3%	0.8%	1.28
Chem97ZtZ	0.00	0.0%	0.00	0.1%	0.1%	0.1%	0.50
ex10hs	0.02	0.3%	0.02	0.6%	0.6%	0.0%	1.27
ex13	0.04	0.0%	0.04	0.1%	0.1%	0.0%	1.26
nasa2910	0.17	0.0%	0.18	1.7%	3.6%	0.1%	1.38
bcsstk23	0.18	0.0%	0.18	0.0%	0.4%	0.4%	1.71
bcsstm23	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.58
mhd3200b	0.00	0.0%	0.00	1.9%	1.9%	1.8%	1.18
bibd_81_2	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.62

Continued on next page

Table 3.7 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
ex9	0.03	0.9%	0.03	1.6%	1.5%	0.6%	1.28
bcsstm24	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.63
bcsstk24	0.08	0.0%	0.11	0.0%	0.0%	0.0%	1.66
bcsstk21	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.25
bcsstm21	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.62
bcsstk15	0.10	2.7%	0.10	1.5%	1.3%	1.3%	1.36
sts4098	0.11	0.0%	0.19	3.7%	19.7%	18.9%	0.68
t2dal_e	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.64
bcsstk28	0.05	6.5%	0.05	4.0%	4.3%	4.3%	1.63
msc04515	0.04	1.7%	0.04	2.3%	2.6%	2.6%	1.20
nasa4704	0.08	0.1%	0.08	0.1%	0.1%	0.1%	1.23
mhd4800b	0.00	0.0%	0.00	4.3%	4.3%	4.3%	1.16
crystm01	0.03	1.4%	0.03	1.4%	1.4%	1.4%	1.24
bcsstk16	0.05	0.0%	0.05	0.0%	0.0%	0.0%	1.51
s3rmt3m3	0.06	0.0%	0.06	0.0%	0.0%	0.0%	1.54
s3rmt3m1	0.07	0.0%	0.07	0.1%	0.1%	0.1%	1.52
s2rmq4m1	0.07	0.3%	0.07	1.3%	1.2%	1.2%	1.55
s1rmt3m1	0.07	0.0%	0.07	0.1%	0.1%	0.1%	1.45
s1rmq4m1	0.07	1.2%	0.07	2.2%	2.0%	2.0%	1.54
s2rmt3m1	0.07	0.0%	0.07	0.0%	0.0%	0.0%	1.55
s3rmq4m1	0.07	0.4%	0.07	1.3%	1.2%	1.2%	1.54
ex15	0.01	0.3%	0.01	0.3%	0.3%	0.3%	1.21
Kuu	0.06	0.1%	0.06	3.6%	3.4%	0.8%	1.19
Muu	0.00	0.0%	0.00	0.0%	0.0%	0.0%	0.98
bcsstk38	0.03	3.1%	0.04	2.5%	2.1%	2.1%	1.17
aft01	0.03	0.0%	0.03	3.6%	3.6%	3.1%	1.17
fv1	0.02	0.0%	0.02	0.0%	0.0%	0.0%	1.42
fv3	0.02	0.0%	0.02	0.6%	0.6%	0.6%	1.42
fv2	0.02	0.0%	0.02	0.6%	0.6%	0.6%	1.42
bundle1	0.13	0.2%	0.10	3.4%	10.1%	1.9%	0.22
ted_B	0.00	0.0%	0.00	0.1%	0.1%	0.0%	0.64
ted_B_unscaled	0.00	0.0%	0.00	0.1%	0.1%	0.0%	0.64
msc10848	0.09	0.0%	0.09	3.8%	1.3%	0.0%	1.82
bcsstk17	0.04	0.0%	0.04	2.5%	2.5%	2.5%	1.30
t2dah_e	0.02	0.0%	0.02	1.0%	1.0%	0.7%	1.10
bcsstk18	0.04	0.0%	0.04	1.0%	0.9%	0.9%	1.00

Continued on next page

Table 3.7 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
cbuckle	0.05	1.2%	0.05	4.0%	2.4%	2.4%	1.33
crystm02	0.02	1.3%	0.02	1.0%	1.0%	1.0%	1.11
Pres_Poisson	0.03	0.5%	0.03	0.7%	0.7%	0.7%	1.41
bcsstm25	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.61
bcsstk25	0.02	4.4%	0.02	0.6%	0.8%	0.8%	1.05
Dubcova1	0.03	0.0%	0.02	1.6%	1.6%	1.0%	1.10
olafu	0.05	0.1%	0.04	0.5%	0.8%	0.8%	1.50
gyro_m	0.00	0.2%	0.00	0.4%	0.6%	0.4%	0.88
gyro	0.01	0.1%	0.01	0.3%	0.4%	0.1%	1.27
bodyy4	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.30
bodyy5	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.32
bodyy6	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.31
raefsky4	0.05	1.5%	0.05	1.8%	1.7%	1.7%	1.46
LFAT5000	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.14
LF10000	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.09
t3dl_e	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.65
msc23052	0.03	2.7%	0.03	2.5%	2.3%	2.3%	1.52
bcsstk36	0.03	1.0%	0.03	2.0%	1.8%	1.8%	1.44
crystm03	0.01	1.8%	0.01	0.7%	0.7%	0.7%	1.19
smt	0.07	0.3%	0.06	1.4%	1.1%	0.1%	1.87
thread	0.10	0.0%	0.10	0.3%	0.3%	0.0%	1.87
wathen100	0.02	0.0%	0.02	0.4%	0.4%	0.2%	0.99
ship_001	0.03	2.5%	0.03	1.0%	1.0%	0.7%	1.83
nd12k	0.31	1.0%	0.29	0.9%	0.9%	0.7%	2.58
wathen120	0.02	0.0%	0.01	0.5%	0.5%	0.3%	1.01
obstclae	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.29
jnlbrng1	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.28
minsurfo	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.30
bcsstm39	0.00	0.0%	0.00	0.0%	0.0%	0.0%	2.86
vanbody	0.02	1.5%	0.02	1.9%	1.8%	1.7%	1.42
gridgena	0.01	0.0%	0.01	1.0%	1.0%	1.0%	1.08
cvxbqp1	0.02	0.0%	0.02	0.0%	0.0%	0.0%	1.48
ct20stif	0.04	4.4%	0.04	3.4%	3.5%	3.3%	1.40
crankseg_1	0.04	0.0%	0.03	2.4%	1.9%	1.2%	1.94
nasasrb	0.01	0.5%	0.01	0.2%	0.2%	0.2%	1.46
Andrews	0.07	0.5%	0.10	1.5%	1.5%	1.5%	1.21

Continued on next page

Table 3.7 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
crankseg_2	0.04	2.0%	0.03	1.0%	0.1%	0.0%	1.97
Dubcova2	0.01	0.0%	0.01	0.1%	0.1%	0.0%	1.05
qa8fm	0.01	0.1%	0.01	0.0%	0.0%	0.0%	1.16
cfd1	0.02	0.8%	0.03	1.7%	1.7%	1.7%	1.14
nd24k	0.24	0.9%	0.23	1.7%	1.6%	0.0%	2.56
oilpan	0.02	1.0%	0.02	0.2%	0.2%	0.0%	1.42
finan512	0.00	0.0%	0.00	0.0%	0.0%	0.0%	0.81
apache1	0.02	0.0%	0.02	0.0%	0.0%	0.0%	1.48
shallow_water1	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.32
shallow_water2	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.32
thermal1	0.00	0.0%	0.00	0.2%	0.2%	0.2%	1.24
denormal	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.12
s3dkt3m2	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.36
s3dkq4m2	0.01	0.0%	0.01	0.7%	0.7%	0.7%	1.56
m_t1	0.02	4.0%	0.02	0.4%	0.7%	0.1%	1.92
2cubes_sphere	0.04	6.5%	0.04	1.9%	1.8%	0.0%	1.11
thermomech_TK	0.00	0.0%	0.00	0.9%	0.9%	0.9%	1.42
thermomech_TC	0.00	0.6%	0.00	0.3%	0.4%	0.3%	1.42
x104	0.03	0.0%	0.03	0.1%	0.3%	0.0%	1.82
shipsec8	0.03	3.0%	0.03	0.1%	0.3%	0.3%	1.52
ship_003	0.02	0.9%	0.02	2.3%	2.3%	2.3%	1.52
cfd2	0.02	1.1%	0.02	0.7%	0.9%	0.9%	1.11
boneS01	0.04	0.3%	0.03	1.5%	1.7%	1.4%	1.43
shipsec1	0.02	0.1%	0.02	0.3%	0.5%	0.5%	1.59
bmw7st_1	0.01	2.3%	0.01	0.4%	0.6%	0.5%	1.52
Dubcova3	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.04
bmwcra_1	0.01	3.5%	0.01	0.0%	0.0%	0.0%	1.42
G2_circuit	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.42
shipsec5	0.01	5.4%	0.02	0.7%	1.1%	1.1%	1.56
thermomech_dM	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.56
pwtk	0.01	0.1%	0.01	0.1%	0.1%	0.1%	1.56
hood	0.01	0.6%	0.01	0.7%	0.8%	0.6%	1.48
BenElechi1	0.01	0.0%	0.01	1.0%	1.0%	1.0%	1.57
offshore	0.02	4.3%	0.02	2.5%	2.4%	1.7%	1.17
F1	0.01	0.0%	0.01	1.0%	1.0%	0.0%	1.21
msdoor	0.00	0.5%	0.00	0.2%	0.2%	0.1%	1.48

Continued on next page

Table 3.7 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
af_2_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.27
af_5_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_1_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_4_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.27
af_3_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_0_k101	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
inline_1	0.01	2.1%	0.01	1.3%	1.3%	0.9%	1.26
af_shell8	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_shell3	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_shell4	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
af_shell7	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.28
parabolic_fem	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.67
apache2	0.01	0.0%	0.01	0.0%	0.0%	0.0%	1.47
tmt_sym	0.00	0.9%	0.00	0.0%	0.1%	0.1%	1.22
boneS10	0.01	3.1%	0.01	0.5%	0.5%	0.0%	1.46
ldoor	0.00	0.0%	0.00	0.3%	0.3%	0.3%	1.47
ecology2	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.29
thermal2	0.00	1.7%	0.00	0.2%	0.2%	0.2%	1.30
G3_circuit	0.00	0.0%	0.00	0.0%	0.0%	0.0%	1.39

Table 3.8: 2-way partitioning performance of the PD matrix collection for connectivity metric with node balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
msc01050	1.21	3.1%	1.29	0.0%	0.77
bcsstm08	1.00	0.0%	1.00	0.0%	2.31
bcsstm09	1.00	0.1%	1.00	0.1%	2.30
bcsstk09	1.11	5.4%	1.12	0.1%	1.71
bcsstk10	1.03	1.3%	1.03	1.3%	1.49
1138_bus	1.02	0.0%	1.02	1.4%	1.77
bcsstk27	1.07	2.7%	1.06	2.8%	1.78
mhd1280b	1.01	0.2%	1.01	0.1%	1.03

Continued on next page

Table 3.8 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
plbuckle	1.00	2.6%	1.00	2.6%	1.53
msc01440	1.11	0.2%	1.11	0.1%	1.49
bcsstk11	1.04	3.6%	1.04	3.7%	1.46
bcsstm11	1.00	0.1%	1.00	0.1%	2.38
bcsstm12	1.05	0.0%	1.05	0.1%	1.49
bcsstk12	1.04	3.6%	1.04	3.7%	1.46
ex33	1.04	3.4%	1.04	3.2%	1.48
bcsstk14	1.11	0.0%	1.11	0.2%	1.27
ex3	1.06	0.1%	1.06	0.9%	1.45
nasa1824	1.11	0.1%	1.13	0.1%	1.15
plat1919	1.04	4.3%	1.04	2.8%	1.31
bcsstm26	1.00	0.0%	1.00	0.0%	2.32
bcsstk26	1.07	1.5%	1.08	0.0%	1.26
bcsstk13	1.22	3.7%	1.29	1.8%	0.81
nasa2146	1.07	2.0%	1.07	2.3%	1.55
ex10	1.02	1.2%	1.02	1.1%	1.20
Chem97ZtZ	1.00	0.0%	1.00	0.0%	0.48
ex10hs	1.02	0.7%	1.02	0.0%	1.19
ex13	1.04	0.1%	1.04	0.9%	1.19
nasa2910	1.18	0.0%	1.16	3.2%	1.06
bcsstk23	1.18	0.0%	1.18	0.5%	1.43
bcsstm23	1.00	0.0%	1.00	0.0%	2.39
mhd3200b	1.00	0.0%	1.00	4.3%	1.08
bibd_81_2	1.00	0.0%	1.00	0.0%	2.41
ex9	1.03	0.5%	1.03	0.6%	1.20
bcsstm24	1.00	0.0%	1.00	0.0%	2.40
bcsstk24	1.08	0.0%	1.11	0.1%	1.37
bcsstk21	1.00	0.0%	1.00	0.0%	1.19
bcsstm21	1.00	0.0%	1.00	0.0%	2.42
bcsstk15	1.09	1.4%	1.09	3.5%	1.15
sts4098	1.10	0.3%	1.11	6.3%	0.46
t2dal_e	1.00	0.0%	1.00	0.0%	2.44
bcsstk28	1.05	5.7%	1.05	3.8%	1.51
msc04515	1.04	3.1%	1.04	3.8%	1.11
nasa4704	1.08	0.2%	1.08	0.3%	1.11
mhd4800b	1.00	0.0%	1.00	1.6%	1.09

Continued on next page

Table 3.8 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
crystm01	1.03	1.4%	1.03	1.4%	1.17
bcsstk16	1.05	0.0%	1.05	0.0%	1.39
s3rmt3m3	1.06	0.0%	1.06	0.0%	1.39
s3rmt3m1	1.07	0.0%	1.07	0.0%	1.38
s2rmq4m1	1.07	0.3%	1.07	0.7%	1.40
s1rmt3m1	1.07	0.0%	1.07	0.1%	1.39
s1rmq4m1	1.07	0.4%	1.07	1.2%	1.39
s2rmt3m1	1.07	0.0%	1.07	0.1%	1.37
s3rmq4m1	1.07	0.2%	1.07	1.0%	1.39
ex15	1.01	0.3%	1.01	0.2%	1.16
Kuu	1.07	0.0%	1.06	2.1%	1.10
Muu	1.00	0.0%	1.00	0.0%	0.97
bcsstk38	1.04	3.3%	1.04	2.0%	1.10
aft01	1.03	0.0%	1.03	2.6%	1.16
fv1	1.02	0.0%	1.02	0.0%	1.33
fv3	1.02	0.1%	1.02	1.0%	1.33
fv2	1.02	0.1%	1.02	1.0%	1.33
bundle1	1.14	0.1%	1.11	18.8%	0.17
ted_B	1.00	0.0%	1.00	0.0%	0.63
ted_B_unscaled	1.00	0.0%	1.00	0.1%	0.63
msc10848	1.09	0.0%	1.09	0.4%	1.51
bcsstk17	1.04	0.0%	1.04	1.0%	1.20
t2dah_e	1.02	0.0%	1.02	0.7%	1.06
bcsstk18	1.04	1.7%	1.04	0.3%	0.93
cbuckle	1.05	2.3%	1.05	2.4%	1.19
crystm02	1.02	1.3%	1.02	1.0%	1.08
Pres_Poisson	1.03	0.8%	1.03	0.7%	1.35
bcsstm25	1.00	0.0%	1.00	0.0%	2.36
bcsstk25	1.02	1.4%	1.02	0.8%	1.04
Dubcova1	1.03	0.0%	1.02	1.0%	1.10
olafu	1.05	0.1%	1.05	0.7%	1.34
gyro_m	1.00	0.0%	1.00	0.1%	0.86
gyro	1.01	0.0%	1.01	0.7%	1.25
bodyy4	1.01	0.0%	1.01	0.0%	1.22
bodyy5	1.01	0.0%	1.01	0.0%	1.22
bodyy6	1.01	0.0%	1.01	0.0%	1.24

Continued on next page

Table 3.8 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
raefsky4	1.05	1.0%	1.05	1.4%	1.33
LFAT5000	1.00	0.0%	1.00	0.0%	1.06
LF10000	1.00	0.0%	1.00	0.0%	1.03
t3dl_e	1.00	0.0%	1.00	0.0%	2.41
msc23052	1.03	1.7%	1.03	2.4%	1.44
bcsstk36	1.03	2.5%	1.03	1.8%	1.36
crystm03	1.01	1.6%	1.01	0.7%	1.15
smt	1.07	2.8%	1.06	0.9%	1.63
thread	1.10	0.4%	1.10	1.0%	1.40
wathen100	1.02	0.0%	1.02	0.7%	0.97
ship_001	1.03	2.0%	1.03	0.7%	1.71
nd12k	1.30	1.2%	1.29	2.1%	0.99
wathen120	1.02	0.0%	1.01	0.6%	0.97
obstclae	1.01	0.0%	1.01	0.0%	1.23
jnlbrng1	1.01	0.0%	1.01	0.0%	1.22
minsurfo	1.01	0.0%	1.01	0.0%	1.23
bcsstm39	1.00	0.0%	1.00	0.0%	2.60
vanbody	1.02	2.1%	1.02	1.7%	1.37
gridgena	1.01	0.0%	1.01	0.1%	1.03
cvxbqp1	1.02	0.0%	1.02	0.0%	1.40
ct20stif	1.04	2.3%	1.04	3.0%	1.27
crankseg_1	1.04	0.0%	1.03	1.8%	1.71
nasasrb	1.01	1.6%	1.01	0.2%	1.41
Andrews	1.07	1.8%	1.11	2.8%	1.13
crankseg_2	1.04	2.0%	1.03	0.5%	1.68
Dubcova2	1.01	0.0%	1.01	0.7%	1.02
qa8fm	1.01	0.3%	1.01	0.0%	1.14
cfd1	1.02	0.2%	1.03	1.7%	1.08
nd24k	1.24	2.6%	1.23	1.1%	1.13
oilpan	1.02	0.0%	1.02	0.1%	1.35
finan512	1.00	0.0%	1.00	0.0%	0.79
apache1	1.02	0.0%	1.02	0.0%	1.38
shallow_water1	1.01	0.0%	1.01	0.0%	1.28
shallow_water2	1.01	0.0%	1.01	0.0%	1.26
thermal1	1.00	0.9%	1.00	0.8%	1.19
denormal	1.01	0.0%	1.01	0.0%	1.09

Continued on next page

Table 3.8 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
s3dkt3m2	1.01	0.0%	1.01	0.0%	1.29
s3dkq4m2	1.01	0.0%	1.01	0.2%	1.48
m_t1	1.03	1.5%	1.02	0.9%	1.81
2cubes_sphere	1.04	6.4%	1.04	0.7%	1.05
thermomech_TK	1.00	1.6%	1.00	0.7%	1.38
thermomech_TC	1.00	1.5%	1.00	0.1%	1.39
x104	1.03	0.0%	1.03	0.9%	1.69
shipsec8	1.03	0.1%	1.03	0.4%	1.41
ship_003	1.02	0.7%	1.02	1.2%	1.44
cfd2	1.02	0.6%	1.02	0.6%	1.08
boneS01	1.04	0.0%	1.04	0.8%	1.30
shipsec1	1.02	0.2%	1.02	0.4%	1.52
bmw7st_1	1.01	1.2%	1.01	0.7%	1.46
Dubcova3	1.01	0.0%	1.01	0.4%	1.02
bmwcra_1	1.01	3.5%	1.01	0.2%	1.36
G2_circuit	1.01	0.0%	1.00	0.4%	1.36
shipsec5	1.01	4.1%	1.02	0.7%	1.48
thermomech_dM	1.00	0.0%	1.00	0.0%	1.50
pwtk	1.01	0.2%	1.01	0.1%	1.51
hood	1.01	0.4%	1.01	0.3%	1.42
BenElechi1	1.01	0.0%	1.01	0.2%	1.52
offshore	1.02	2.9%	1.02	1.6%	1.18
F1	1.01	0.0%	1.01	0.5%	1.16
msdoor	1.00	0.3%	1.00	0.3%	1.43
af_2_k101	1.00	0.0%	1.00	0.0%	1.24
af_5_k101	1.00	0.0%	1.00	0.0%	1.23
af_1_k101	1.00	0.0%	1.00	0.0%	1.24
af_4_k101	1.00	0.0%	1.00	0.0%	1.23
af_3_k101	1.00	0.0%	1.00	0.0%	1.23
af_0_k101	1.00	0.0%	1.00	0.0%	1.23
inline_1	1.01	0.0%	1.01	0.9%	1.22
af_shell8	1.00	0.0%	1.00	0.0%	1.24
af_shell3	1.00	0.0%	1.00	0.0%	1.23
af_shell4	1.00	0.0%	1.00	0.0%	1.24
af_shell7	1.00	0.0%	1.00	0.0%	1.23
parabolic_fem	1.00	0.0%	1.00	0.0%	1.60

Continued on next page

Table 3.8 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
apache2	1.01	0.0%	1.01	0.0%	1.43
tmt_sym	1.00	0.9%	1.00	0.1%	1.18
boneS10	1.01	3.0%	1.01	0.4%	1.42
ldoor	1.00	0.4%	1.00	0.4%	1.44
ecology2	1.00	0.0%	1.00	0.0%	1.24
thermal2	1.00	1.6%	1.00	0.3%	1.25
G3_circuit	1.00	0.0%	1.00	0.0%	1.36

Table 3.9: 64-way partitioning performance of the LP matrix collection for cut-net metric with net balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
deter4	0.33	7.0%	0.85	338.5%	1.60
lpl2	0.12	13.1%	0.15	14.0%	3.93
model7	0.28	37.9%	0.32	36.1%	3.25
sctap1-2c	0.32	16.9%	0.39	22.9%	0.78
lp_cre_a	0.09	6.0%	0.09	18.6%	2.22
lpi_ceria3d	0.54	24.5%	0.98	189.6%	0.32
ch	0.17	16.9%	0.30	12.9%	3.02
aircraft	0.00	2460.0%	1.00	1180.0%	0.12
lpi_gosh	0.24	17.9%	0.33	41.7%	5.29
deter8	0.20	11.0%	0.22	14.3%	2.33
fxm2-16	0.22	19.8%	0.20	26.9%	2.26
nemsemm1	0.53	55.5%	0.80	204.3%	98.13
pcb3000	0.22	8.4%	0.22	13.0%	4.93
pgp2	0.57	13.7%	0.99	52.3%	0.53
rlfddd	0.83	310.3%	0.84	290.7%	13.01
deter6	0.21	4.5%	0.22	12.9%	2.32
large	0.17	11.1%	0.18	14.7%	2.35
lp_osa_30	0.02	0.4%	0.02	3.4%	3.26
stormg2-8	0.25	28.2%	0.25	25.8%	2.79
model10	0.44	68.9%	0.65	97.9%	6.38

Continued on next page

Table 3.9 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
nsir	0.28	17.2%	0.58	35.0%	5.57
seymourl	0.86	362.4%	0.93	279.1%	1.07
cq5	0.15	12.8%	0.16	22.7%	4.47
p05	0.18	10.8%	0.22	15.8%	8.00
deter5	0.20	5.5%	0.21	13.4%	2.35
scsd8-2b	0.52	18.4%	0.96	169.3%	1.76
r05	0.20	9.2%	0.24	18.0%	9.84
bas1lp	0.65	1289.7%	0.86	296.0%	3.84
deter1	0.20	7.6%	0.20	16.4%	2.33
co5	0.14	13.4%	0.16	21.9%	4.50
stat96v1	0.18	10.5%	0.17	10.2%	59.23
lp_dfl001	0.37	17.3%	0.47	14.1%	3.25
deter2	0.22	5.9%	0.24	20.2%	1.79
fxm3_6	0.09	9.2%	0.08	14.0%	2.93
deter7	0.16	9.2%	0.15	12.6%	2.60
lp_cre_d	0.23	11.5%	0.48	47.2%	13.96
ulevimin	0.17	7.6%	0.24	18.3%	5.77
nemswrld	0.30	7.3%	0.58	68.7%	6.25
nemsemm2	0.16	13.9%	0.17	21.0%	12.29
nl	0.12	7.1%	0.15	17.9%	3.03
lp_cre_b	0.20	31.8%	0.36	34.6%	11.63
deter3	0.16	2.4%	0.16	15.9%	2.31
rlfdual	0.81	184.1%	0.79	184.7%	5.53
scsd8-2r	0.50	16.1%	0.98	205.9%	1.18
cq9	0.11	13.7%	0.16	19.9%	4.97
pf2177	0.81	36.3%	0.98	119.3%	1.97
scagr7-2b	0.26	126.6%	0.86	118.5%	0.26
lp_pds_06	0.14	8.9%	0.17	21.3%	3.25
p010	0.11	9.9%	0.16	15.9%	1.11
ge	0.11	10.8%	0.12	19.3%	2.08
lp_osa_60	0.01	0.4%	0.01	10.1%	2.29
co9	0.11	11.7%	0.23	21.0%	5.24
lpl3	0.04	3.5%	0.10	15.2%	5.71
fome11	0.33	18.9%	0.43	16.7%	3.47
scrs8-2r	0.75	798.7%	0.99	321.9%	0.14
stormg2-27	0.15	7.9%	0.15	20.0%	2.71

Continued on next page

Table 3.9 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
lp_ken_11	0.02	7.8%	0.02	21.6%	2.58
sctap1-2b	0.30	7.5%	0.83	84.1%	0.33
car4	0.02	2.6%	0.05	5.0%	0.41
lp_pds_10	0.14	8.9%	0.18	20.5%	3.16
lp_stocfor3	0.05	4.6%	0.04	8.2%	1.65
ex3sta1	0.43	16.8%	0.44	30.7%	1.42
testbig	0.23	12.3%	0.88	262.1%	0.26
dbir1	0.17	12.6%	0.84	23.0%	8.76
dbir2	0.17	11.6%	0.86	24.4%	7.94
scfxm1-2b	0.06	4.8%	0.06	10.6%	1.39
route	0.18	8.5%	0.20	17.6%	5.17
ts-palko	0.60	139.1%	0.99	290.2%	8.47
fxm4_6	0.05	4.9%	0.04	16.0%	2.90
fome12	0.24	14.6%	0.40	15.1%	3.60
e18	0.62	12.6%	0.85	14.9%	3.64
pltexpa	0.15	4.0%	0.21	20.7%	2.35
baxter	0.28	27.3%	0.43	47.4%	0.87
lp_ken_13	0.02	5.1%	0.02	15.4%	2.92
stat96v2	0.07	3.0%	0.07	8.1%	70.20
lp_pds_20	0.14	9.0%	0.18	17.8%	3.81
stat96v3	0.06	2.4%	0.07	8.3%	45.83
world	0.11	9.4%	0.16	15.6%	2.27
mod2	0.12	6.5%	0.16	16.5%	2.14
sc205-2r	0.23	10.1%	0.97	65.6%	0.15
scfxm1-2r	0.06	4.3%	0.24	17.3%	1.25
fxm3_16	0.05	5.1%	0.05	7.9%	2.89
dbic1	0.04	2.7%	0.04	13.6%	9.43
fome13	0.20	13.2%	0.38	15.7%	3.59
pds-30	0.13	10.0%	0.16	15.9%	3.44
rlfprim	0.63	56.5%	0.80	173.6%	0.45
stormg2-125	0.13	8.0%	0.29	44.2%	1.90
pds-40	0.12	8.3%	0.17	16.9%	3.92
fome21	0.12	8.4%	0.16	15.1%	3.85
pds-50	0.12	9.0%	0.16	15.3%	3.70
pds-60	0.12	10.1%	0.16	15.3%	3.82
pds-70	0.11	6.5%	0.16	15.7%	3.94

Continued on next page

Table 3.9 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
pds-80	0.11	6.2%	0.15	15.1%	4.03
pds-90	0.10	6.7%	0.14	13.6%	3.79
pds-100	0.10	7.9%	0.13	14.7%	4.04
watson_1	0.02	1.0%	0.01	4.2%	3.95
sgpf5y6	0.03	7.2%	0.03	25.8%	1.97
watson_2	0.01	3.6%	0.01	16.4%	3.80
stormG2_1000	0.13	11.9%	0.54	113.3%	0.57
cont11_l	0.01	3.2%	0.01	5.9%	1.21

Table 3.10: 64-way partitioning performance of the PD matrix collection for cut-net metric with node balancing.

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
bcsstk13	0.94	1.8%	0.98	375.6%	664.8%	78.3%	3.06
Chem97ZtZ	0.20	3.5%	0.20	79.2%	55.2%	33.9%	0.93
mhd3200b	0.09	1.2%	0.08	13.4%	10.1%	8.1%	2.46
bibd_81_2	0.00	0.7%	0.00	0.7%	0.7%	0.7%	4.19
ex9	0.81	1.3%	0.81	222.1%	99.3%	31.1%	2.02
bcsstm24	0.00	0.6%	0.00	0.6%	0.6%	0.6%	4.14
bcsstk24	0.87	1.5%	0.85	307.7%	191.4%	98.9%	2.73
bcsstm21	0.00	1.3%	0.00	1.3%	1.3%	1.3%	4.19
bcsstk21	0.58	2.0%	0.61	34.4%	33.1%	25.6%	1.89
bcsstk15	0.92	2.7%	0.94	401.2%	157.2%	36.0%	1.94
sts4098	0.67	11.3%	0.72	156.6%	271.8%	175.8%	1.18
t2dal_e	0.00	0.7%	0.00	0.7%	0.7%	0.7%	4.23
bcsstk28	0.86	2.3%	0.80	225.7%	121.0%	69.8%	2.81
msc04515	0.61	2.8%	0.59	70.9%	61.4%	57.2%	1.96
nasa4704	0.65	2.8%	0.67	113.0%	82.1%	62.0%	1.86
mhd4800b	0.06	1.6%	0.06	10.1%	8.7%	7.4%	2.38
crystm01	0.72	4.7%	0.74	114.6%	59.7%	45.9%	1.97
bcsstk16	0.91	2.2%	0.91	420.4%	214.2%	76.9%	2.87
s3rmt3m3	0.73	1.5%	0.69	92.7%	83.6%	61.4%	2.53

Continued on next page

Table 3.10 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
s2rmq4m1	0.83	2.3%	0.78	121.7%	72.0%	58.1%	2.93
s1rmt3m1	0.77	1.4%	0.75	93.8%	67.6%	45.1%	2.53
s1rmq4m1	0.83	2.4%	0.77	134.6%	72.0%	58.9%	2.91
s3rmt3m1	0.77	1.4%	0.75	94.5%	73.7%	50.2%	2.52
s3rmq4m1	0.83	1.9%	0.77	130.1%	75.6%	61.0%	2.92
s2rmt3m1	0.77	1.4%	0.75	93.1%	75.0%	53.0%	2.53
ex15	0.40	2.9%	0.39	42.1%	32.4%	31.2%	2.25
Muu	0.59	0.9%	0.60	84.1%	52.2%	5.8%	1.60
Kuu	0.77	0.9%	0.79	248.7%	112.5%	14.5%	2.17
bcsstk38	0.72	2.5%	0.73	165.1%	111.9%	68.5%	2.08
aft01	0.45	0.8%	0.43	27.5%	27.7%	16.8%	1.69
fv1	0.29	3.3%	0.28	14.5%	14.6%	14.4%	2.43
fv3	0.29	3.2%	0.28	11.6%	12.8%	12.6%	2.42
fv2	0.29	3.1%	0.28	12.2%	12.9%	12.8%	2.41
bundle1	1.00	2.9%	1.00	1458.9%	974.9%	258.9%	0.31
ted_B	0.18	1.2%	0.16	21.8%	51.2%	35.1%	1.26
ted_B_unscaled	0.18	1.1%	0.16	20.3%	47.7%	31.8%	1.25
msc10848	0.88	0.2%	0.91	592.1%	238.4%	73.4%	3.64
bcsstk17	0.61	3.2%	0.62	86.9%	53.3%	44.7%	2.47
t2dah_e	0.39	0.9%	0.37	15.5%	17.6%	9.3%	1.89
bcsstk18	0.44	4.9%	0.47	80.7%	56.4%	42.7%	1.50
cbuckle	0.60	2.9%	0.56	36.0%	32.5%	31.4%	2.81
crystm02	0.54	4.9%	0.54	49.6%	52.2%	49.9%	1.89
Pres_Poisson	0.60	3.6%	0.57	42.2%	32.1%	31.0%	2.81
bcsstk25	0.54	2.2%	0.57	50.0%	37.0%	28.4%	1.62
bcsstm25	0.00	0.3%	0.00	0.3%	0.3%	0.3%	4.09
Dubcova1	0.36	0.3%	0.32	15.2%	13.6%	2.9%	1.81
olafu	0.60	3.0%	0.57	106.8%	66.4%	56.0%	3.05
gyro	0.57	1.1%	0.60	301.4%	118.4%	57.0%	2.09
gyro_m	0.31	2.6%	0.30	61.5%	55.1%	39.7%	1.37
bodyy4	0.20	1.7%	0.20	9.1%	9.2%	8.8%	2.18
bodyy5	0.20	1.5%	0.20	8.6%	8.5%	8.1%	2.20
bodyy6	0.20	1.3%	0.20	8.2%	7.8%	7.4%	2.26
raefsky4	0.69	3.6%	0.68	138.1%	61.1%	51.4%	2.77
LFAT5000	0.01	0.1%	0.01	0.9%	0.8%	0.8%	1.89
LF10000	0.01	0.5%	0.01	0.7%	1.0%	1.0%	1.85

Continued on next page

Table 3.10 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
t3dl_e	0.00	0.3%	0.00	0.3%	0.3%	0.3%	4.14
bcsstk36	0.49	3.2%	0.47	35.2%	34.1%	31.7%	2.70
msc23052	0.49	2.4%	0.47	36.0%	32.4%	29.9%	2.90
crystm03	0.45	4.8%	0.43	24.6%	36.2%	35.8%	1.85
smt	0.85	0.1%	0.87	394.0%	136.6%	50.4%	3.83
thread	0.86	0.3%	0.89	313.3%	112.3%	38.5%	3.72
wathen100	0.26	0.2%	0.22	6.5%	8.7%	4.3%	1.61
ship_001	0.79	1.2%	0.81	367.6%	107.1%	41.9%	3.72
nd12k	0.99	0.3%	1.00	459.7%	461.2%	2.2%	5.13
wathen120	0.24	0.3%	0.20	8.6%	10.5%	6.6%	1.59
obstclae	0.12	1.5%	0.12	8.0%	8.6%	8.5%	2.06
jnlbrng1	0.12	1.4%	0.12	8.9%	9.3%	9.3%	2.06
minsurfo	0.12	1.4%	0.12	8.6%	9.4%	9.4%	2.06
bcsstm39	0.00	0.0%	0.00	0.0%	0.0%	0.0%	4.22
vanbody	0.33	3.6%	0.32	32.6%	30.4%	28.8%	2.58
gridgena	0.17	1.8%	0.16	10.0%	12.7%	12.3%	1.76
cvxbqp1	0.13	2.3%	0.16	11.3%	12.6%	12.3%	2.03
ct20stif	0.37	3.0%	0.35	30.4%	48.1%	45.5%	2.50
crankseg_1	0.83	0.0%	0.86	522.3%	120.2%	23.3%	4.02
nasasrb	0.34	3.0%	0.31	14.9%	17.8%	17.6%	2.72
Andrews	0.65	5.0%	0.70	66.3%	53.6%	31.2%	1.50
crankseg_2	0.80	0.2%	0.84	482.2%	123.2%	32.1%	4.19
Dubcova2	0.19	0.0%	0.16	10.7%	6.0%	0.4%	1.65
qa8fm	0.42	4.3%	0.40	15.5%	27.8%	27.5%	1.73
cfd1	0.40	5.0%	0.39	22.7%	27.3%	27.1%	1.69
nd24k	0.97	0.8%	0.99	600.6%	273.2%	8.7%	5.54
oilpan	0.29	1.8%	0.28	15.0%	16.5%	13.5%	2.62
finan512	0.16	1.4%	0.12	5.0%	5.3%	1.8%	1.14
apache1	0.24	1.8%	0.23	12.2%	13.2%	13.1%	2.02
shallow_water2	0.08	0.7%	0.08	7.3%	6.9%	6.8%	1.98
shallow_water1	0.08	1.2%	0.08	7.2%	6.8%	6.7%	1.98
thermal1	0.09	2.3%	0.09	10.9%	10.8%	10.8%	1.91
denormal	0.17	1.8%	0.15	9.7%	10.9%	10.9%	1.78
s3dkt3m2	0.23	0.6%	0.22	6.8%	8.6%	8.1%	2.57
s3dkq4m2	0.26	1.9%	0.22	10.4%	11.6%	11.6%	3.00
m_t1	0.42	1.4%	0.39	25.8%	29.1%	20.0%	3.88

Continued on next page

Table 3.10 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
2cubes_sphere	0.43	1.4%	0.44	18.5%	17.6%	9.0%	1.54
thermomech_TK	0.08	2.4%	0.08	12.3%	12.0%	11.9%	2.07
thermomech_TC	0.08	2.2%	0.08	12.8%	12.7%	12.7%	2.07
x104	0.34	1.1%	0.33	29.4%	31.2%	22.8%	3.63
shipsec8	0.32	2.6%	0.31	22.5%	23.1%	22.5%	2.71
ship_003	0.41	2.4%	0.40	22.4%	23.8%	22.4%	2.66
cfd2	0.29	3.7%	0.27	19.7%	21.0%	20.9%	1.66
boneS01	0.33	2.5%	0.31	17.4%	27.3%	24.2%	2.47
shipsec1	0.27	2.5%	0.26	15.5%	19.1%	18.9%	2.83
bmw7st_1	0.20	3.6%	0.19	18.9%	21.4%	21.2%	2.75
Dubcova3	0.18	0.1%	0.14	10.3%	8.6%	1.5%	1.54
bmwcra_1	0.39	4.8%	0.38	27.7%	34.9%	34.3%	2.44
G2_circuit	0.10	1.0%	0.10	9.0%	9.1%	9.0%	1.81
shipsec5	0.25	3.2%	0.24	18.9%	18.7%	18.6%	2.81
thermomech_dM	0.05	1.9%	0.05	11.0%	11.1%	11.1%	2.13
pwtk	0.18	1.7%	0.16	10.7%	9.5%	9.5%	2.99
hood	0.14	1.6%	0.14	15.3%	14.9%	13.3%	2.67
BenElechi1	0.15	1.4%	0.13	10.9%	11.8%	11.8%	2.96
offshore	0.23	3.3%	0.23	17.2%	18.6%	14.5%	1.51
F1	0.31	0.4%	0.31	25.2%	28.1%	15.3%	1.84
msdoor	0.08	2.7%	0.08	12.0%	13.1%	12.7%	2.70
af_2_k101	0.09	0.5%	0.09	5.6%	5.8%	5.8%	2.19
af_3_k101	0.09	0.6%	0.09	5.1%	6.0%	5.9%	2.20
af_5_k101	0.09	1.0%	0.09	5.6%	6.3%	6.2%	2.17
af_0_k101	0.09	0.4%	0.09	5.9%	6.0%	5.9%	2.17
af_1_k101	0.09	0.7%	0.09	6.2%	6.7%	6.7%	2.20
af_4_k101	0.09	0.5%	0.09	5.7%	6.2%	6.1%	2.17
inline_1	0.18	1.0%	0.17	15.8%	18.5%	13.0%	2.04
af_shell4	0.09	0.7%	0.08	6.3%	6.3%	6.3%	2.18
af_shell7	0.09	1.0%	0.08	5.8%	6.2%	6.2%	2.19
af_shell8	0.09	0.9%	0.08	6.3%	6.9%	6.8%	2.18
af_shell3	0.09	0.7%	0.08	5.5%	6.4%	6.3%	2.18
parabolic_fem	0.04	0.2%	0.04	6.8%	7.0%	6.9%	2.05
apache2	0.09	0.3%	0.08	7.7%	8.1%	8.1%	1.74
tmt_sym	0.03	2.2%	0.03	9.6%	9.6%	9.5%	1.63
boneS10	0.08	3.3%	0.08	16.5%	16.1%	15.8%	2.50

Continued on next page

Table 3.10 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
ldoor	0.06	2.2%	0.06	11.3%	11.2%	10.9%	2.67
ecology2	0.03	0.0%	0.03	4.7%	4.8%	4.8%	1.59
thermal2	0.02	2.2%	0.02	7.6%	7.7%	7.7%	1.79
G3_circuit	0.03	0.6%	0.02	6.5%	6.3%	6.3%	1.67

Table 3.11: 64-way partitioning performance of the PD matrix collection for connectivity metric with node balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
bcsstk13	5.23	4.2%	5.01	111.7%	0.61
Chem97ZtZ	1.21	3.5%	1.22	42.1%	0.85
mhd3200b	1.10	1.1%	1.08	5.9%	2.12
bibd_81_2	1.00	0.7%	1.00	0.7%	3.28
ex9	2.41	1.8%	2.39	42.3%	1.14
bcsstm24	1.00	0.6%	1.00	0.6%	3.26
bcsstk24	2.69	3.3%	2.56	46.5%	1.31
bcsstm21	1.00	1.3%	1.00	1.3%	3.30
bcsstk21	1.75	1.9%	1.79	9.2%	1.69
bcsstk15	2.75	4.1%	2.75	35.6%	1.06
sts4098	2.54	11.3%	2.71	111.2%	0.43
t2dal_e	1.00	0.7%	1.00	0.7%	3.30
bcsstk28	2.50	3.8%	2.35	50.2%	1.48
msc04515	1.84	3.8%	1.83	23.9%	1.51
nasa4704	2.10	3.6%	2.11	38.0%	1.24
mhd4800b	1.06	1.5%	1.06	6.0%	2.04
crystm01	2.10	4.5%	2.10	33.7%	1.34
bcsstk16	3.24	4.3%	3.25	61.8%	0.95
s3rmt3m3	2.07	2.7%	2.04	31.4%	1.57
s2rmq4m1	2.37	3.2%	2.28	35.3%	1.60
s1rmt3m1	2.13	2.4%	2.10	31.4%	1.57
s1rmq4m1	2.38	3.0%	2.26	32.5%	1.62
s3rmt3m1	2.14	2.7%	2.10	34.0%	1.57

Continued on next page

Table 3.11 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
s3rmq4m1	2.38	2.9%	2.26	32.2%	1.62
s2rmt3m1	2.14	2.3%	2.11	28.7%	1.57
ex15	1.48	3.1%	1.50	15.7%	1.80
Muu	1.84	1.1%	1.82	27.0%	1.13
Kuu	2.35	1.1%	2.31	44.4%	1.08
bcsstk38	2.29	3.9%	2.26	72.5%	1.06
aft01	1.56	1.2%	1.53	12.3%	1.48
fv1	1.33	3.7%	1.33	7.6%	2.04
fv3	1.33	3.1%	1.32	8.6%	2.03
fv2	1.33	3.2%	1.33	8.6%	2.03
bundle1	6.16	7.0%	4.49	275.0%	0.17
ted_B	1.20	1.5%	1.16	30.4%	1.11
ted_B_unscaled	1.20	1.4%	1.17	32.5%	1.11
msc10848	3.05	1.2%	2.74	73.4%	1.14
bcsstk17	1.87	4.3%	1.82	27.1%	1.58
t2dah_e	1.47	0.9%	1.44	8.7%	1.48
bcsstk18	1.64	5.1%	1.66	34.8%	1.14
cbuckle	1.79	3.2%	1.76	15.5%	1.73
crystm02	1.72	4.6%	1.70	17.3%	1.41
Pres_Poisson	1.80	3.2%	1.77	16.3%	1.82
bcsstk25	1.77	2.6%	1.79	13.0%	1.17
bcsstm25	1.00	0.3%	1.00	0.3%	3.26
Dubcova1	1.42	0.4%	1.37	7.7%	1.48
olafu	1.86	3.5%	1.79	29.0%	1.75
gyro	1.91	1.8%	1.82	75.3%	1.00
gyro_m	1.34	2.7%	1.30	30.4%	1.10
bodyy4	1.22	1.9%	1.22	6.2%	1.92
bodyy5	1.22	1.6%	1.22	6.2%	1.93
bodyy6	1.22	1.5%	1.22	7.1%	1.91
raefsky4	2.16	4.5%	2.11	29.9%	1.43
LFAT5000	1.01	0.2%	1.01	0.3%	1.60
LF10000	1.01	0.5%	1.01	0.5%	1.57
t3dl_e	1.00	0.3%	1.00	0.3%	3.33
bcsstk36	1.65	3.8%	1.63	18.8%	1.85
msc23052	1.65	3.9%	1.63	16.9%	1.95
crystm03	1.55	4.1%	1.54	16.3%	1.45

Continued on next page

Table 3.11 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
smt	2.86	1.0%	2.67	57.8%	1.11
thread	3.09	1.0%	2.90	40.7%	1.14
wathen100	1.29	0.8%	1.25	4.9%	1.37
ship_001	2.30	1.3%	2.23	31.8%	1.51
nd12k	5.33	4.2%	5.69	64.5%	0.57
wathen120	1.26	0.8%	1.23	4.8%	1.40
obstclae	1.13	1.6%	1.12	6.9%	1.79
jnlbrng1	1.13	1.8%	1.12	5.9%	1.78
minsurfo	1.13	1.7%	1.12	6.4%	1.78
bcsstm39	1.00	0.0%	1.00	0.0%	3.40
vanbody	1.40	3.8%	1.38	22.3%	1.96
gridgena	1.18	1.8%	1.17	9.2%	1.50
cvxbqp1	1.16	2.2%	1.18	8.7%	1.77
ct20stif	1.48	4.0%	1.45	22.5%	1.80
crankseg_1	2.70	0.2%	2.45	39.5%	1.10
nasasrb	1.39	2.8%	1.38	8.3%	2.10
Andrews	2.06	6.8%	2.24	30.7%	1.11
crankseg_2	2.65	0.1%	2.39	51.6%	1.14
Dubcova2	1.21	0.2%	1.17	4.7%	1.46
qa8fm	1.56	3.6%	1.54	13.9%	1.38
cfd1	1.50	4.7%	1.50	17.6%	1.38
nd24k	4.08	3.7%	4.16	52.2%	0.80
oilpan	1.32	2.0%	1.32	10.3%	2.11
finan512	1.18	1.5%	1.13	4.3%	1.05
apache1	1.26	1.7%	1.25	10.3%	1.73
shallow_water2	1.08	0.9%	1.08	8.9%	1.74
shallow_water1	1.08	0.7%	1.08	8.3%	1.74
thermal1	1.10	2.2%	1.09	10.4%	1.68
denormal	1.18	1.5%	1.17	7.7%	1.53
s3dkt3m2	1.24	0.6%	1.25	4.5%	2.07
s3dkq4m2	1.28	1.5%	1.25	9.5%	2.44
m_t1	1.52	1.5%	1.46	12.0%	2.73
2cubes_sphere	1.55	2.3%	1.57	8.4%	1.22
thermomech_TK	1.08	2.3%	1.08	11.1%	1.83
thermomech_TC	1.08	2.0%	1.08	11.9%	1.83
x104	1.43	1.0%	1.39	12.1%	2.63

Continued on next page

Table 3.11 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
shipsec8	1.38	3.1%	1.36	13.4%	2.06
ship_003	1.51	3.1%	1.48	16.3%	1.90
cfd2	1.34	3.4%	1.32	14.8%	1.40
boneS01	1.40	3.0%	1.37	13.9%	1.85
shipsec1	1.31	2.5%	1.30	12.5%	2.26
bmw7st_1	1.22	3.7%	1.20	15.5%	2.28
Dubcova3	1.19	0.1%	1.15	4.1%	1.37
bmwcra_1	1.48	5.5%	1.49	22.3%	1.73
G2_circuit	1.10	1.3%	1.10	8.4%	1.60
shipsec5	1.28	3.5%	1.27	13.8%	2.26
thermomech_dM	1.06	2.0%	1.05	11.2%	1.90
pwtk	1.19	2.0%	1.18	9.9%	2.53
hood	1.15	1.8%	1.15	10.8%	2.28
BenElechi1	1.16	1.2%	1.14	9.9%	2.55
offshore	1.28	4.1%	1.28	11.5%	1.32
F1	1.40	0.7%	1.40	14.6%	1.33
msdoor	1.08	2.2%	1.09	12.9%	2.39
af_2_k101	1.09	0.3%	1.09	5.7%	1.92
af_3_k101	1.09	0.4%	1.09	5.8%	1.92
af_5_k101	1.09	0.6%	1.09	5.4%	1.94
af_0_k101	1.09	0.6%	1.09	6.4%	1.92
af_1_k101	1.09	0.3%	1.09	6.4%	1.93
af_4_k101	1.09	0.3%	1.09	6.1%	1.92
inline_1	1.19	1.2%	1.18	11.0%	1.73
af_shell4	1.09	0.3%	1.09	6.2%	1.94
af_shell7	1.09	0.3%	1.09	5.9%	1.93
af_shell8	1.09	0.2%	1.09	6.2%	1.93
af_shell3	1.09	0.6%	1.09	5.6%	1.93
parabolic_fem	1.04	0.1%	1.04	6.9%	1.87
apache2	1.09	0.3%	1.09	7.8%	1.58
tmt_sym	1.03	2.3%	1.03	8.6%	1.46
boneS10	1.09	3.7%	1.08	15.0%	2.28
ldoor	1.06	2.2%	1.06	11.7%	2.43
ecology2	1.03	0.0%	1.03	5.1%	1.44
thermal2	1.03	2.3%	1.02	8.9%	1.65
G3_circuit	1.03	0.3%	1.03	6.2%	1.53

Table 3.12: 128-way partitioning performance of the LP matrix collection for cut-net metric with net balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
lp_cre_d	0.27	24.3%	0.50	59.1%	12.01
ulevimin	0.23	28.1%	0.30	28.9%	5.76
nemswrld	0.34	36.3%	0.61	81.3%	6.06
nemsemm2	0.21	14.3%	0.24	26.9%	10.63
nl	0.17	13.5%	0.20	24.9%	3.09
lp_cre_b	0.25	33.5%	0.41	59.4%	10.32
deter3	0.21	8.1%	0.22	20.8%	2.28
rlfdual	0.87	276.1%	0.90	290.7%	5.57
scsd8-2r	0.53	21.8%	0.98	268.3%	1.33
cq9	0.18	21.3%	0.21	32.2%	4.67
pf2177	0.90	104.8%	0.98	157.7%	2.16
scagr7-2b	0.29	153.3%	0.87	147.1%	0.31
lp_pds_06	0.17	9.1%	0.20	23.1%	3.21
p010	0.18	9.3%	0.19	19.5%	1.43
ge	0.19	17.3%	0.21	34.9%	2.24
lp_osa_60	0.01	0.4%	0.01	8.1%	2.57
co9	0.17	16.3%	0.28	26.8%	4.91
lpl3	0.07	4.8%	0.13	19.7%	5.95
fome11	0.38	23.5%	0.45	17.5%	3.35
scrs8-2r	0.76	1783.4%	0.99	276.7%	0.14
stormg2-27	0.20	14.7%	0.20	24.6%	2.84
lp_ken_11	0.04	9.4%	0.03	17.0%	2.64
sctap1-2b	0.31	12.3%	0.86	79.2%	0.41
car4	0.03	2.9%	0.06	6.9%	0.45
lp_pds_10	0.16	8.9%	0.22	26.3%	3.04
lp_stocfor3	0.11	5.9%	0.09	11.9%	1.80
ex3sta1	0.48	28.6%	0.49	33.6%	1.46
testbig	0.23	12.5%	0.85	194.4%	0.30
dbir1	0.19	17.7%	0.83	32.1%	8.94
dbir2	0.19	16.6%	0.84	25.6%	8.63
scfxm1-2b	0.07	4.6%	0.07	17.1%	1.61
route	0.43	37.8%	0.54	77.9%	4.67

Continued on next page

Table 3.12 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
ts-palko	0.85	1195.9%	1.00	576.4%	8.88
fxm4_6	0.07	5.6%	0.07	20.3%	3.08
fome12	0.33	17.0%	0.44	19.6%	3.63
e18	0.66	10.3%	0.85	21.8%	3.63
pltexpa	0.19	8.1%	0.23	25.9%	2.42
baxter	0.31	40.6%	0.47	70.0%	0.94
lp_ken_13	0.03	5.5%	0.03	16.5%	2.89
stat96v2	0.11	3.8%	0.10	9.0%	72.51
lp_pds_20	0.16	7.7%	0.21	21.5%	3.43
stat96v3	0.10	2.9%	0.10	9.4%	49.37
world	0.14	9.2%	0.19	22.2%	2.19
mod2	0.15	12.1%	0.20	23.2%	2.16
sc205-2r	0.23	11.2%	0.97	82.4%	0.17
scfxm1-2r	0.06	4.7%	0.17	16.5%	1.31
fxm3_16	0.07	5.7%	0.05	10.1%	2.93
dbic1	0.05	4.1%	0.04	15.8%	9.22
fome13	0.25	15.3%	0.42	24.9%	3.68
pds-30	0.15	9.6%	0.20	22.6%	3.54
rlfprim	0.72	97.6%	0.87	243.0%	0.51
stormg2-125	0.15	11.7%	0.29	44.0%	2.03
pds-40	0.15	10.3%	0.19	20.6%	3.72
fome21	0.14	9.4%	0.18	19.0%	3.62
pds-50	0.15	9.1%	0.18	18.7%	3.77
pds-60	0.15	13.0%	0.17	17.6%	3.72
pds-70	0.14	10.3%	0.18	18.6%	3.91
pds-80	0.13	9.6%	0.17	18.7%	4.03
pds-90	0.13	9.7%	0.17	18.8%	3.79
pds-100	0.12	10.0%	0.15	20.8%	3.84
watson_1	0.04	2.7%	0.02	8.7%	3.89
sgpf5y6	0.03	7.7%	0.04	27.8%	1.87
watson_2	0.02	3.2%	0.02	18.4%	3.79
stormG2_1000	0.14	11.3%	0.55	114.6%	0.58
cont11_l	0.02	2.7%	0.02	8.2%	1.21

Table 3.13: 128-way partitioning performance of the PD matrix collection for cut-net metric with node balancing.

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
bcsstk13	0.97	1.6%	0.98	931.4%	1480.7%	353.6%	2.43
Chem97ZtZ	0.49	75.6%	0.56	216.7%	187.0%	79.1%	0.94
sts4098	0.79	38.7%	0.84	321.0%	326.9%	136.4%	1.20
ex15	0.57	3.1%	0.54	76.3%	61.8%	57.6%	2.08
Muu	0.77	1.2%	0.78	203.9%	98.4%	12.3%	1.60
Kuu	0.91	0.9%	0.91	1008.2%	244.6%	12.8%	2.24
bcsstk38	0.85	2.3%	0.87	460.8%	237.6%	88.2%	2.10
aft01	0.62	1.6%	0.60	52.6%	48.6%	28.3%	1.65
fv1	0.42	3.1%	0.43	17.5%	18.9%	18.3%	2.25
fv2	0.42	3.4%	0.43	16.2%	20.0%	19.4%	2.30
fv3	0.42	3.3%	0.43	16.2%	20.0%	19.4%	2.30
bundle1	1.00	5.9%	1.00	3222.4%	1918.4%	621.5%	0.34
ted_B_unscaled	0.38	1.5%	0.35	180.0%	98.1%	48.4%	1.24
ted_B	0.38	1.5%	0.34	175.1%	64.9%	24.1%	1.23
msc10848	0.97	0.4%	0.96	1372.4%	440.5%	89.9%	3.71
bcsstk17	0.79	3.0%	0.78	238.4%	126.7%	78.6%	2.49
t2dah_e	0.54	0.9%	0.52	33.7%	37.3%	22.5%	1.90
bcsstk18	0.53	3.6%	0.57	131.5%	117.6%	80.7%	1.57
cbuckle	0.80	2.6%	0.74	94.2%	56.6%	50.6%	2.80
crystm02	0.70	5.5%	0.71	69.8%	65.4%	54.1%	1.95
Pres_Poisson	0.81	3.6%	0.76	71.4%	54.3%	47.9%	2.80
bcsstm25	0.00	0.3%	0.00	0.3%	0.3%	0.3%	4.18
bcsstk25	0.66	2.0%	0.70	91.3%	66.8%	44.4%	1.69
Dubcova1	0.50	0.6%	0.48	33.5%	27.8%	10.2%	1.84
olafu	0.81	2.5%	0.79	309.3%	136.6%	80.8%	3.13
gyro_m	0.45	2.0%	0.49	160.4%	92.7%	52.9%	1.45
gyro	0.72	1.0%	0.81	790.8%	255.5%	70.1%	2.14
bodyy4	0.29	1.8%	0.30	11.7%	14.1%	13.1%	2.28
bodyy5	0.29	1.7%	0.29	11.3%	12.4%	11.4%	2.38
bodyy6	0.29	1.6%	0.29	10.8%	11.9%	11.0%	2.44
raefsky4	0.85	3.6%	0.83	264.8%	107.9%	71.5%	2.87
LFAT5000	0.02	0.8%	0.02	2.0%	1.1%	1.1%	2.20
LF10000	0.03	1.2%	0.03	1.8%	2.4%	2.4%	2.14
t3dl_e	0.00	0.6%	0.00	0.6%	0.6%	0.6%	4.22

Continued on next page

Table 3.13 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
bcsstk36	0.65	2.8%	0.62	85.1%	58.7%	49.4%	2.80
msc23052	0.65	2.6%	0.62	72.9%	61.9%	53.7%	2.98
crystm03	0.61	4.5%	0.61	45.8%	47.0%	43.6%	1.93
smt	0.94	0.4%	0.97	1277.2%	298.9%	54.4%	3.92
thread	0.95	0.5%	0.97	882.5%	282.9%	54.8%	3.74
wathen100	0.36	0.9%	0.35	12.8%	15.2%	7.6%	1.67
ship_001	0.90	0.6%	0.94	712.7%	265.4%	55.5%	3.81
nd12k	1.00	0.6%	1.00	701.4%	937.0%	101.7%	5.11
wathen120	0.34	0.8%	0.31	11.6%	13.9%	7.2%	1.64
obstclae	0.18	1.7%	0.16	11.3%	11.1%	11.0%	2.17
jnlbrng1	0.18	1.8%	0.17	11.6%	12.4%	12.4%	2.16
minsurfo	0.17	1.5%	0.17	11.8%	13.0%	12.9%	2.17
bcsstm39	0.00	0.2%	0.00	0.2%	0.2%	0.2%	4.30
vanbody	0.48	3.2%	0.46	45.6%	54.3%	49.6%	2.69
gridgena	0.25	1.8%	0.24	12.8%	16.0%	15.2%	1.87
cvxbqp1	0.17	2.2%	0.19	18.1%	17.7%	16.6%	2.13
ct20stif	0.50	3.2%	0.47	50.4%	71.2%	65.4%	2.60
crankseg_1	0.91	0.1%	0.95	932.0%	293.2%	26.2%	4.11
nasasrb	0.49	3.2%	0.45	24.9%	26.7%	26.2%	2.85
Andrews	0.77	4.3%	0.85	162.6%	98.9%	31.1%	1.53
crankseg_2	0.90	0.4%	0.94	800.4%	298.9%	48.5%	4.31
Dubcova2	0.27	0.2%	0.25	13.1%	9.2%	0.8%	1.72
qa8fm	0.53	4.7%	0.51	25.0%	37.1%	36.2%	1.79
cfd1	0.52	5.2%	0.51	30.5%	39.0%	38.2%	1.77
nd24k	0.99	0.5%	1.00	615.4%	454.5%	0.3%	5.56
oilpan	0.41	1.5%	0.39	23.1%	22.3%	17.5%	2.70
finan512	0.31	1.8%	0.25	9.4%	11.0%	3.9%	1.16
apache1	0.33	2.0%	0.32	18.6%	17.8%	17.4%	2.00
shallow_water1	0.11	1.3%	0.11	10.1%	10.0%	9.8%	2.12
shallow_water2	0.11	0.9%	0.11	11.7%	11.1%	10.9%	2.11
thermal1	0.14	2.5%	0.13	15.2%	14.9%	14.8%	2.03
denormal	0.23	1.6%	0.22	12.7%	12.7%	12.6%	1.89
s3dkt3m2	0.33	0.8%	0.32	9.5%	10.7%	9.4%	2.68
s3dkq4m2	0.37	2.1%	0.32	11.4%	13.3%	13.2%	3.09
m_t1	0.57	1.0%	0.52	45.3%	51.6%	35.9%	4.05
2cubes_sphere	0.53	1.7%	0.54	25.5%	24.1%	11.4%	1.56

Continued on next page

Table 3.13 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
thermomech_TK	0.12	2.2%	0.12	15.2%	15.4%	15.2%	2.18
thermomech_TC	0.12	2.2%	0.12	15.6%	15.3%	15.2%	2.17
x104	0.47	0.8%	0.46	39.6%	60.4%	45.4%	3.79
shipsec8	0.43	2.6%	0.42	33.9%	36.6%	34.6%	2.81
ship_003	0.53	2.9%	0.53	40.1%	34.9%	31.5%	2.74
cfd2	0.39	4.4%	0.37	25.8%	29.8%	29.6%	1.72
boneS01	0.42	2.0%	0.40	27.5%	35.4%	30.7%	2.56
shipsec1	0.36	2.8%	0.35	24.0%	30.7%	30.3%	2.95
bmw7st_1	0.29	3.5%	0.27	28.6%	28.7%	28.1%	2.82
Dubcova3	0.26	0.2%	0.22	12.3%	12.4%	1.5%	1.58
bmwcra_1	0.52	5.0%	0.50	47.7%	49.3%	48.0%	2.53
G2_circuit	0.14	1.3%	0.14	9.4%	11.9%	11.8%	1.90
shipsec5	0.34	3.3%	0.33	22.2%	26.5%	26.2%	2.90
thermomech_dM	0.08	2.3%	0.08	13.4%	13.2%	13.2%	2.19
pwtk	0.25	2.1%	0.23	15.3%	16.0%	15.9%	3.11
hood	0.21	1.5%	0.21	18.9%	18.4%	15.8%	2.77
BenElechi1	0.23	1.8%	0.20	14.7%	16.0%	16.0%	3.09
offshore	0.32	3.4%	0.32	24.8%	27.6%	21.0%	1.55
F1	0.43	0.8%	0.43	43.4%	56.6%	33.9%	1.90
msdoor	0.12	2.4%	0.12	17.7%	17.4%	16.8%	2.78
af_3_k101	0.13	0.9%	0.13	7.6%	7.2%	7.1%	2.26
af_1_k101	0.13	0.8%	0.13	7.2%	7.7%	7.5%	2.27
af_2_k101	0.13	0.9%	0.13	6.7%	7.5%	7.4%	2.27
af_4_k101	0.13	0.8%	0.13	7.2%	7.8%	7.7%	2.25
af_0_k101	0.13	0.9%	0.13	7.2%	7.7%	7.6%	2.26
af_5_k101	0.13	0.8%	0.13	7.9%	8.1%	8.0%	2.25
inline_1	0.27	0.8%	0.26	25.0%	26.0%	16.8%	2.09
af_shell7	0.13	1.0%	0.12	7.2%	7.6%	7.4%	2.26
af_shell3	0.13	1.0%	0.12	7.8%	8.2%	8.1%	2.27
af_shell4	0.13	0.9%	0.12	8.2%	8.8%	8.6%	2.25
af_shell8	0.13	1.1%	0.12	9.3%	9.8%	9.6%	2.25
parabolic_fem	0.06	0.6%	0.06	10.1%	10.2%	10.1%	2.10
apache2	0.13	1.0%	0.12	12.2%	12.0%	11.9%	1.79
tmt_sym	0.05	2.3%	0.05	12.2%	12.3%	12.3%	1.66
boneS10	0.13	3.7%	0.12	20.6%	23.3%	22.8%	2.58
ldoor	0.09	2.3%	0.09	13.8%	13.2%	12.8%	2.75

Continued on next page

Table 3.13 – continued from previous page

	PaToH		*onmetisHP*				
name	cutsize	$\%LI$	cutsize	$exp\%LI_p$	$act\%LI_p$	$act\%LI_c$	speedup
ecology2	0.04	0.3%	0.04	8.0%	8.0%	8.0%	1.63
thermal2	0.04	2.3%	0.04	10.2%	10.1%	10.1%	1.80
G3_circuit	0.04	0.6%	0.04	7.6%	7.2%	7.2%	1.68

Table 3.14: 128-way partitioning performance of the PD matrix collection for connectivity metric with node balancing.

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
bcsstk13	7.65	7.7%	6.92	151.5%	0.55
Chem97ZtZ	1.52	75.6%	1.53	230.4%	0.75
sts4098	3.41	38.7%	3.45	141.8%	0.43
ex15	1.79	3.1%	1.77	30.1%	1.70
Muu	2.32	1.4%	2.30	43.6%	1.01
Kuu	3.21	1.4%	3.06	75.0%	0.91
bcsstk38	2.99	3.7%	2.88	94.4%	0.94
aft01	1.86	1.3%	1.83	25.2%	1.41
fv1	1.52	2.8%	1.54	9.7%	2.00
fv2	1.51	3.0%	1.53	10.3%	1.99
fv3	1.51	3.0%	1.53	11.0%	1.99
bundle1	10.02	16.5%	5.71	484.2%	0.15
ted_B_unscaled	1.42	1.9%	1.34	42.1%	1.03
ted_B	1.42	2.6%	1.34	48.2%	1.03
msc10848	4.33	1.3%	3.80	133.1%	0.92
bcsstk17	2.37	3.9%	2.32	47.9%	1.39
t2dah_e	1.71	1.2%	1.68	17.7%	1.47
bcsstk18	1.97	4.7%	2.02	40.3%	1.13
cbuckle	2.23	3.6%	2.16	35.2%	1.60
crystm02	2.19	4.6%	2.20	32.0%	1.33
Pres_Poisson	2.29	3.2%	2.24	31.5%	1.65
bcsstm25	1.00	0.3%	1.00	0.3%	3.28
bcsstk25	2.11	2.9%	2.15	23.6%	1.16
Dubcova1	1.64	1.0%	1.61	15.5%	1.49

Continued on next page

Table 3.14 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
olafu	2.38	3.6%	2.30	56.5%	1.48
gyro_m	1.63	2.4%	1.62	51.3%	1.04
gyro	2.52	1.5%	2.47	105.0%	0.82
bodyy4	1.33	2.2%	1.34	7.9%	2.04
bodyy5	1.33	1.8%	1.33	8.8%	2.02
bodyy6	1.32	1.6%	1.33	6.9%	2.02
raefsky4	2.82	4.7%	2.81	39.6%	1.22
LFAT5000	1.02	0.8%	1.02	1.1%	1.85
LF10000	1.03	1.1%	1.03	1.6%	1.81
t3dl_e	1.00	0.6%	1.00	0.6%	3.35
bcsstk36	1.98	4.0%	1.92	35.5%	1.74
msc23052	1.97	3.7%	1.93	34.7%	1.84
crystm03	1.89	4.2%	1.88	21.8%	1.41
smt	3.86	1.2%	3.62	67.4%	0.91
thread	4.05	1.3%	3.82	62.3%	0.93
wathen100	1.43	1.2%	1.41	6.6%	1.38
ship_001	3.02	1.4%	2.95	60.4%	1.21
nd12k	7.60	4.1%	8.38	106.1%	0.47
wathen120	1.39	1.3%	1.36	6.4%	1.40
obstclae	1.18	1.6%	1.18	9.6%	1.87
jnlbrng1	1.19	2.2%	1.18	8.7%	1.86
minsurfo	1.19	1.8%	1.18	7.7%	1.86
bcsstm39	1.00	0.2%	1.00	0.2%	3.45
vanbody	1.64	4.3%	1.60	26.3%	1.88
gridgena	1.27	1.9%	1.27	9.4%	1.57
cvxbqp1	1.23	1.9%	1.26	12.2%	1.81
ct20stif	1.71	3.8%	1.66	31.8%	1.73
crankseg_1	3.65	0.7%	3.39	73.3%	0.82
nasasrb	1.60	3.2%	1.57	10.2%	2.04
Andrews	2.50	5.9%	2.76	43.8%	1.05
crankseg_2	3.59	0.6%	3.28	93.7%	0.84
Dubcova2	1.31	0.7%	1.28	5.9%	1.48
qa8fm	1.79	4.3%	1.75	17.2%	1.37
cfd1	1.74	5.1%	1.73	20.5%	1.38
nd24k	5.60	4.7%	5.95	80.5%	0.64
oilpan	1.48	2.0%	1.47	11.5%	2.08

Continued on next page

Table 3.14 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
finan512	1.34	2.2%	1.25	8.2%	1.06
apache1	1.38	2.1%	1.36	11.0%	1.77
shallow_water1	1.12	1.3%	1.12	9.6%	1.84
shallow_water2	1.12	0.9%	1.12	9.2%	1.84
thermal1	1.14	2.1%	1.14	13.2%	1.76
denormal	1.26	1.5%	1.24	9.3%	1.59
s3dkt3m2	1.37	1.3%	1.37	6.2%	2.09
s3dkq4m2	1.42	1.9%	1.38	9.0%	2.42
m_t1	1.77	1.4%	1.70	26.1%	2.53
2cubes_sphere	1.74	2.3%	1.76	9.6%	1.23
thermomech_TK	1.13	2.2%	1.12	13.7%	1.90
thermomech_TC	1.13	2.4%	1.12	14.5%	1.89
x104	1.65	1.0%	1.60	22.6%	2.48
shipsec8	1.55	3.3%	1.53	18.3%	2.02
ship_003	1.72	3.3%	1.71	20.9%	1.82
cfd2	1.50	4.0%	1.47	20.8%	1.42
boneS01	1.55	2.6%	1.51	18.5%	1.82
shipsec1	1.44	2.8%	1.43	17.4%	2.25
bmw7st_1	1.34	3.8%	1.32	19.4%	2.25
Dubcova3	1.28	0.5%	1.24	7.5%	1.38
bmwcra_1	1.71	5.0%	1.71	30.0%	1.65
G2_circuit	1.15	1.4%	1.14	8.8%	1.66
shipsec5	1.42	3.7%	1.41	16.0%	2.22
thermomech_dM	1.08	2.3%	1.08	13.3%	1.94
pwtk	1.28	2.5%	1.26	9.9%	2.54
hood	1.23	1.7%	1.23	14.2%	2.28
BenElechi1	1.24	1.5%	1.22	12.6%	2.58
offshore	1.40	3.9%	1.40	16.2%	1.31
F1	1.61	0.7%	1.60	22.4%	1.25
msdoor	1.13	2.7%	1.13	16.1%	2.42
af_3_k101	1.14	0.5%	1.14	7.1%	1.96
af_1_k101	1.14	0.8%	1.14	7.7%	1.96
af_2_k101	1.14	0.7%	1.14	7.0%	1.96
af_4_k101	1.14	0.5%	1.14	6.8%	1.96
af_0_k101	1.14	0.6%	1.14	8.2%	1.96
af_5_k101	1.14	0.7%	1.14	8.3%	1.96

Continued on next page

Table 3.14 – continued from previous page

	PaToH		*onmetisHP*		
name	cutsize	$\%LI$	cutsize	$\%LI$	speedup
inline_1	1.30	1.0%	1.29	13.8%	1.68
af_shell7	1.13	0.7%	1.13	7.7%	1.95
af_shell3	1.13	0.7%	1.13	8.4%	1.97
af_shell4	1.13	1.0%	1.13	7.5%	1.95
af_shell8	1.13	0.9%	1.13	7.6%	1.96
parabolic_fem	1.06	0.5%	1.06	9.9%	1.88
apache2	1.13	1.0%	1.12	10.9%	1.61
tmt_sym	1.05	2.5%	1.05	11.1%	1.50
boneS10	1.14	3.6%	1.12	17.3%	2.27
ldoor	1.10	2.4%	1.10	14.3%	2.45
ecology2	1.04	0.2%	1.04	8.6%	1.47
thermal2	1.04	2.2%	1.04	11.3%	1.66
G3_circuit	1.04	0.6%	1.04	6.3%	1.55

3.5 Conclusions

We have presented how the hypergraph partitioning problem can be efficiently and effectively implemented through recursive graph bipartitioning by vertex separators. Our empirical study on a wide set of test matrices showed that runtimes can be as much as 4.17 times faster, where the cutsize quality is preserved on average (and improved in many cases), while balance was achieved when the number of parts is small and remained acceptable when the number of parts is large. Moreover, we proposed techniques that can trade off cutsize and runtime against balance, showing that balance can be achieved even when the number of parts is very large.

What motivates us to investigate NIGs to solve HP problems arising in scientific computing applications is that in many applications, definition of balance cannot be very precise [2, 35, 36] or there are additional constraints that cannot

be easily incorporated into partitioning algorithms and tools [37]; or partitioning is used as part of a divide-and-conquer algorithm [38]. For instance, hypergraph models can be used to permute a linear program (LP) constraint matrix to a block angular form for parallel solution with decomposition methods. Load balance can be achieved by balancing subproblems during partitioning. However, it is not possible to accurately predict solution time of an LP, and equal sized subproblems only increase the likelihood of computational balance.

Hypergraph models have recently been used to find null-space bases that have a sparse inverse [38]. This application requires finding a column-space basis B as a submatrix of a sparse matrix A, so that B^{-1} is sparse. Choosing B to have a block angular form limits the fill in B^{-1}, but merely a block angular form for B will not be sufficient, since B has to be nonsingular to be a column-space basis for A. Enforcing numerical or even structural nonsingularity of subblocks during partitioning is a nontrivial task, if at all possible, and thus partitioning is used as part of a divide-and-conquer paradigm, where the partitioning phase is followed by a correction phase, if subblocks are non-singular. Both of these cases present examples of applications, where hypergraphs provide effective models, but balance among parts is only weakly defined.

Overall results prove that the proposed hypergraph partitioning through vertex separators on graphs is ideal for applications where balance is not well-defined, which is the main motivation for our work, and competitive for applications where balance is important.

Chapter 4

Term-based Inverted Index Partitioning based on Hypergraph Partitioning

There are two main parallelism of query processing based on the partitioned object which can be either documents or terms. The comparisons between term-based and doc-based inverted index partitioned query processing can be found in [48–51]. In document-based index partitioning [52–54], each index server is assigned a subset of documents and a local inverted index is built upon this subset. Query processing is realized as follows. A query is processed in all local indexes in a parallel manner and the partial results are merged by a central broker. The biggest drawback of this system is the large number of disk accesses while achieving good load balance.

In term-based index partitioning [52,53,55–58], the global index over the whole document collection is partitioned among the index servers. That is, each index server gets the responsibility of processing a subset of terms of the vocabulary. The main objective of this work is to come up with a term-based index partitioning model that captures both load imbalance and the communication overhead incurred during query processing. In this work, this objective is satisfied by a

hypergraph partitioning model. The closest works are [55] and [58], but these works does not make use of query logs.

4.1 Background

4.1.1 Term-based Index Partitioning

An inverted index $\mathcal{L}$ contains a set of term and corresponding inverted list pairs, i.e., $\mathcal{L}=\{(t_1,\mathcal{I}_1),(t_2,\mathcal{I}_2),\ldots,(t_T,\mathcal{I}_T)\}$, where $T=|\mathcal{T}|$ is the size of vocabulary $\mathcal{T}$ of the indexed document collection. Each posting $p\in\mathcal{I}_i$ keeps information about a document in which term t_i appears. Typically, this information includes the document's id and term's frequency in the document.

In a distributed text retrieval system with K nodes, postings of an index are partitioned on a set $\mathcal{S}=\{S_1,S_2,\ldots,S_K\}$ of K index servers. A term-based index partition $\Phi=\{\mathcal{T}_1,\mathcal{T}_2,\ldots,\mathcal{T}_K\}$ is a partition of $\mathcal{T}$ such that

$$\mathcal{T}=\bigcup_{1\leq k\leq K}\mathcal{T}_k \tag{4.1}$$

with the condition that

$$\mathcal{T}_k\cap\mathcal{T}_\ell=\emptyset,\qquad \text{for } 1\leq k,\ell\leq K \text{ and } k\neq\ell. \tag{4.2}$$

Based on partition Φ, every term subset $\mathcal{T}_k$ is uniquely assigned to an index server S_k, and each index server S_k constructs a subindex $\mathcal{L}_k$ as

$$\mathcal{L}_k=\{(t_i,\mathcal{I}_i):t_i\in\mathcal{T}_k\}, \tag{4.3}$$

i.e., index servers are responsible for maintaining only their own sets of terms and hence all postings of an inverted list are assigned together to the same server. Typically, the partitioning is performed such that the computational load distribution on index servers is balanced.

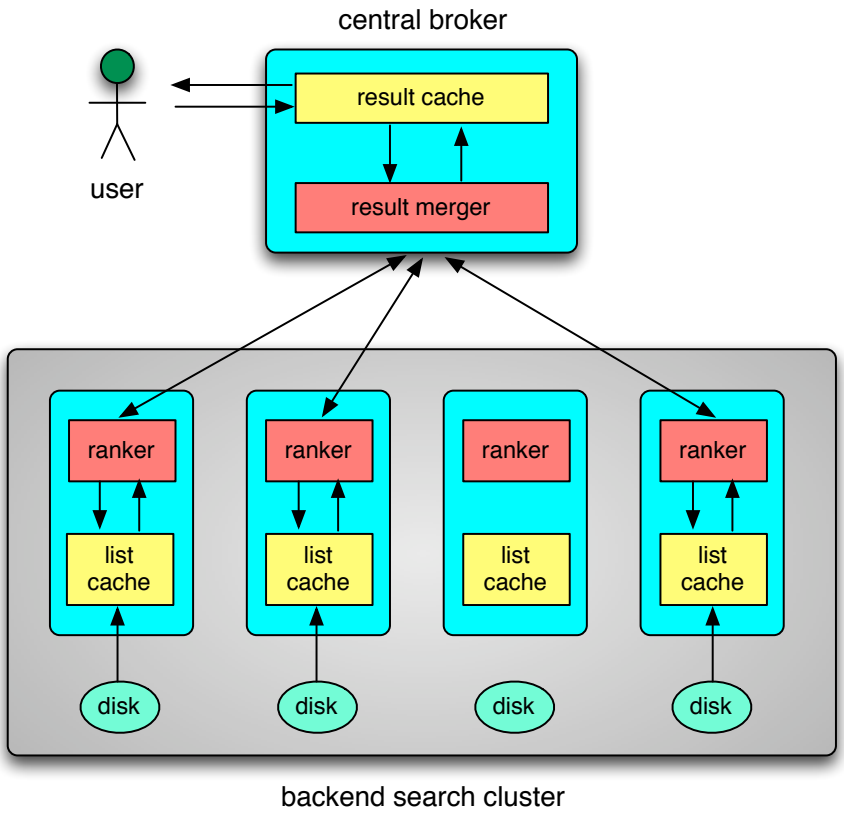


Figure 4.1: Query processing architecture with a central broker and a number of index servers

4.1.2 Parallel Query Processing

Given a term-based-partitioned index, the query processing scheme [48–50, 52] performs as follows. The architecture, as shown in Figure 5.5, is composed of a central broker and a number of index servers. The central broker is responsible for preprocessing the query and issuing it to index servers in the search cluster. Query processing is performed as follows. First, the broker divides the original query $q = \{t_1, t_2, \ldots, t_{|q|}\}$ into a set $\{\widehat{q}_1, \widehat{q}_2, \ldots, \widehat{q}_K\}$ of K subqueries, in compliance with the way the index is partitioned. Each subquery $\widehat{q}_k$ contains the query terms whose responsibility is assigned to index server S_k, i.e., $\widehat{q}_k = \{t_i \in q : (t_i, \mathcal{I}_i) \in \mathcal{L}_k\}$. Then, the central broker issues each subquery to its respective index server. Depending on the terms in the query, it is possible to have $\widehat{q}_k = \emptyset$, in which case no subquery is issued to S_k. Each index server S_k accesses its disk and reads the

inverted lists associated with the terms in $\widehat{q}_k$, i.e., for each query term $t_i \in \widehat{q}_k$, inverted list $\mathcal{I}_i$ is fetched from the disk. There are two typical matching criteria. In the conjunctive mode which is based on the AND logic, a document is said to be a match whenever it appears in all inverted lists associated with the query terms. On the other hand, in the disjunctive mode which is based on the OR logic, a document is said to be a match whenever it appears in any inverted lists associated with the query terms. The matching documents are sorted in decrasing order of scores and this sorted document list constitutes the partial ranking of documents by the index server, and this partial ranking is transferred to the central broker for final ranking.

High computational load imbalance causes high skewness in sizes of inverted lists and the access frequencies. A solution to this problem is to replicate a small fraction of load-intensive term inverted lists accross all index servers [56,57]. This approach introduces server selection problem during the query processing. Since a replicated list is available to all servers, for a query that contains a term with replicated list, the central broker should decide which server should be responsible for the processing of the term. The technique in [57] restricts the set of responsible servers to those that hold at least one non-replicated term of the same query, thus ensuring that the coherency of list accesses is not disturbed.

4.2 Problem Formulation

Given an invertex index $\mathcal{L} = \{(t_1, \mathcal{I}_1), (t_2, \mathcal{I}_2), \ldots\}$ that contains the vocabulary set $\mathcal{T} = \{t_1, t_2, \ldots\}$ of terms and a stream $\mathcal{Q} = \{q_1, q_2, \ldots\}$ of queries. Each query $q_j \in \mathcal{Q}$ is composed of terms in the vocabulary, i.e., $q_j \subseteq \mathcal{T}$. Each term $t_i \in \mathcal{T}$ is associated with an inverted list $\mathcal{I}_i$ and a query frequency p_i. We are also a given a set $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ of K index servers. Given these, we provide a number of definitions before we formally state our problem.

Definition 4 (Hitting set of a query) *For a term partition Φ, the hitting set $h(q_j, \Phi)$ of a query $q_j \in \mathcal{Q}$ is defined as the set of index servers that hold at least*

one term of q_j [57], *i.e.,*

$$h(q_j, \Phi) = \{S_k \in \mathcal{S} : q_j \cap \mathcal{T}_k \neq \emptyset\}. \tag{4.4}$$

Let $c(q_j, \Phi)$ denote the communication overhead incurred when processing q_j under term partition Φ. Also, let $\widehat{q}_{jk}$ denote the subquery to be processed by index server S_k for a query $q_j \in \mathcal{Q}$, i.e., $\widehat{q}_{jk} = q_j \cap \mathcal{T}_k$. Depending on the properties of the distributed system, the communication overhead may be determined by the number of network messages sent by index servers or the volume of data communicated over the network. The number of network messages is proportional to the size of the hitting set of the query. In this case, we have

$$c(q_j, \Phi) = 2|h(q_j, \Phi)|, \tag{4.5}$$

independent of the query processing scheme. The communication volume, however, depends on the query processing scheme. In case of the traditional query processing scheme, the communication overhead can be written as

$$c(q_j, \Phi) = \begin{cases} 0, & |h(q_j, \Phi)| = 1; \\ \sum_k |f(\widehat{q}_{jk})|, & \text{otherwise.} \end{cases} \tag{4.6}$$

Here and hereafter, f denotes a matching function that maps a query/subquery to a set of documents, i.e., $f(q_j)/f(\widehat{q}_{jk})$ represents the set of documents (in the collection) that match $q_j/\widehat{q}_{jk}$.

In processing of every subquery, we can assume that each term $t \in \widehat{q}_j$ incurs a workload proportional to its inverted list size [59]. This implies that each term $t_i \in \mathcal{T}$ introduces a workload $p_i \times |\mathcal{I}_i|$. Hence, the overall workload $L_k(\Phi)$ of a server S_k with respect to a given term partition Φ becomes

$$L_k(\Phi) = \sum_{t_i \in \mathcal{T}_k} p_i \times |\mathcal{I}_i|. \tag{4.7}$$

Definition 5 (ϵ-balanced partition) *Given an inverted index* $\mathcal{L}$*, a query stream* $\mathcal{Q}$*, and a server set* $\mathcal{S}$*, a term partition* Φ *is said to be* ϵ*-balanced if the workload* $L_k(\Phi)$ *of each server* S_k *satisfies the constraint*

$$L_k(\Phi) \leq L_{\text{avg}}(\Phi)(1 + \epsilon), \tag{4.8}$$

where $L_{\text{avg}}(\Phi)$ refers to the average workload of index servers.

Problem 1 (Term-based index partitioning problem) *Given an inverted index $\mathcal{L}$, a query stream $\mathcal{Q}$, a server set $\mathcal{S}$, and a parameter $\epsilon \geq 0$, find an ϵ-balanced term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ that induces an index partition $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_K\}$ such that the total communication overhead $\Psi(\Phi)$ is minimized, where*

$$\Psi(\Phi) = \sum_{q_j \in Q} c(q_j, \Phi). \tag{4.9}$$

4.3 The Hypergraph Model

In our model, we represent the interaction between the queries in a query set $\mathcal{Q}$ and the inverted lists in an inverted index $\mathcal{L}$ by means of a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. In $\mathcal{H}$, each term $t_i \in \mathcal{T}$ is represented by a vertex $v_i \in \mathcal{V}$, and each query $q_j \in \mathcal{Q}$ is represented by a net $n_j \in \mathcal{N}$. Each net n_j connects the set of vertices representing the terms that constitute query q_j. That is,

$$\text{Pins}(n_j) = \{v_i : t_i \in q_j\}. \tag{4.10}$$

Each vertex v_i is associated with a weight $w_i = f_i \times |\mathcal{I}_i|$ which represents the computational load that is estimated to be incurred by t_i.

A K-way vertex partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of hypergraph $\mathcal{H}$ is decoded as a K-way term partition $\Phi = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_K\}$ of $\mathcal{T}$. That is, each vertex part $\mathcal{V}_\ell$ in Π corresponds to the subset $\mathcal{T}_\ell$ of terms assigned to index server S_ℓ. Due to the adopted vertex weighting scheme, balancing the part weights according to the balance criterion in Eq. 2.2 effectively balances the computational load among the index servers, satisfying the constraint in Eq. 4.8.

In a K-way vertex partition Π of $\mathcal{H}$, consider a net n_j with connectivity set $\Lambda(n_j, \Pi)$. By definition, for each part $V_\ell \in \Lambda(n_j, \Pi)$, we have $\text{Pins}(n_j) \cap \mathcal{V}_\ell \neq \emptyset$, i.e., $q_j \cap \mathcal{T}_\ell \neq \emptyset$. Thus, $\mathcal{T}_\ell \in h(q_j, \Phi)$ if and only if $V_\ell \in \Lambda(n_j, \Pi)$. This implies

$$h(q_j, \Phi) = \{\mathcal{S}_\ell \in \mathcal{S} : \mathcal{V}_\ell \in \Lambda(n_j, \Pi)\}, \tag{4.11}$$

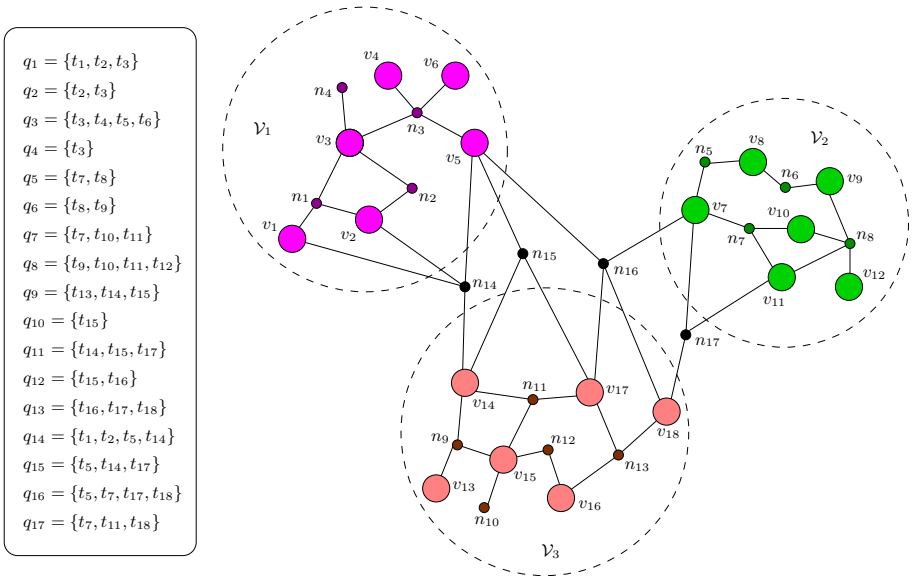


Figure 4.2: A three-way partitioning of the hypergraph representing an inverted index.

which shows the one-to-one correspondence between the connectivity set $\Lambda(n_j, \Pi)$ of a net n_j in Π and the hitting set $h(q_j, \Phi)$ of query q_j in Φ, induced by Π. Hence, the minimization of the cutsize according to the connectivity metric (Eq. 2.9) accurately captures the minimization of the total communication cost $\Psi(\Phi)$ when $c(q_j, \Phi)$ is modeled as Eq. 4.5. Moreover, the minimization of the cutsize according to the cutnet (Eq. 2.8) metric approximates the minimization of the total communication cost $\Psi(\Phi)$ when $c(q_j, \Phi)$ is modeled as Eq. 4.6.

We demonstrate the model over an example, involving a toy inverted index with vocabulary $\mathcal{T} = \{t_1, t_2, \ldots, t_{18}\}$ and a query set $\mathcal{Q} = \{q_1, q_2, \ldots, q_{17}\}$, where $q_1 = \{t_1, t_2, t_3\}$, $q_2 = \{t_2, t_3\}$, $q_3 = \{t_3, t_4, t_5, t_6\}$, $q_4 = \{t_3\}$, $q_5 = \{t_7, t_8\}$, $q_6 = \{t_8, t_9\}$, $q_7 = \{t_7, t_{10}, t_{11}\}$, $q_8 = \{t_9, t_{10}, t_{11}, t_{12}\}$, $q_9 = \{t_{13}, t_{14}, t_{15}\}$, $q_{10} = \{t_{15}\}$, $q_{11} = \{t_{14}, t_{15}, t_{17}\}$, $q_{12} = \{t_{15}, t_{16}\}$, $q_{13} = \{t_{16}, t_{17}, t_{18}\}$, $q_{14} = \{t_1, t_2, t_5, t_{14}\}$, $q_{15} = \{t_5, t_{14}, t_{17}\}$, $q_{16} = \{t_5, t_7, t_{17}, t_{18}\}$, and $q_{17} = \{t_7, t_{11}, t_{18}\}$. We assume that the retrieval system has three index servers ($\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$).

Figure 4.2 shows a three-way partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \mathcal{V}_3\}$ of the toy index. We can interpret the figure as follows. Terms t_1 to t_6 are assigned to server $\mathcal{S}_1$; terms t_7 to t_{12} are assigned to server $\mathcal{S}_2$; and the remaining terms t_{13} to t_{18} are assigned

Table 4.1: Fraction of queries with a particular length

	Avg	Uniq	1	2	3	4	≥ 5
$\mathtt{S}_2$	2.76	0.15	0.12	0.36	0.31	0.14	0.08
$\mathtt{S}_3$	2.76	0.15	0.12	0.36	0.31	0.14	0.08

to server $\mathcal{S}_3$. According to this assignment, the queries in sets $\{q_1, q_2, q_3, q_4\}$, $\{q_5, q_6, q_7, q_8\}$, and $\{q_9, q_{10}, q_{11}, q_{12}\}$ can be fully processed by servers $\mathcal{S}_1$, $\mathcal{S}_2$, and $\mathcal{S}_3$, respectively. For these queries, the servers communicate only their final top k result sets, i.e., the communication of the partial score information is not necessary. However, the remaining queries q_{14}, q_{15}, q_{16}, and q_{17} necessitate the communication of partial scores. In particular, during the processing of q_{16}, all three servers communicate their partial scores. For queries q_{14} and q_{15}, only servers $\mathcal{S}_1$ and $\mathcal{S}_3$, and for query q_{17}, only servers $\mathcal{S}_2$ and $\mathcal{S}_3$ need to communicate data. Consequently, the number of queries that require communication of partial scores is 4, whereas the number of messages needed to process all the queries (refer to Eq. 4.5) is equal to $2 \times (13 \times 1 + 3 \times 2 + 1 \times 3) = 44$.

4.4 Experimental Results

We sample 1.7 million web pages that are predicted by a proprietary classifier as belonging to the `.fr` domain. We also sample about 6.3 million queries from the French front-end of Yahoo! web search for three consecutive days (query sets $\mathtt{S}_1$, $\mathtt{S}_2$, and $\mathtt{S}_3$, each containing queries of a different day). Queries in $\mathtt{S}_2$ are used in model construction to obtain the co-occurrence relations of terms while queries in $\mathtt{S}_3$ are used for performance evaluation.

To create a more realistic setup, we assume a query result cache with infinite capacity [60], i.e., only unique queries are used in the model construction and evaluation steps. In each of the two steps, the queries of the previous day ($\mathtt{S}_1$ and $\mathtt{S}_2$, respectively) are used to warm up the result cache. We note that, although filtered by the result cache, frequencies of terms in queries still follow a power-low distribution as demonstrated in [61, Figure 5]. The query length distribution, calculated over miss queries, is given in Table 4.1.

To partition constructed hypergraphs, we use the PaToH tool [6] with its default parameters (except for the imbalance constraint, which we set to 5%). We vary the number of index servers (parts) such that $K \in \{4, 8, 12, 16\}$ and linearly scale the size of the document collection with increasing K. In particular, $K \times 100,000$ documents are used in a partitioning run with K servers.

Training set $\mathtt{S_2}$ is used to extract the popularity values of terms in order to calculate estimated workloads. We assume that queries are processed in the conjunctive (AND) mode. Hence, a query is processed only if all of its terms occur in the vocabulary. The fraction of such queries ranges between 83% and 87% (over miss queries), depending on the collection size. All reported results are given over miss and vocabulary queries. As the baseline index partitioning technique, we use the bin packing approach, where inverted lists are assigned to index servers in decreasing order of their sizes, as in greedy bin packing. The baseline technique and the proposed hypergraph-partitioning-based model are denoted as `BIN` and `HP`, respectively. Due to the randomized nature of heuristics used in PaToH, experiments using `HP` are repeated five times and averages are reported.

We couple the above-mentioned partitioning schemes with index replication. In particular, we replicate the longest 100 lists on all servers [56] with an overhead of 20%, 47%, 73%, and 100% increase in the total index size for $K = 1, 2, 3,$ and 4, respectively. We employ three different techniques to identify the server responsible for processing a replicated list. The first technique selects the server arbitrarily, referred as `GLB`, and is adopted from [56]. The second one uses a server among the servers those are active due to a non-replicated term in the query, referred as `LOC`, and is adopted from [57]. The third technique selects the server that keeps the longest list of the terms of the current query, referred as `MAX`, and is a novel approach.

In Figures 4.3 and 4.4, `S3` identifies the worst-case scenario, that is, it represents the number of queries with single term in Figure 4.3 and the number of queries with the number of terms equal to the number of servers in x-axis in Figure 4.4. Thus, `S3` shows the worst-case scenario. Figure 4.3 displays the fraction

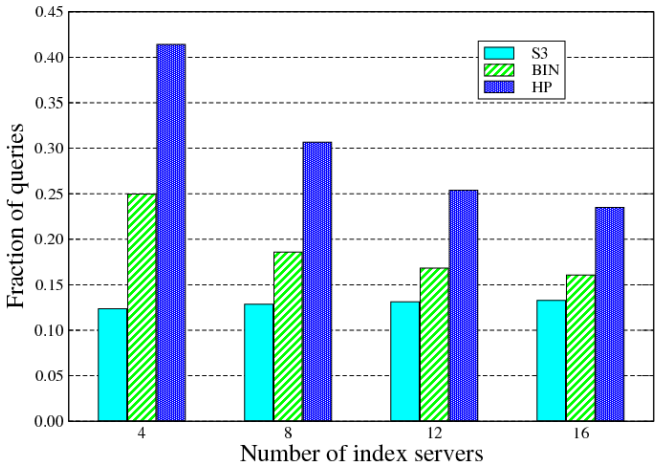


Figure 4.3: Fraction of locally processed queries.

of queries that processed only on one index server. The `BIN` approach achieves considerably better locality than this lower bound since the query lengths are not sufficiently small relative to the number of index servers. This is supported by the decreasing in the gap between the two with increasing number of index servers. The locality decreases with increasing number of index servers, which is expected since the query length remains the same while the number of index servers increases. In the figure, `HP` refers to hypergraph partitioning according to the cutnet metric as defined in Equation 2.8. As seen in the figure, the HP-based approach effectively improves the locality. We also note that the locality relative to the baseline also decreases. Figure 4.4 presents the fraction of queries with a given number s of index servers among all queries, for $s \in \{1, 2, 3, 4\}$, and $s > 4$ when the number of index servers is equal to 16. In this figure, `HP` refers to hypergraph partitioning according to the connectivity metric as defined in Equation 2.9. As seen in the figure, remarkably more queries are processed on fewer index servers when the HP-based approach is used.

Figure 4.5 illustrates the savings (one figure for each cost model) in the communication overhead, when modeled as Equation 4.6, as normalized by those of the `BIN` approach with global randomization of replicated terms, referred to as

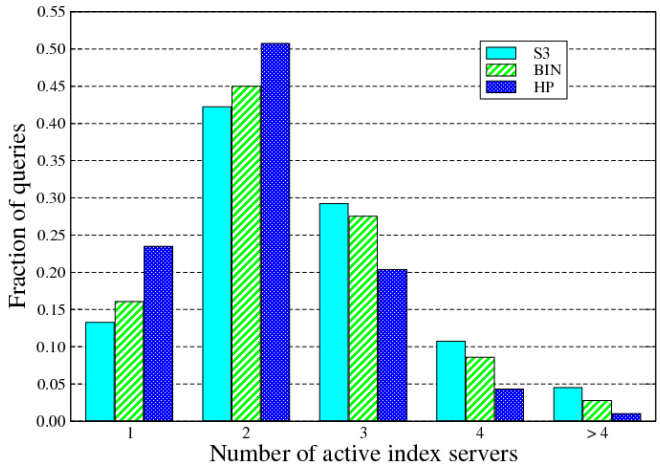


Figure 4.4: Fraction of queries with a given number of active index servers (right) among all queries.

BIN-GLB. The figures reveal that huge savings in the network overhead can be obtained by preserving the coherency of query terms. The savings decreases with increasing number of index servers for both the BIN and HP methods, which can be explained by a reasoning similar to that of the locality. As seen in the figure, hypergraph-partitioning-based approaches achieve significant savings (about 25%–42%) in the communication overhead in this scenario.

The improved savings in the communication costs come at the cost of workload imbalance. Table 4.2 illustrates how much degradation is observed in the workload balance. Each row represents experiments with different number of index servers (K). The first five columns contain the results related to the BIN approach while the latter columns are those of the HP approach. For each of the two groups, we present imbalance values for all four replicated-term assignment strategies. The fifth column of each group represents the storage imbalance values that are side results of assignments. As seen in the figure, GLB approaches achieve perfect load balance independent of the method (at the cost of higher communication volumes), which also reflects the dominancy of the replicated terms. The LOC method behaves similar to the GLB method in case of the BIN approach, with conserving any locality that may occur for a query. Thanks to the balancing

Figure 4.5: Savings in communication overhead where cost is modeled as in Eq. 4.6 as normalized to those of `BIN-GLB`.

Table 4.2: Comparative query processing load imbalance values of `BIN` and `HP`.

	`BIN`				`HP`			
K	GLB	LOC	MAX	Storage	GLB	LOC	MAX	Storage
4	1.00	1.00	1.03	1.05	1.00	1.04	1.11	1.12
8	1.00	1.00	1.03	1.08	1.00	1.06	1.13	1.26
12	1.00	1.00	1.05	1.11	1.00	1.06	1.19	1.32
16	1.00	1.00	1.09	1.16	1.00	1.08	1.19	1.32

constraint in the hypergraph model, we observe satisfactory workload balance (less than 10%). Storage imbalances are important side results and observed at admisable amounts, i.e., up to 16% and 32% for the `BIN` and `HP` approaches, respectively.

4.5 Conclusion

In this chapter, we proposed a combinatorial model for term-based inverted index partitioning and showed that the proposed model accurately captures the

communication costs incurred in query processing on term-based-partitioned indexes. We empirically demonstrated that, relative to a standard bin-packing-based partitioning strategy, the proposed model achieves higher savings in the communication of partial scores.

Chapter 5

Row-Columnwise Sparse Matrix Partitioning based on Hypergraph Partitioning with Cooccurence

Given an $n \times n$ matrix A and a vector x, the sequential matrix vector multiplication $y = Ax$ performs as

$$y_i = \sum_{j:\ a_{ij} \neq 0} x_j \times a_{ij},\ \forall\ 1 \leq j \leq n. \tag{5.1}$$

In parallel matrix vector multiplication, each processor holds a separate subset of x-vector entries and responsible for the computing of a separate subset of y-vector entries. In many applications, the successive multiplication is required where an x-vector entry of a multiplication is obtained by a linear transformation of the corresponding y-vector entry computed in the previous multiplication. Thus, we investigate the parallel matrix vector multiplication where a processor is responsible for the computation of a y-vector entry, say y_i, whenever that processor holds the corresponding x-vector entry, namely x_i. This scheme requires a row and column partition of matrix such that row r_i and column c_i are both

partitioned into the same part for $1 \leq i \leq n$, so called *symmetric partition.*

5.1 Background

There are three main parallel sparse matrix vector multiplication schemes exist in the literature. These are row-parallel, column-parallel and row-column-parallel, each of which is associated with a hypergraph model [6,7] where the objective and balance accurately captures minimization of communication volume respecting to the computational load balance on processors. In this section, we present the three parallel multiplication schemes and the associated hypergraph models.

5.1.1 Row-parallel Sparse Matrix Vector Multiplication

Let K be the number of processors. Given a sparse matrix A, we obtain a decomposition $A = A^r + A^d$, where A^r and A^d hold a separate subset of nonzeros, and A^r is partitioned rowwise.

$$A^r = \begin{bmatrix} A^r_{1,*} \\ A^r_{2,*} \\ \vdots \\ A^r_{K,*} \end{bmatrix} = \begin{bmatrix} 0 & A^r_{1,2} & \dots & A^r_{1,K} \\ A^r_{2,1} & 0 & \dots & A^r_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ A^r_{K,1} & A^r_{K,2} & \dots & 0 \end{bmatrix}, \qquad A^d = \begin{bmatrix} A^d_1 & & & \\ & A^d_2 & & \\ & & \ddots & \\ & & & A^d_K \end{bmatrix}.$$

For a given decomposition $A = A^r + A^d$, each processor P_k holds the row stripe $A^r_{k,*}$ and the diagonal A^d_k. The row and column partition is obtained with respect to rowwise partition. In parallel matrix vector multiplication, each processor P_k executes the following steps:

1. For each nonzero off-diagonal block $A^r_{\ell,k}$, form sparse vector $\hat{x}^\ell_k$, which contains only those entries of x_k corresponding to the nonzero columns in $A^r_{\ell,k}$.

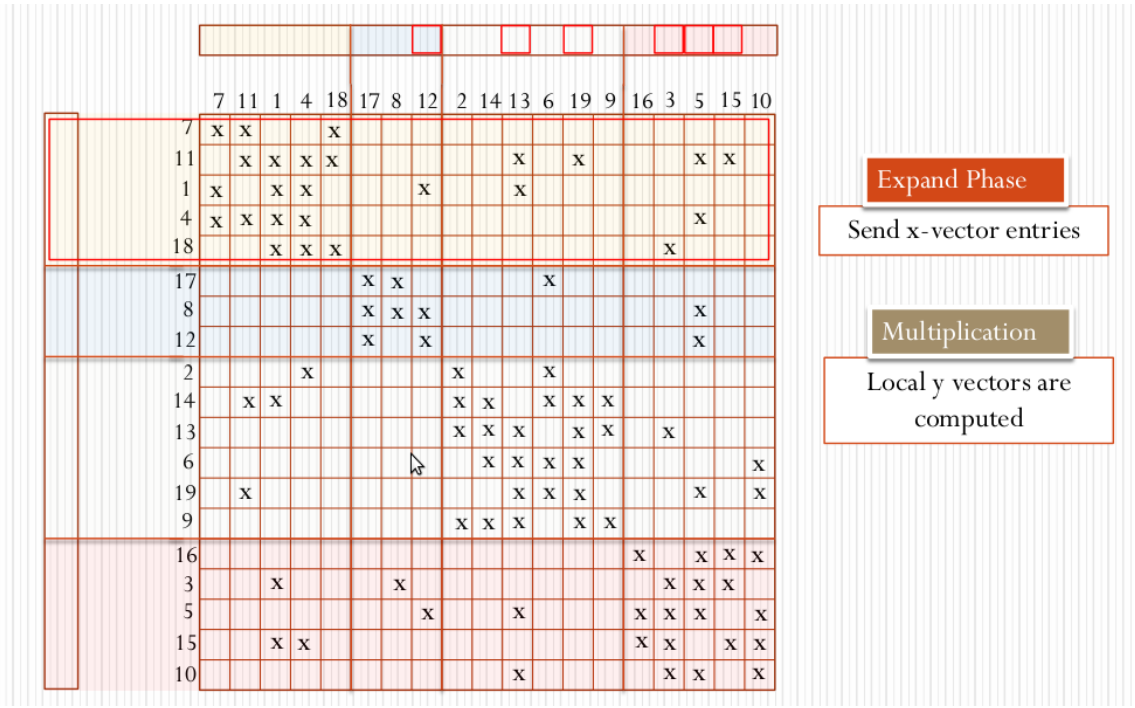


Figure 5.1: Row-parallel sparse matrix vector multiplication.

2. (Expand) Send $[\hat{x}_k^\ell]$ to processor P_ℓ.

3. Compute the diagonal block product $y_k^k = A_k^d \times x_k$, and set $y_k = y_k^k$.

4. For each nonzero off-diagonal block $A_{k,\ell}^r$, receive $[\hat{x}_\ell^k]$ from processor P_ℓ, then compute $y_k^\ell = A_{k,\ell}^r \times \hat{x}_\ell^k$, and update $y_k = y_k + y_k^\ell$

Figure 5.1 illustrates row-parallel matrix vector multiplication on a sample 4-way rowwise partitioned matrix. As seen in the figure, Processor 1 requires input-vector entries x_{12} from Processor 2, x_{13}, x_{19} from Processor 3, and x_3, x_5, x_{15} from Processor 4 in the expand phase.

The column-net hypergraph $\mathcal{H}_{\text{CN}}(A)$ of the matrix A is constructed as follows. We introduce a node u_i and a net n_j in $\mathcal{H}_{\text{CN}}(A)$ for each row r_i and column c_j in A, respectively. Each net n_j connects nodes in $\{u_k : a_{kj} \neq 0\}$. Each node u_i has a weight $|\{\ell : a_{i\ell} \neq 0\}|$ and each net has a unit cost. As a result, partitioning the column-net hypergraph $\mathcal{H}_{\text{CN}}(A)$ minimizing the cutsize according to the connectivity metric regarding to node-weight balance constraint corresponds to row-wise

partitioning of A minimizing communication volume regarding to computation load balance of processors in row-parallel sparse matrix vector multiplication.

5.1.2 Column-parallel Sparse Matrix Vector Multiplication

Let K be the number of processors. Given a sparse matrix A, we obtain a decomposition $A = A^c + A^d$, where A^c and A^d hold a separate subset of nonzeros, $A^c = \left[A^c_{*,1}, A^c_{*,2}, \ldots, A^c_{*,K}\right]$ is partitioned columnwise,

$$A^c = \begin{bmatrix} 0 & A^c_{1,2} & \ldots & A^c_{1,K} \\ A^c_{2,1} & 0 & \ldots & A^c_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ A^c_{K,1} & A^c_{K,2} & \ldots & 0 \end{bmatrix}, \quad A^d = \begin{bmatrix} A^d_1 & & & \\ & A^d_2 & & \\ & & \ddots & \\ & & & A^d_K \end{bmatrix}.$$

position $A = A^c + A^d$, each processor P_k holds the column stripe $A^c_{*,k}$, and the diagonal A^d_k. The row and column partition is obtained with respect to columnwise partition. In parallel matrix vector multiplication, each processor P_k executes the following steps:

1. For each nonzero off-diagonal block $A^c_{\ell,k}$, form sparse vector $\hat{y}^k_\ell$, which contains only those results of $y^k_\ell = A^c_{\ell,k} \times x_k$ corresponding to the nonzero rows in $A_{\ell,k}$.
2. (Fold) Send $[\hat{y}^k_\ell]$ to processor P_ℓ.
3. Compute the diagonal block product $y^k_k = A^d_k \times x_k$, and set $y_k = y^k_k$.
4. For each nonzero off-diagonal block $A^c_{k,\ell}$, receive $[\hat{y}^\ell_k]$ from processor P_ℓ, then compute $y^\ell_k = \hat{y}^\ell_k$, and update $y_k = y_k + y^\ell_k$.

Figure 5.2 illustrates column-parallel matrix vector multiplication on a sample 4-way columnwise partitioned matrix. As seen in the figure, Processor 1 requires output-vector entries, that is the partial results, of y_{11} from Processor 3 and 4, of y_1 from Processor 2 and 3, of y_4 and y_{18} only from Processor 4, in the fold phase.

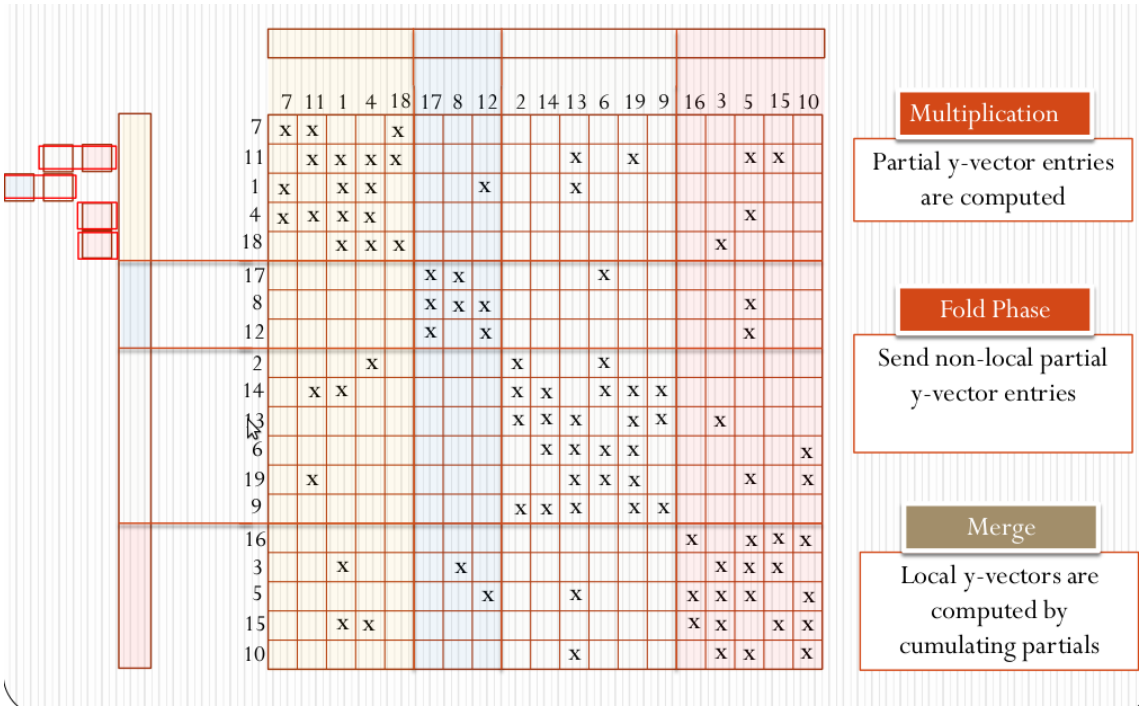


Figure 5.2: Column-parallel sparse matrix vector multiplication.

The row-net hypergraph $\mathcal{H}_{\mathrm{RN}}(A)$ of the matrix A is constructed as follows. We introduce a node u_i and a net n_j in $\mathcal{H}_{\mathrm{RN}}(A)$ for each column c_i and row r_j in A, respectively. Each net n_j connects nodes in $\{u_k : a_{jk} \neq 0\}$. Each node u_i has a weight $|\{\ell : a_{\ell i} \neq 0\}|$, and each net has a unit cost. As a result, partitioning the row-net hypergraph $\mathcal{H}_{\mathrm{RN}}(A)$ with the objective of minimizing the cutsize according to the connectivity metric while satisfying node-weight balance constraint corresponds to columnwise partitioning of A with the objective of minimizing communication volume while satisfying to computation load balance constraint of processors in column-parallel sparse matrix vector multiplication.

5.1.3 Row-column-parallel Sparse Matrix Vector Multiplication

Let K be the number of processors. Given a sparse matrix A, we obtain a K-way decomposition $A = A^1 + A^2 + \ldots + A^K$, so called *fine-grain partition*, where each

A^k holds a separate subset of nonzeros,

$$A^k = \begin{bmatrix} A^k_{1,1} & A^k_{1,2} & \dots & A^k_{1,k} & \dots & A^k_{1,K} \\ A^k_{2,1} & A^k_{2,2} & \dots & A^k_{2,k} & \dots & A^k_{2,K} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ A^k_{k,1} & A^k_{k,2} & \dots & A^k_{k,k} & \dots & A^k_{k,K} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ A^k_{K,1} & A^k_{K,2} & \dots & A^k_{K,k} & \dots & A^k_{K,K} \end{bmatrix} \tag{5.2}$$

For a given decomposition $A = A^1 + A^2 + \ldots + A^k$, each processor P_k holds the submatrix A^k. For a given row and column partition, in parallel matrix vector multiplication, each processor P_k executes the following steps:

1. For each nonzero column-stripe $[A^\ell_{*,k}]$, form sparse vector $\hat{x}^\ell_k$, which contains only those entries of x_k corresponding to the nonzero columns in $[A^\ell_{*,k}]$.
2. (Expand) Send $[\hat{x}^\ell_k]$ to processor P_ℓ.
3. Compute the product $y^k_k = [A^k_{*,k}] \times x_k$, and set $y^k = y^k_k$.
4. For each nonzero column-stripe $[A^k_{*,\ell}]$, receive $[\hat{x}^k_\ell]$ from processor P_ℓ, then compute the product $y^k_\ell = [A^k_{*,\ell}] \times \hat{x}^k_\ell$, and update $y^k = y^k + y^k_\ell$.
5. For each nonzero row-stripe $[A^k_{\ell,*}]$, form sparse vector $\hat{y}^k_\ell$, which contains only those results of y^k_ℓ corresponding to the nonzero rows in $A^k_{\ell,*}$.
6. (Fold) Send $[\hat{y}^k_\ell]$ to processor P_ℓ
7. For each nonzero off-diagonal block $[A^\ell_{k,*}]$, receive $[\hat{y}^\ell_k]$ from processor P_ℓ, then compute $y^\ell_k = \hat{y}^\ell_k$, and update $y_k = y_k + y^\ell_k$.

Figure 5.3 illustrates row-column-parallel matrix vector multiplication on a sample 4-way fine-grain partitioned matrix. As seen in the figure, Processor 1 requires input-vector entries x_2, x_{19} from Processor 3, and x_{16}, x_5 from Processor 4 in the expand phase. Similarly, Processor 1 requires output-vector entries, that is the partial results, of $y_{11}, y_4, y_8, y_{12}, y_2, y_{14}, y_{13}, y_9, y_{15}$ from other processors in

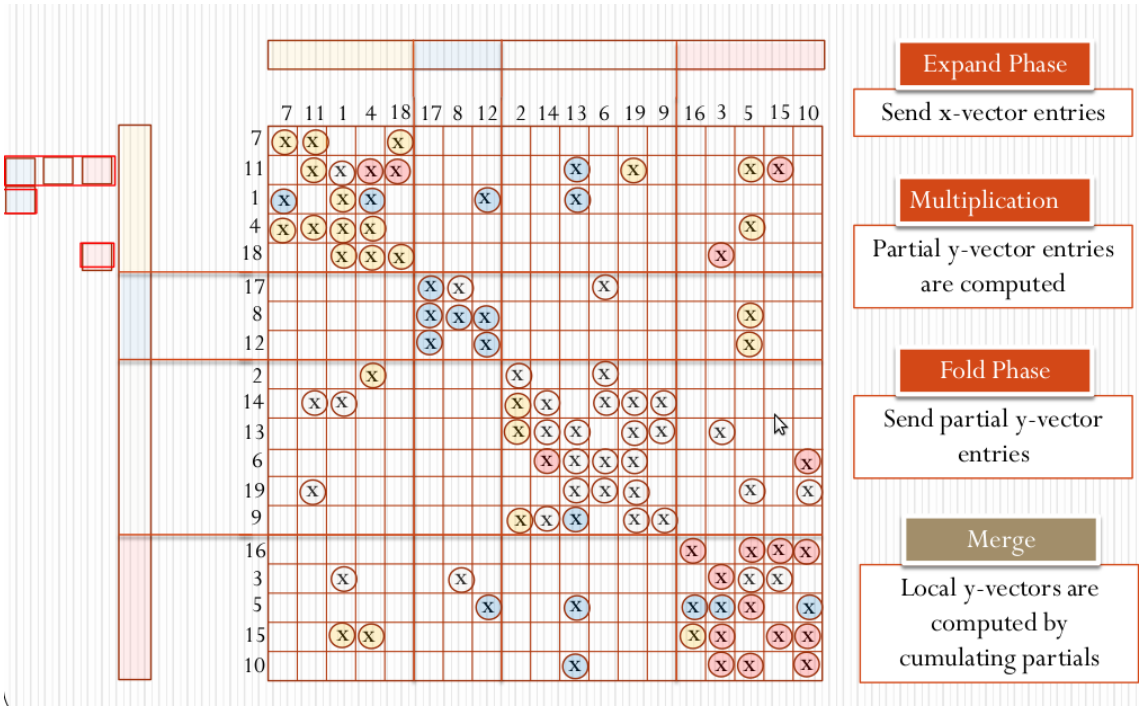


Figure 5.3: Row-column-parallel sparse matrix vector multiplication.

the fold phase. Note that partial results of y_{11} and y_{15} are required from multiple parts.

The row-column-net hypergraph $\mathcal{H}_{\mathrm{RCN}}(A)$ of the matrix A is constructed as follows. We introduce a node u_{ij} for each nonzero a_{ij}, and two nets n_i^r and n_i^c for each row and column, respectively. Each net n_i^r connects nodes in $\{u_{ij} : a_{ij} \neq 0\}$, and each net n_j^c connects nodes in $\{u_{ij} : a_{ij} \neq 0\}$. Each node has a unit weight and each net has a unit cost. As a result, partitioning the row-column-net hypergraph $\mathcal{H}_{\mathrm{RCN}}(A)$ with the objective of minimizing the cutsize according to the connectivity metric while satisfying node-weight balance constraint corresponds to nonzero partitioning of A with the objective of minimizing communication volume while satisfying to computation load balance constraint of processors in row-column-parallel sparse matrix vector multiplication.

5.2 Single-phased Row-column-parallel Sparse Matrix Vector Multiplication

In this section, we define the proposed row-column-parallel sparse matrix vector multiplication scheme that requires single communication phase. This scheme can also be interpreted as row-column-parallel sparse matrix vector multiplication where expand and fold phases are merged.

Let K be the number of processors. Given a sparse matrix A, we obtain a decomposition $A = A^r + A^d + A^c$, also referred as row-columnwise partition of A, where A^r, A^d and A^c holds a separate subset of nonzeros, A^r and A^c are partitioned rowwise and columnwise, respectively,

$$A^r = \begin{bmatrix} A^r_{1,*} \\ A^r_{2,*} \\ \vdots \\ A^r_{K,*} \end{bmatrix} = \begin{bmatrix} 0 & A^r_{1,2} & \dots & A^r_{1,K} \\ A^r_{2,1} & 0 & \dots & A^r_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ A^r_{K,1} & A^r_{K,2} & \dots & 0 \end{bmatrix}, \qquad A^d = \begin{bmatrix} A^d_1 & & & \\ & A^d_2 & & \\ & & \ddots & \\ & & & A^d_K \end{bmatrix},$$

$$A^c = \begin{bmatrix} A^c_{*,1}, A^c_{*,2}, \dots, A^c_{*,K} \end{bmatrix} = \begin{bmatrix} 0 & A^c_{1,2} & \dots & A^c_{1,K} \\ A^c_{2,1} & 0 & \dots & A^c_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ A^c_{K,1} & A^c_{K,2} & \dots & 0 \end{bmatrix}. \tag{5.3}$$

For a given decomposition $A = A^r + A^d + A^c$, each processor P_k holds the row stripe $A^r_{k,*}$, the column stripe $A^c_{*,k}$, and the diagonal A^d_k. The aforementioned partition can be decoded as a nonzero partition $A = A^1 + A^2 + \ldots + A^k$ where for each submatrix A^k; $A^k_{\ell,m} = 0$ for $k \notin \{\ell, m\}$, i.e.,

$$A^k = \begin{bmatrix} 0 & 0 & \dots & A^k_{1,k} & \dots & 0 \\ 0 & 0 & \dots & A^k_{2,k} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ A^k_{k,1} & A^k_{k,2} & \dots & A^k_{k,k} & \dots & A^k_{k,K} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & A^k_{K,k} & \dots & 0 \end{bmatrix} \tag{5.4}$$

Thus, row-columnwise partition can be refered as a nonzero partition where each task of nonzero multiplication $a_{ij} \times x_j$ is held by either the processor that holds input-vector entry x_j, or the processor that is responsible for the computation of output-vector entry y_i, where such property is refered here as *consistency property*. The row and column partition is obtained with respect to row-columnwise partition. In parallel matrix vector multiplication, each processor P_k executes the following steps:

1. For each nonzero off-diagonal block $A^r_{\ell,k}$, form sparse vector $\hat{x}^\ell_k$, which contains only those entries of x_k corresponding to the nonzero columns in $A^r_{\ell,k}$.
2. For each nonzero off-diagonal block $A^c_{\ell,k}$, form sparse vector $\hat{y}^k_\ell$, which contains only those results of $y^k_\ell = A^c_{\ell,k} \times x_k$ corresponding to the nonzero rows in $A^c_{\ell,k}$.
3. Send $[\hat{x}^\ell_k, \hat{y}^k_\ell]$ to processor P_ℓ.
4. Compute the diagonal block product $y^k_k = A^d_k \times x_k$, and set $y_k = y^k_k$.
5. For each nonzero off-diagonal block $(A^r_{k,\ell}+A^c_{k,\ell})$, receive $[\hat{x}^k_\ell, \hat{y}^\ell_k]$ from processor P_ℓ, then compute $y^\ell_k = \hat{y}^\ell_k + A^r_{k,\ell} \times \hat{x}^k_\ell$, and update $y_k = y_k + y^\ell_k$.

Figure 5.4 illustrates single-phased row-column-parallel matrix vector multiplication on a sample 4-way row-columnwise partitioned matrix. As seen in the figure, Processor 1 requires input-vector entries x_{13}, x_{19}, x_5 from other processors, requires partial results of y_{11} from Processor 4, of y_1 from Processors 2 and 3, and of y_18 from Processor 4. Note that these input- and output-vector entries are communicated together in the single communication phase.

5.3 The Hypergraph Model

Let $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ be a hypergraph. The hypergraph partitioning problem is finding a K-way node partition $\Pi(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ that satisfies the balance criterion. In dependent hypergraph partitioning problem, there is one additional

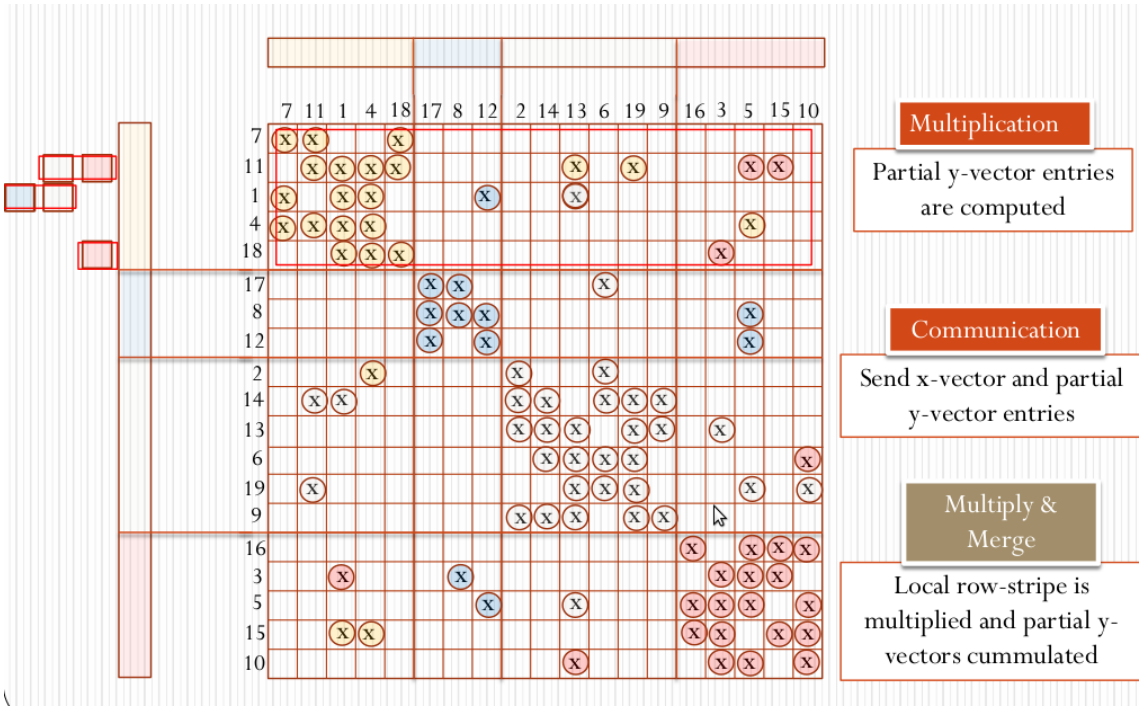


Figure 5.4: Single-phased row-column-parallel sparse matrix vector multiplication.

constraint which is referred as the *cooccurance constraint*. We are given a function $\mathcal{C} : \mathcal{U} \rightarrow 2^{\mathcal{U}} - \emptyset$. The cooccurance constraint requires that $\pi(u_i) \in \{\pi(u_k) : u_k \in \mathcal{C}(u_i)\}$, for each $u_i \in \mathcal{U}$, where $\pi(u_\ell) = k : u_\ell \in \mathcal{U}_k$.

Definition 6 (Hypergrph Partitioning with Cooccurence (HPc) Prob.) *Given a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$, an integer K, a cooccurence function $\mathcal{C} : \mathcal{U} \rightarrow 2^{\mathcal{U}} - \emptyset$, and a maximum allowable imbalance ratio ϵ, the hypergraph partitioning with cooccurence (HPc) problem is finding a K-way node partition $\Pi_{\mathcal{U}}(\mathcal{H}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ that both satisfies the ϵ-balance and the $\mathcal{C}$-cooccurance criteria while minimizing the cutsize.*

The extended row-column-net hypergraph $\widehat{\mathcal{H}}_{\text{RCN}}(A)$ of the matrix A is constructed as follows. We introduce a node u_{ij} for each nonzero a_{ij}, a row node u_i^r and a row net n_i^r for each row r_i, and a column node u_i^c and a column net n_i^c for each column c_i. Each net n_i^r connects nodes in $\{u_{ij} : a_{ij} \neq 0\}$, and each net n_j^c connects nodes in $\{u_{ij} : a_{ij} \neq 0\}$. The row and column nodes are weightless,

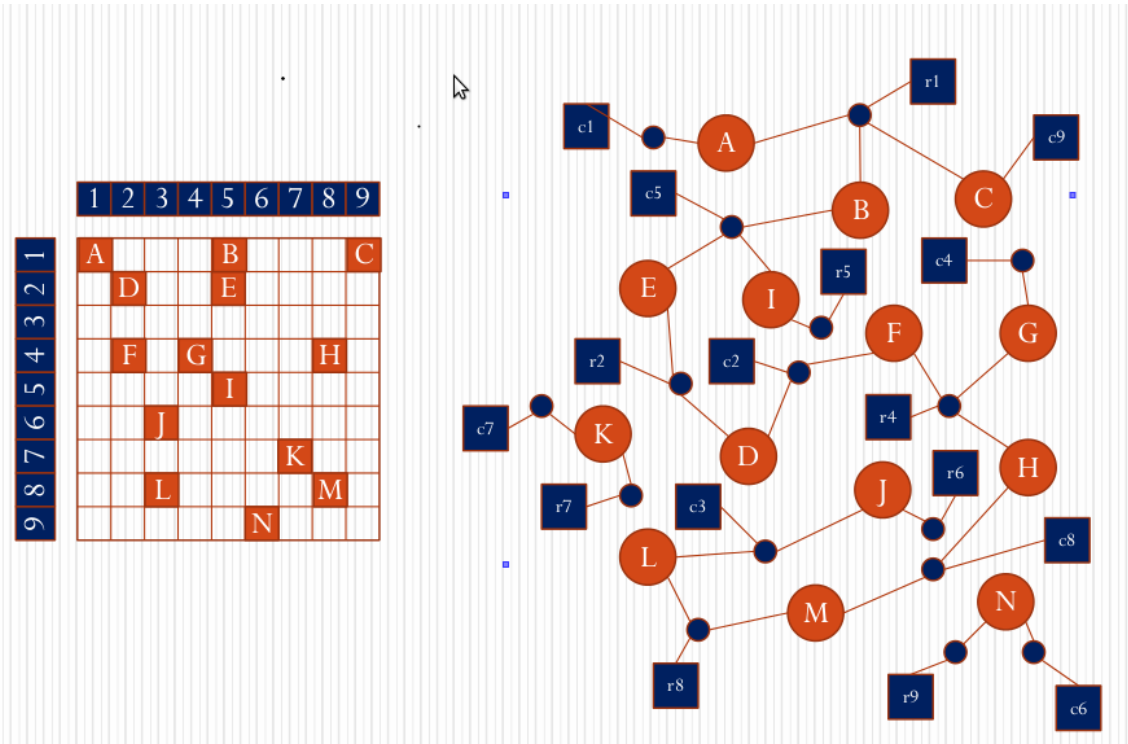


Figure 5.5: A sample matrix A and the corresponding extended row-columnet hypergraph $\widehat{\mathcal{H}}_{\text{RCN}}(A)$.

whereas each net has a unit cost. We construct cooccurance relation $\mathcal{C}$ as,

$$\mathcal{C}(u_{ij}) = \{u_i^r, u_j^c\}, \quad \forall u_{ij} \in \widehat{\mathcal{H}}_{\text{RCN}}(A) \tag{5.5}$$

$$\mathcal{C}(u_i^r) = \{u_i^c\}, \quad \forall 1 \leq i \leq n \tag{5.6}$$

$$\mathcal{C}(u_i^c) = \{u_i^r\}, \quad \forall 1 \leq i \leq n \tag{5.7}$$

Equation 5.5 ensures that a nonzero partition possesses the consistency property, whereas Equations 5.6 and 5.7 together reinforce the symmetricity of the partition.

As a result, partitioning the extended row-column-net hypergraph $\widehat{\mathcal{H}}_{\text{RCN}}(A)$ with the objective of minimizing the cutsize according to the connectivity metric while satisfying node-weight balance constraint corresponds to row-columnwise partitioning of A with the objective of minimizing communication volume while satisfying to computation load balance constraint of processors in single-phased row-column-parallel sparse matrix vector multiplication.

5.4 Row-columnwise Partitioning Framework

We propose a two-step solution framework to find a row-columnwise partitioning to a given $n \times n$ sparse symmetric matrix A. In the first step, we obtain a K-way partition $\Pi(\mathcal{H}_{\text{CN}}) = \{\mathcal{U}_1^R, \mathcal{U}_2^R, \ldots, \mathcal{U}_K^R\}$ of the column-net hypergraph $\mathcal{H}_{\text{CN}}$ of A. Then, we decode $\Pi(\mathcal{H}_{\text{CN}})$ as a K-way partition $\Pi(\widehat{\mathcal{H}}_{\text{RCN}}) = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of the extended row-column hypergraph $\widehat{\mathcal{H}}_{\text{RCN}}$ as,

$$\mathcal{U}_k = \{u_{ij} : u_i^R \in \mathcal{U}_k^R\} \cup \{u_i^r, u_i^c : u_i^R \in \mathcal{U}_k^R\}, \quad \forall 1 \leq k \leq K. \tag{5.8}$$

In the second step, we refine the partititon $\Pi(\widehat{\mathcal{H}}_{\text{RCN}})$ with respect to $\mathcal{C}$-cooccurence criterion. First, we should define pair selecting problem.

Definition 7 (Pair Selecting Problem) *Given an integer K, a set $\mathcal{P} \subset \{(k,\ell) : 1 \leq k, \ell \leq K\}$, a weight function $W : \mathcal{P} \rightarrow \mathbb{R}^+$, a value $V : \mathcal{P} \rightarrow \mathbb{R}^+$ function, an initial weight $w_k \in \mathbb{R}^+$ for each $1 \leq k \leq K$, and a value $W_{max} \in \mathbb{R}^+$, find a subset $\mathcal{P}' \subset \mathcal{P}$ such that*

$$\max_{1 \leq k \leq K} \{w_k + \sum_{(\ell,k) \in \mathcal{P}'} W(\ell,k) - \sum_{(k,\ell) \in \mathcal{P}'} W(k,\ell)\} \leq W_{max}$$

and the total gain $\Phi(\mathcal{P}')$ is maximized, where

$$\Phi(\mathcal{P}') = \sum_{(k,\ell) \in \mathcal{P}'} V(\ell,k).$$

Let $\mathcal{V}_{k,\ell} = \{u_{ij} \in \widehat{\mathcal{H}}_{\text{RCN}} : u_i^r \in \mathcal{U}_k, u_j^c \in \mathcal{U}_\ell\}$ for all $1 \leq k, \ell \leq K$. Let $\mathcal{P} = \{(k,\ell) : k \neq \ell, \ \mathcal{V}_{k,\ell} \neq \emptyset\}$. Let the off-diagonal block $A_{k,\ell}$ refers to the submatrix regarding to $\mathcal{V}_{k,\ell}$. Then, we find a subset $\mathcal{V}'_{k,\ell} \subset \mathcal{V}_{k,\ell}$, for each pair $(k,\ell) \in \mathcal{P}$ using Dulmage-Mendhelson decomposition on the submatrix $A_{k,\ell}$ as

$$A_{k,\ell} = \begin{bmatrix} 0 & A_{k,\ell}^h & A_{k,\ell}^{hs} & A_{k,\ell}^{hv} \\ 0 & 0 & A_{k,\ell}^s & A_{k,\ell}^{sv} \\ 0 & 0 & 0 & A_{k,\ell}^v \\ 0 & 0 & 0 & 0 \end{bmatrix}, \tag{5.9}$$

where $A^h_{k,\ell}$, $A^s_{k,\ell}$, and $A^v_{k,\ell}$ represents the horizontal, square, and vertical blocks, respectively. The horizontal block $A^h_{k,\ell}$ is used to construct $\mathcal{V}'_{k,\ell}$ as

$$\mathcal{V}'_{k,\ell} = \{u_{ij} : a_{ij} \in A^h_{k,\ell}\}, \quad \forall (k,\ell) \in \mathcal{P} \tag{5.10}$$

We define the initial weight of a part in pair selecting problem as

$$w_k = W(\mathcal{U}_k) = |\{u_{ij} \in \widehat{\mathcal{H}}_{\mathrm{RCN}}\}|, \quad \forall 1 \leq k \leq K \tag{5.11}$$

and the values and weights of pairs as

$$V(k,\ell) = |\{c_j : u_{ij} \in V'_{k,\ell}\}| - |\{r_i : u_{ij} \in V'_{k,\ell}\}| \tag{5.12}$$

$$W(k,\ell) = |\mathcal{V}'_{k,\ell}| \tag{5.13}$$

Note that $V'_{k,\ell}$ is optimum in the sense that $V(k,\ell)$ has the maximum value among all possible subsets of $V_{k,\ell}$, for each $(k,\ell) \in \mathcal{P}$, due to the property of Dulmage-Mendhelson decomposition. We set $W_{max} = (1+\epsilon)|\mathcal{U}_A| \ /K$. Finally, we decode a solution to the pair selecting problem as a final row-columnwise partition $\Pi^f(\mathcal{H}_A) = \{\mathcal{U}^f_1, \mathcal{U}^f_2, \ldots, \mathcal{U}^f_K\}$ as,

$$\mathcal{U}^f_k = \mathcal{U}_k \cup \{\mathcal{V}'_{\ell,k} : (\ell,k) \in \mathcal{P}'\} - \{\mathcal{V}'_{k,\ell} : (k,\ell) \in \mathcal{P}'\}, \quad \forall 1 \leq k \leq K. \tag{5.14}$$

5.5 Conclusion

In this chapter, we defined a novel parallel matrix vector multiplication scheme which requires only one communication phase, and thus decreasing the latency with respect to row-column-parallel vector multiplication scheme, and a corresponding matrix partitioning method, called row-columnwise partitioning, which does not restrict the solution space too much as in coarse-grain partitionings. Secondly, we defined a novel version of hypergraph partitioning which has very promising modeling power, and model the row-columnwise partitioning problem as hypergraph partitioning with cooccurence. Lastly, we give a row-columnwise partitioning solution that is based on one-dimensional partitioning and relaxing the assignment of off-diagonal nonzeros with the use of Dulmage-Mendhelson

decomposition on those off-diagonal blocks. With Dulmage-Mendhelson decomposition we guarantee the optimality of selecting a subset of nonzeros, whose multiplication tasks will be reassigned to sender processor, of a off-diagonal block in terms of minimizing communication volume.

Chapter 6

Conclusion and Future Research

The main theme of this thesis is hypergraph, more specifically hypergraph partitioning. Since hypergraphs are generalization of graphs, in the literature there are many success stories those employing hypergraphs instead of graphs. The contribution of this thesis is two folds. One is an effective and efficient implementation of a hypergraph partition tool that runs faster than the previous tools as our tool make use of graph partitioning. The second contribution is to show the modeling power of hypergraphs in two data partitioning problems encountered on two separate domains of scientific and parallel computing.

In Chapter 3, we investigated and effective implementation of hypergraph partitioning using recursive graph bipartioning and encountered very promising results. We believe that the success of the proposed methods point to several future research directions. First, better vertex weighting schemes to approximate the node balance is an area that can make a significant impact. We believe exploiting domain specific information or devising techniques that can apply to certain classes of graphs, as opposed to constructing generic approximations that can work for all graphs, is a promising avenue to explore. Secondly, the algorithms we have used in this study, were only slightly adjusted for the particular problem we were solving. There is a lot of room for improvements in algorithms for finding vertex separators with balanced hypergraph partitions, and we believe

these algorithms can be designed and implemented within the existing partitioning graph partitioning frameworks, which means strong algorithmic ideas can be translated into effective software tools with relatively little effort. Finally, this study is only an example of the growing importance of graph partitioning and the need for more flexible models for graph partitioning. Graph partitioning now is an internal step for divide-and-conquer based methods, whose popularity will only increase with the growing problem sizes. As such, requirements for graph partitioning will keep growing and broadening. While, the state of the art for graph partitioning has drastically improved from the days of merely minimizing the number of cut edges, we believe there is still a lot of room for growth for more general models for graph partitioning.

Chapters 4 and 5 propose hypergraph partitioning models for data partitioning on the domains of query processing and sparse matrix vector multiplication, respectively. Despite of the fact that these two domains are very different from each other, the data partitioning problems in those domains both can be powerfully modeled by hypergraphs. Especially, our emprical study in Chapter 4 supported the use of hypergraphs to partition inverted-lists among to processors. A possible extension to our work is a multi-constraint model, where the storage load imbalance can also be formulated as a constraint within the model. Another direction is to replace the replication heuristics we evaluated in this study with the recently proposed techniques that couple hypergraph partitioning with replication. This may lead to further reduction in communication costs. We propose a novel and more powerful hypergraph model in Chapter 5. As a research direction, we may suggest an implementation of this model and investigation of modeling other important problems in scientific computing with hypergraph partitioning with cooccurences.

Bibliography

[1] U. V. Çatalyürek and C. Aykanat, "Decomposing irregularly sparse matrices for parallel matrix-vector multiplications," in *Proceedings of 3rd International Symposium on Solving Irregularly Structured Problems in Parallel, Irregular'96*, vol. 1117 of *Lecture Notes in Computer Science*, pp. 75–86, Springer-Verlag, 1996.

[2] A. Pınar, U. V. Çatalyürek, C. Aykanat, and M. Pınar, "Decomposing linear programs for parallel solution," *Lecture Notes in Computer Science*, vol. 1041, pp. 473–482, 1996.

[3] K. Akbudak, E. Kayaaslan, and C. Aykanat, "Hypergraph partitioning based models and methods for exploiting cache locality in sparse matrix-vector multiplication," *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. C237–C262, 2013.

[4] R. H. Bisseling and I. Flesch, "Mondriaan sparse matrix partitioning for attacking cryptosystems by a parallel block Lanczos algorithm: a case study," *Parallel Computing*, vol. 32, no. 7, pp. 551–567, 2006.

[5] B. B. Cambazoglu and C. Aykanat, "Hypergraph-partitioning-based remapping models for image-space-parallel direct volume rendering of unstructured grids," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, pp. 3–16, Jan 2007.

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

[7] U. 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.

[8] U. V. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *ACM/IEEE SC2001*, (Denver, CO), November 2001.

[9] U. V. Çatalyürek, C. Aykanat, and B. Ucar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 656–683, 2010.

[10] C. Chang, T. Kurc, A. Sussman, U. V. Çatalyürek, and J. Saltz, "A hypergraph-based workload partitioning strategy for parallel data aggregation," in *Proceedings of the Eleventh SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Mar. 2001.

[11] N. Dingle, P. Harrison, and W. Knottenbelt, "Uniformization and Hypergraph Partitioning for the Distributed Computation of Response Time Densities in Very Large Markov Models," *Journal of Parallel and Distributed Computing*, vol. 64, pp. 908–920, August 2004.

[12] K. Kaya and C. Aykanat, "Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 8, pp. 883–896, 2006.

[13] K. Kaya, B. Uçar, and C. Aykanat, "Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories," *Journal of Parallel and Distributed Computing*, vol. 67, no. 3, pp. 271–285, 2007.

[14] G. Khanna, N. Vydyanathan, T. Kurc, U. Catalyurek, P. Wyckoff, J. Saltz, and P. Sadayappan, "A hypergraph partitioning based approach for scheduling of tasks with batch-shared I/O," in *Proceedings of the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, May 2005.

[15] M. M. Strout and P. D. Hovland, "Metrics and models for reordering transformations," in *Proc. of the Second ACM SIGPLAN Workshop on Memory System Performance (MSP04)*, (Washington DC.), pp. 23–34, ACM, June 2004.

[16] G. Szederknyi, "Computing sparse and dense realizations of reaction kinetic systems," *Journal of Mathematical Chemistry*, vol. 47, pp. 551–568, 2010. 10.1007/s10910-009-9525-5.

[17] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1837–1859, 2004.

[18] B. Uçar and C. Aykanat, "Partitioning sparse matrices for parallel preconditioned iterative methods," *SIAM Journal on Scientific Computing*, vol. 29, pp. 1683–1709, June 2007.

[19] B. Uçar and C. Aykanat, "Revisiting hypergraph models for sparse matrix partitioning," *SIAM Review*, vol. 49, no. 4, pp. 595–603, 2007.

[20] B. Uçar, C. Aykanat, M. C. Pinar, and T. Malas, "Parallel image restoration using surrogate constraint methods," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 186–204, 2007.

[21] 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.

[22] v. Vrba, H. Espeland, P. Halvorsen, and C. Griwodz, "Limits of work-stealing scheduling," *Job Scheduling Strategies for Parallel Processing: 14th International Workshop, JSSPP 2009, Rome, Italy, May 29, 2009. Revised Papers*, pp. 280–299, 2009.

[23] A. N. Yzelman and R. H. Bisseling, "Cache-oblivious sparse matrix-vector multiplication by using sparse matrix partitioning methods," *SIAM Journal on Scientific Computing*, vol. 31, no. 4, pp. 3128–3154, 2009.

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

[25] U. 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://bmi.osu.edu/~umit/software.htm`, 1999.

[26] A. Trifunovic and W. J. Knottenbelt, "Parkway 2.0: A parallel multilevel hypergraph partitioning tool," in *Proc. 19th International Symposium on Computer and Information Sciences (ISCIS 2004)*, vol. 3280 of *LNCS*, pp. 789–800, Springer, 2004.

[27] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and U. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2006.

[28] E. Ihler, D. Wagner, and F. Wagner, "Modeling hypergraphs by graphs with the same mincut properties," *Information Processing Letters*, vol. 45, pp. 171–175, March 1993.

[29] P. K. Chan, D. F. Schlag, and J. Y. Zien, "Spectral K-way ratio-cut partitioning and clustering," in *Proceedings of the 30th Design Automation Conference*, pp. 749–754, ACM/IEEE, 1993.

[30] S. W. Hadley, B. L. Mark, and A. Vanelli, "An efficient eigenvector approach for finding netlist partitions," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 885–892, July 1992.

[31] T. N. Bui and C. Jones, "Finding good approximate vertex and edge partitions is NP-hard," *Infomation Processing Letters*, vol. 42, pp. 153–159, May 1992.

[32] A. Pothen, H. D. Simon, and K.-P. Liou, "Partitioning sparse matrices with eigenvectors of graphs," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 3, pp. 430–452, 1990.

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

[34] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *VLSI Journal*, vol. 19, no. 1–2, pp. 1–81, 1995.

[35] C. Aykanat, A. Pınar, and U. V. Çatalyürek, "Permuting sparse rectangular matrices into block-diagonal form," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1860–1879, 2004.

[36] A. Pınar and C. Aykanat, "An effective model to decompose linear programs for parallel solution," in *Proceedings of the Third International Workshop on Applied Parallel Computing, PARA'96*, vol. 1184 of *Lecture Notes in Computer Science*, pp. 592–601, Springer-Verlag, 1997.

[37] A. Pinar and B. Hendrickson, "Partitioning for complex objectives," in *Parallel and Distributed Processing Symposium., Proceedings 15th International*, pp. 1232 –1237, apr 2001.

[38] A. Pinar, E. Chow, and A. Pothen, "Combinatorial techniques for constructing sparse null-space bases," *Electronic Transactions on Numerical Analysis*, vol. 22, pp. 122–145, 2006. special volume on saddle point problems: numerical solution and applications.

[39] E. Kayaaslan, A. Pinar, Ü. Çatalyürek, and C. Aykanat, "Partitioning hypergraphs in scientific computing applications through vertex separators on graphs," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. 970–992, 2012.

[40] S. Sahni, "General techniques for combinatorial approximation," *Operations Research*, vol. 25, pp. 920–936, Nov. 1977.

[41] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proceedings of the 19th ACM/IEEE Design Automation Conference*, pp. 175–181, 1982.

[42] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell System Technical Journal*, vol. 49, pp. 291–307, Feb. 1970.

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

[44] J. A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, pp. 345–363, Apr 1973.

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

[46] T. Davis, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, 2011.

[47] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, 2006.

[48] B. A. Ribeiro-Neto and R. A. Barbosa, "Query performance for tightly coupled distributed digital libraries," in *Proceedings of the 3rd ACM Conference on Digital Libraries*, pp. 182–190, 1998.

[49] A. MacFarlane, J. A. McCann, and S. E. Robertson, "Parallel search using partitioned inverted files," in *Proceedings of the 7th International Symposium on String Processing and Information Retrieval*, pp. 209–220, 2000.

[50] C. Badue, B. Ribeiro-Neto, R. Baeza-Yates, and N. Ziviani, "Distributed query processing using partitioned inverted files," in *Proceedings of the 8th International Symposium on String Processing and Information Retrieval*, pp. 10–20, 2001.

[51] B. B. Cambazoglu, A. Catal, and C. Aykanat, "Effect of inverted index partitioning schemes on performance of query processing in parallel text retrieval systems," in *Computer and Information Sciences* (A. Levi, E. Savas, H. Yenigün, S. Balcisoy, and Y. Saygin, eds.), vol. 4263 of *Lecture Notes in Computer Science*, pp. 717–725, Springer Berlin / Heidelberg, 2006.

[52] A. Tomasic and H. Garcia-Molina, "Performance of inverted indices in shared-nothing distributed text document information retrieval systems," in

Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems, pp. 8–17, 1993.

[53] B.-S. Jeong and E. Omiecinski, "Inverted file partitioning schemes in multiple disk systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 2, pp. 142–153, 1995.

[54] C. S. Badue, R. Baeza-Yates, B. Ribeiro-Neto, A. Ziviani, and N. Ziviani, "Analyzing imbalance among homogeneous index servers in a web search system," *Information Processing & Management*, vol. 43, pp. 592–608, 2007.

[55] B. B. Cambazoglu and C. Aykanat, "A term-based inverted index organization for communication-efficient parallel query processing," in *Proceedings of IFIP International Conference on Network and Parallel Computing*, 2006.

[56] A. Moffat, W. Webber, and J. Zobel, "Load balancing for term-distributed parallel retrieval," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 348–355, 2006.

[57] C. Lucchese, S. Orlando, R. Perego, and F. Silvestri, "Mining query logs to optimize index partitioning in parallel web search engines," in *Proceedings of the 2nd International Conference on Scalable Information Systems*, pp. 43:1–43:9, 2007.

[58] J. Zhang and T. Suel, "Optimized inverted list assignment in distributed search engine architectures," in *Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, pp. 1–10, 2007.

[59] Q. Gan and T. Suel, "Improved techniques for result caching in web search engines," in *Proceedings of the 18th International Conference on World Wide Web*, pp. 431–440, 2009.

[60] B. B. Cambazoglu, F. P. Junqueira, V. Plachouras, S. Banachowski, B. Cui, S. Lim, and B. Bridge, "A refreshing perspective of search engine caching," in *Proceedings of the 19th International Conference on World Wide Web*, pp. 181–190, 2010.

[61] G. Skobeltsyn, F. Junqueira, V. Plachouras, and R. Baeza-Yates, "Resin: a combination of results caching and index pruning for high-performance web search engines," in *Proceedings of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 131–138, 2008.