HYPERGRAPH MODELS FOR SPARSE MATRIX PARTITIONING AND REORDERING

A DISSERTATION SUBMITTED TO
THE DEPARTMENT OF COMPUTER ENGINEERING AND
INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BİLKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Ümit V. Çatalyürek
November, 1999

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. 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.

Prof. Kemal Efe

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.

Asst. Prof. Attila Gürsoy

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. İrşadi Aksun

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.

Asst. Prof. Kıvanç Dinçer

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

HYPERGRAPH MODELS FOR SPARSE MATRIX PARTITIONING AND REORDERING

Ümit V. Çatalyürek
Ph.D. in Computer Engineering and Information Science
Supervisor: Assoc. Prof. Cevdet Aykanat
November, 1999

Graphs have been widely used to represent sparse matrices for various scientific applications including one-dimensional (1D) decomposition of sparse matrices for parallel sparse-matrix vector multiplication (SpMxV) and sparse matrix reordering for low fill factorization. The standard graph-partitioning based 1D decomposition of sparse matrices does not reflect the actual communication volume requirement for parallel SpMxV. We propose two computational hypergraph models which avoid this crucial deficiency of the graph model on 1D decomposition. The proposed models reduce the 1D decomposition problem to the well-known hypergraph partitioning problem. In the literature, there is a lack of 2D decomposition heuristic which directly minimizes the communication requirements for parallel SpMxV computations. Three novel hypergraph models are introduced for 2D decomposition of sparse matrices for minimizing the communication volume requirement. The first hypergraph model is proposed for fine-grain 2D decomposition of the sparse matrices for parallel SpMxV. The second hypergraph model for 2D decomposition is proposed to produce jagged-like decomposition of the sparse matrix. The checkerboard decomposition based parallel matrix-vector multiplication algorithms are widely encountered in the literature. However, only the load balancing problem is addressed in those works. Here, we propose a new hypergraph model which aims the minimization of communication volume while maintaining the load balance among the processors for checkerboard decomposition, as the third model for 2D decomposition. The proposed model reduces the decomposition problem to the multi-constraint hypergraph partitioning problem. The notion of multi-constraint partitioning has recently become popular in graph partitioning. We applied the multi-constraint partitioning to the hypergraph partitioning problem for solving checkerboard partitioning. Graph partitioning by vertex separator (GPVS) is widely used for nested dissection based low fill ordering of sparse matrices for direct solution of linear systems. In this work,

we also show that the GPVS problem can formulated as hypergraph partitioning. We exploit this finding to develop a novel hypergraph partitioning-based nested dissection ordering. The recently proposed successful multilevel framework is exploited to develop a multilevel hypergraph partitioning tool PaToH for the experimental verification of our proposed hypergraph models. Experimental results on a wide range of realistic sparse test matrices confirm the validity of the proposed hypergraph models. In terms of communication volume, the proposed hypergraph models produce 30% and 59% better decompositions than the graph model in 1D and 2D decompositions of sparse matricies for parallel SpMxV computations, respectively. The proposed hypergraph partitioning-based nested dissection produces 25% to 45% better orderings than the widely used multiple mimimum degree ordering in the ordering of various test matrices arising from different applications.

Keywords: Sparse matrices, parallel matrix-vector multiplication, parallel processing, matrix decomposition, computational graph model, graph partitioning, computational hypergraph model, hypergraph partitioning, fill reducing ordering, nested dissection.

ÖZET

SEYREK MATRİS BÖLÜMLEME VE YENİDEN-DÜZENLEME İÇİN HİPERÇİZGE MODELLERİ

Ümit V. Çatalyürek
Bilgisayar ve Enformatik Mühendisliği, Doktora
Tez Yöneticisi: Doç. Dr. Cevdet Aykanat
Kasım, 1999

Çizgeler, koşut seyrek-matris vektör çarpımında (SpMxV) seyrek matrislerin ayrıştırılması ve az doluluk faktorizasyonu için kullanılan seyrek matrislerin yeniden düzenlenmesini içeren çeşitli bilimsel uygulamalarda seyrek matrislerin gösterimi için yaygın olarak kullanılmaktadır. Ancak seyrek matrislerin standart çizge-bölümlemeye dayalı tek-boyutlu ayrıştırılması koşut SpMxV işlemi için gerekli iletişim hacmini yansıtamamaktadır. Çizge modelinin tek-boyutlu ayrıştırmadaki bu önemli eksikliğine karşılık benzer bir eksiği olmayan iki bilişimsel hiperçizge modeli sunuyoruz. Önerdiğimiz modeller tek-boyutlu ayrıştırma problemini iyi bilinen hiperçizge bölümleme problemine indirgemektedir. Literatürde koşut SpMxV hesaplamaları için iletişim gereksinimini doğrudan azaltan iki-boyutlu ayrıştırma yöntemi yoktur. İletişim hacmi gereksinimini azaltmak için seyrek matrislerin iki-boyutlu ayrıştırmasını sağlayan üç yeni hiperçizge modeli tanıtıyoruz. Bunlardan ilki koşut SpMxV işlemindeki seyrek matrislerin fine-grain iki-boyutlu ayrıştırması için önerildi. İki-boyutlu ayrıştırmada kullanılan ikinci hiperçizge modeli seyrek matrislerin çentikli-benzeri ayrıştırmalarının üretilmesi için önerildi. Literatürde dama tahtası tabanlı ayrıştırmaya dayanan koşut matris vektor çarpımı algoritmaları yaygınca bulunmaktadır. Bununla birlikte bu çalışmalarda sadece yük dengeleme problemine işaret edilmiştir. Biz bu çalışmada iki-boyutlu ayrıştırmanın üçüncü modeli olarak dama tahtası tabanlı ayrıştırmada işlemciler arası yük dengesini korurken iletişim hacmini de azaltmayı hedefleyen yeni bir hiperçizge modeli öneriyoruz. Önerdiğimiz model ayrıştırma problemini çoklu-kısıt hiperçizge bölümleme problemine indirgemektedir. Çoklu-kısıt bölümleme fikri çizge bölümleme alanında yakın zamanda popüler olmuştur. Biz de dama tahtası bölümleme problemini çözmek için bu çoklu-kısıt bölümleme fikrini hiperçizge parçalama yöntemine uyguladık. Düğüm ayırıcıları ile çizge bölümleme yöntemi

doğrusal sistemlerin çözümünde kullanılan, seyrek matrislerin içiçe ayırma tabanlı az doluluklu düzenlenmesinde çokca kullanılmaktadır. Bu çalışmada, düğüm ayırıcılar ile çizge bölümleme probleminin de hiperçizge bölümleme olarak formüle edilebileceğini gösterdik. Bu buluşumuzu hiperçizge bölümlemeye dayanan yeni bir içiçe ayırarak düzenleme yöntemi geliştirmek için kullandık. Önerdiğimiz hiperçizge modellerinin deneysel doğruluğunu sınamak için yakın zamanda önerilen başarılı çokludüzey çatıyı kullanarak bir çokludüzey hiperçizge bölümleme aracı olan PaToH'u geliştirdik. Gerceğe uygun, sınama amaçlı seyrek matrisler üzerindeki deneysel sonuçlar önerilen hiperçizge modellerinin geçerliliğini doğruladı. İletişim hacmi anlamında, önerdiğimiz hiperçizge modelleri koşut SpMxV hesaplamalarında çizge modeline göre yapılan tek-boyutlu ve iki-boyutlu ayrıştırmalara kıyasla anılan sıraya göre birinden yüzde 30 ve digerinden yüzde 59 daha iyi ayrıştırmalar üretmektedir. Önerilen hiperçizge tabanlı içiçe bölümlere ayırma yöntemi de farklı uygulamalarda ortaya çıkan çeşitli sınama amaçlı matrisleri düzenleme işleminde yaygın olarak kullanılan çoklu en düşük derece düzenlemesine kıyasla yüzde 25'ten yüzde 45'e kadar daha iyi olan düzenlemeler üretmektedir.

Anahtar sözcükler: Seyrek matrisler, koşut matris-vektör çarpımı, koşut işlem, matris ayrıştırma, bilişimsel çizge modeli, çizge bölümleme, bilişimsel hiperçizge modeli, hiperçizge bölümleme, doluluk azaltan sıralama, içiçe ayırma.

Acknowledgement

I would like to express my deepest gratitude to my supervisor Assoc. Prof. Cevdet Aykanat for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis. His creativity, lively discussions and cheerful laughter provided an invaluable and joyful atmosphere for our work.

I am grateful to Prof. Kemal Efe, Asst. Prof. Attila Gürsoy, Prof. İrşadi Aksun and Asst. Prof. Kıvanç Dinçer for reading and commenting on the thesis.

I owe special thanks to Prof. Mehmet Baray for providing a pleasant environment for study.

I am grateful to my family and my friends for their infinite moral support and help. I owe special thanks to my friends İlker Cengiz, Bora Uçar, Mehmet Koyutürk and Ferhat Büyükkökten for reading the thesis.

I would also like to thank George Karypis and Vipin Kumar, and Cleve Ashcraft and Joseph Liu for supplying their state-of-the-art codes, MeTiS and SMOOTH.

I would like to express very special thanks to Bruce Hendrickson and Cleve Ashcraft for their instructive comments.

Finally, I would like to thank my wife Gamze for her endless patience while I spent untold hours in front of my computer. Her support in so many ways deserves all I can give.

To my parents, my wife Gamze and my son Kaan

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs have been widely used to represent sparse matrices for various scientific applications including one-dimensional decomposition of sparse matrices for parallel sparse-matrix vector multiplication (SpMxV) and sparse matrix reordering for low fill factorization. In this work, we show the flaws of the graph models in these applications. We propose novel hypergraph models to avoid the flaws of the graph models.

In the subsequent sections of this chapter, the contributions are briefly summarized. Chapter 2 introduces the notation and background information for graph and hypergraph partitioning, and matrix reordering problems. The thesis work is mainly divided into four groups:

1. one-dimensional (1D) decomposition for parallel SpMxV,
2. two-dimensional (2D) decomposition for parallel SpMxV,
3. hypergraph partitioning-based sparse matrix ordering
4. development of a multilevel hypergraph partitioning tool for experimental verification of the proposed methods.

These works are described and discussed into detail in Chapters 3, 4, 5, and 6, respectively.

1.1 Sparse Matrix Decomposition for Parallel Matrix-Vector Multiplication

Iterative solvers are widely used for the solution of large, sparse, linear system of equations on multicomputers. Two basic types of operations are repeatedly performed at each iteration. These are linear operations on dense vectors and sparse-matrix vector product of the form $\mathbf{y}=\mathbf{A}\mathbf{x}$, where $\mathbf{A}$ is an $M \times M$ square matrix with the same sparsity structure as the coefficient matrix [10, 14, 17, 64], and $\mathbf{y}$ and $\mathbf{x}$ are dense vectors. In order to avoid the communication of vector components during the linear vector operations, a symmetric partitioning scheme is adopted. That is, all vectors used in the solver are divided conformally with each other. In particular, the $\mathbf{x}$ and $\mathbf{y}$ vectors are divided as $[\mathbf{x}_1, \ldots, \mathbf{x}_K]^t$ and $[\mathbf{y}_1, \ldots, \mathbf{y}_K]^t$, respectively. To compute the matrix vector product in parallel, matrix $\mathbf{A}$ is distributed among processors of the underlying parallel architecture. $\mathbf{A}$ can be written as $\mathbf{A}=\sum_k \mathbf{A}^k$, where the $\mathbf{A}^k$ matrix is owned by processor P_k, and the structure of the $\mathbf{A}^k$ matrices are mutually disjoint. The matrix-vector multiply is then computed as $\mathbf{y} = \sum_k \mathbf{y}^k$, where $\mathbf{y}^k = \mathbf{A}^k x$. Depending on the way in which $\mathbf{A}$ is partitioned among the processors, entries in $\mathbf{x}$ and/or entries in $\mathbf{y}^k$ may need to be communicated among the processors. Our goal here, is to find a decomposition that minimizes the total communication volume among the processors. Two kind of decompositions can be applied; 1D and 2D decomposition. In 1D decomposition, each processor is enforced to own either entire rows, (*rowwise* decomposition) or entire columns (*columnwise* decomposition). In parallel SpMxV, the rowwise and columnwise decomposition schemes require communication before or after the local SpMxV computations, thus they can also be considered as *pre* and *post* communication schemes, respectively. In rowwise decomposition, only the entries in $\mathbf{x}$ need to be communicated just before the local SpMxV computations. In columnwise decomposition, only the entries in $\mathbf{y}^k$ need to be communicated after local SpMxV computations. In 2D decomposition, processors are not imposed to own entire rows or columns. Therefore, both the entries in $\mathbf{x}$ and $\mathbf{y}^k$ need to be communicated among the processors. That is, both pre and post communication phases are needed in the 2D decomposition schemes.

In SpMxV computation, each nonzero element in a row/column incurs a multiply-add operation. Hence by assigning nonzero count to each row/column, load balancing problem in 1D decomposition can be considered as the *number partitioning* problem. Nastea et. al. [61] proposed a greedy heuristic to allocate rows of the matrix to the processors, namely GALA. GALA is simply first-fit-decreasing bin packing heuristic. They noticed that if the matrix has very dense rows, the resulting load balance is not good. To elevate this problem, they split the rows that have significantly large number of nonzero elements into several parts prior to allocation process. Thus finer granularity of the allocation problem leads to better load balancing results. However, the decomposition heuristics [61, 70] proposed for computational load balancing may result in an extensive communication volume, since they do not consider the minimization of the communication volume during the decomposition.

Heuristics proposed for load balancing problem [62, 56, 55] in 2D decomposition assumes that the underlying parallel algorithm for matrix-vector multiplication is based on 2D checkerboard partitioning running on a 2D mesh architecture. In checkerboard partitioning, assignment of matrix elements to processors preserves the *integrity* of the matrix by placing every row (column) of the matrix into the processors lying in a single row (column) of the 2D mesh. Ogielski and Aiello [62] proposed two heuristics which are based on the random permutation of rows and columns. Hendrickson et.al. [37] noticed that most matrices used in real applications have nonzero diagonal elements, and they state that it may be advantageous to force an even distribution of these diagonal elements among processors and to randomly distribute the remaining nonzeros. Lewis and Geijn [56] and Lewis et.al. [55] proposed a new scattered distribution of the matrix which totally avoids the transpose operation required in [37].

In a K-processor parallel architecture, load balancing heuristics for both 1D and 2D decomposition schemes may introduce an extensive amount of communication since they do not consider the minimization of communication requirement explicitly. For an $M \times M$ sparse matrix $\mathbf{A}$, the worst-case communication requirement in 1D decomposition is $K(K-1)$ messages and $(K-1)M$ words, and it occurs when each submatrix $\mathbf{A}^k$ has at least one nonzero in each column (row) in rowwise (columnwise) decomposition. The matrix-vector multiplication

algorithms based on 2D checkerboard partitioning [37, 56, 55] reduce the worst-case communication to $2K(\sqrt{K}-1)$ messages and $2(\sqrt{K}-1)M$ words. In this approach, the worst-case occurs when each row and column of each submatrix has at least one nonzero.

The computational graph model is widely used in the representation of computational structures of various scientific applications, including repeated SpMxV computations, to decompose the computational domains for parallelization [14, 15, 41, 46, 50, 51, 60, 68]. In this model, the problem of 1D sparse matrix decomposition for minimizing the communication volume while maintaining the load balance is formulated as the well-known K-way *graph partitioning by edge separator* (GPES) problem. In this work, we show the deficiencies of the graph model for decomposing sparse matrices for parallel SpMxV. The first deficiency is that it can only be used for structurally symmetric square matrices. In order to avoid this deficiency, we propose a generalized graph model in Section 3.1.3 which enables the decomposition of structurally nonsymmetric square matrices as well as symmetric matrices. The second deficiency is the fact that none of the graph models reflects the actual communication requirement as will be described in Section 3.2. These flaws are also mentioned in a concurrent work [33].

In this work, we propose two *computational hypergraph* models which avoid all deficiencies of the graph model for 1D decomposition. The proposed models enable the representation and hence the 1D decomposition of rectangular matrices [63] as well as symmetric and nonsymmetric square matrices. Furthermore, they introduce an exact representation for the communication volume requirement as described in Section 3.3. The proposed hypergraph models reduce the decomposition problem to the well-known *K-way hypergraph partitioning* problem widely encountered in circuit partitioning in VLSI layout design. Hence, the proposed models will be amenable to the advances in the circuit partitioning heuristics in the VLSI community. The detailed discussion and presentation of the proposed hypergraph models can be found in Chapter 3.

There is no work in the literature which directly aims at the minimization of communication requirements in 2D decomposition for parallel SpMxV computations. We propose three novel hypergraph models for 2D decomposition of sparse matrices. A fine-grain hypergraph model is proposed in Section 4.1. In

this fine-grain model, the nonzeros of the matrix are considered as the atomic tasks in the decomposition. Two coarse-grain hypergraph models are proposed in Sections 4.2 and 4.3. The coarse-grain models have two objectives. The first objective is to reduce the decomposition overhead. The second objective is an implicit effort towards reducing the amount of communication which is a valuable asset in parallel architectures with high start-up cost. The first coarse-grain hypergraph model, produces jagged-like 2D decompositions of the sparse matrices. The second hypergraph model is specifically designed for checkerboard partitioning which is commonly used in the literature by the matrix-vector multiplication algorithms [62, 56, 55, 37]. Details of these models are presented and discussed in Chapter 4.

1.2 Sparse Matrix Ordering for Low Fill Factorization

For a symmetric matrix, the evolution of the nonzero structure during the Cholesky factorization can easily be described in terms of its graph representation. In graph terms, the elimination of a vertex creates edges for every pair of its adjacent vertices. In other words, elimination of a vertex makes its adjacent vertices clique of size its degree minus one. In this process, the added edges directly correspond to the *fill* in the matrix. The number of floating-point operations, also known as *operation count*, required to perform the factorization is equal to the sum of the squares of the nonzeros of each eliminated row/column. Hence it is also equal to the sum of the squares of the degrees of corresponding vertices during the elimination. Obviously, the amount of fill and operation count depends on the row/column elimination order. The aim of ordering is to reduce these quantities, which yields both faster and less memory intensive factorization.

One of the most popular ordering methods is *Minimum Degree* (MD) heuristic [74]. Success of the MD heuristic is followed by many variants of it, such as Quotient Minimum Degree (QMD) [28], Multiple Minimum Degree (MMD) [57], Approximate Minimum Degree (AMD) [3], and Approximate Minimum Fill (AMF) [69]. An alternative method *nested dissection* (ND) was proposed by

George [27]. The intuition behind this method is as follows. First a set of columns **S** (*separator*), whose removal decouples the matrix into two parts, say **X** and **Y**, is found. If we order **S** after **X** and **Y**, then no fill can occur in the off-diagonal blocks. Elimination process in **X** and **Y** are independent tasks and they do not incur any fill to each other. Hence, ordering of **X** and **Y** can be computed by applying the algorithm recursively, or using any other technique. It is clear that, the quality of the ordering depends on the size of **S**. In ND, separator finding problem is usually formulated as *graph partitioning by vertex separator* (GPVS) problem on the standard graph representation of the matrix.

In a recent work [11], we have shown that the hypergraph partitioning problem can be formulated as a GPVS problem on its *net intersection graph* (NIG). In matrix terms, this work shows that permuting a sparse matrix **A** into singly-bordered block-diagonal form can also be formulated as permuting $\mathbf{AA^T}$ into a doubly-bordered block-diagonal (DB). Note that, nested dissection also requires a DB form, in particular, borders in DB form correspond to separator **S** and block-diagonals correspond to the **X** and **Y** parts. In this work, we exploit this equivalence in the reverse direction. However, for a given hypergraph, although its NIG representation is well-defined, there is no unique reverse construction. In matrix terms, for a symmetric matrix **Z** there is no unique construction of $\mathbf{Z} = \mathbf{AA}^T$ decomposition. Luckily, in linear programming (LP) applications, interior point type solvers require the solution of $\mathbf{Z}x = b$ repeatedly, where $\mathbf{Z} = \mathbf{ADA}^T$. Here, **D** is a diagonal matrix whose numerical values are changed in each iteration. However, since it is diagonal, it doesn't effect the sparsity pattern of the **Z** matrix. In graph terms, if we represent **A** by its row-net hypergraph model, its NIG is the graph representation of **Z**. Therefore we can use the hypergraph representation of **A** for a hypergraph partitioning-based nested dissection ordering of **Z**. For generalization, if **A** is unknown, we also propose a 2-clique decomposition **C** of any symmetric matrix **Z** into $\mathbf{Z} = \mathbf{CC}^T$. Details of this decomposition and hypergraph partitioning-based ordering is presented in Chapter 5.

1.3 Multilevel Hypergraph Partitioning

Decomposition and reordering are preprocessings introduced for the sake of efficient parallelization and low fill factorization, respectively. Hence, heuristics should run in low order polynomial time. Recently, multilevel graph partitioning heuristics [13, 35, 46] have been proposed leading to fast and successful graph partitioning tools Chaco [36], MeTiS [44], WGPP [31] and reordering tools BEND [38], oMeTiS [44], and ordering code of WGPP [30]. We have exploited the multilevel partitioning methods for the experimental verification of the proposed hypergraph models in both sparse matrix decomposition problems and sparse matrix ordering. The lack of a multilevel hypergraph partitioning tool at the time of this work was carried, led us to develop a multilevel hypergraph partitioning tool PaToH. The main objective in the implementation of PaToH was a fair experimental comparison of the hypergraph models with the graph models both in sparse matrix decomposition and in sparse matrix ordering. Another objective in our PaToH implementation was to investigate the performance of multilevel approach in hypergraph partitioning as described in Chapter 6.

Chapter 2

Preliminaries

In this chapter we will review definition of graph, hypergraph and partitioning problems in Section 2.1 and 2.2, respectively. Attempts to solve hypergraph partitioning problem as graph partitioning problem are presented in Section 2.3. Various partitioning heuristics and tools are summarized in Section 2.4. Sparse matrix ordering heuristics and tools are presented in Section 2.5. We will review how the graph partitioning by vertex separator problem is solved using graph partitioning by edge separator methods in Section 2.6, and finally, we will discuss the overlooked non-optimality of the this solution in Section 2.7.

2.1 Graph Partitioning

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. 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 in graph $\mathcal{G}$. We extend this operator to include the adjacency set of a vertex subset $\mathcal{V}' \subset \mathcal{V}$, i.e., $Adj(\mathcal{V}') = \{v_j \in \mathcal{V} - \mathcal{V}' : v_j \in Adj(v_i) \text{ for some } v_i \in \mathcal{V}'\}$. The degree d_i of a vertex v_i is equal to the number of edges incident to v_i, i.e., $d_i = |Adj(v_i)|$. Weights and costs can be assigned to the vertices and edges of the graph, respectively. Let w_i and c_{ij} denote the weight of vertex $v_i \in \mathcal{V}$ and the cost of edge $e_{ij} \in \mathcal{E}$, respectively. Two partitioning problems can be defined on the graph, these are

graph partitioning by edge separator and graph partitioning by node separator. In the following subsections we will briefly describe these problems.

2.1.1 Graph Partitioning by Edge Separator (GPES)

An edge subset $\mathcal{E}_S \subseteq \mathcal{E}$ is a K-way *edge separator* if its removal disconnects the graph into at least K connected components. $\Pi_{GPES} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a *K-way partition* of $\mathcal{G}$ by edge separator $\mathcal{E}_S$ if the following conditions hold:

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

Note that all edges between the vertices of different parts belong to $\mathcal{E}_S$. Edges in $\mathcal{E}_S$ are called *cut (external)* edges and all other edges are called *uncut (internal)* edges. In a partition Π_{GPES} of $\mathcal{G}$, a vertex is said to be a *boundary* vertex if it is incident to a cut edge. A K-way partition is also called a *multiway* partition if $K > 2$ and a *bipartition* if $K = 2$. A partition is said to be balanced if each part $\mathcal{V}_k$ satisfies the *balance criterion*

$$W_k \leq W_{avg}(1 + \varepsilon), \quad for\ k = 1, 2, \ldots, K. \tag{2.1}$$

In (2.1), weight W_k of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in that part (i.e. $W_k = \sum_{v_i \in \mathcal{V}_k} w_i$), $W_{avg} = (\sum_{v_i \in \mathcal{V}} w_i)/K$ denotes the weight of each part under the perfect load balance condition, and ε represents the predetermined maximum imbalance ratio allowed. The *cutsize* definition for representing the cost $\chi(\Pi_{GPES})$ of a partition Π_{GPES} is

$$\chi(\Pi_{GPES}) = \sum_{e_{ij} \in \mathcal{E}_S} c_{ij}. \tag{2.2}$$

In (2.2), each cut edge e_{ij} contributes its cost c_{ij} to the cutsize. Hence, the GPES problem can be defined as the task of dividing a graph into two or more parts such that the cutsize is minimized, while the balance criterion (2.1) on part weights is maintained. The GPES problem is known to be NP-hard even for bipartitioning unweighted graphs [26].

2.1.2 Graph Partitioning by Vertex Separator (GPVS)

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. $\Pi_{GPVS} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K; \mathcal{V}_S\}$ is a K-way vertex partition of $\mathcal{G}$ by vertex separator $\mathcal{V}_S \subset \mathcal{V}$ if the following conditions hold:

- each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, i.e., $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$,
- parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_\ell = \emptyset$ for all $1 \leq k < \ell \leq K$
- parts and separator are disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_S = \emptyset$ for $1 \leq k < K$
- union of K parts and separator is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k \cup \mathcal{V}_S = \mathcal{V}$,
- the removal of $\mathcal{V}_S$ gives K disconnected parts $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K$, i.e., $Adj(\mathcal{V}_k) \subseteq \mathcal{V}_S$ for $1 \leq k \leq K$.

In a partition Π_{GPVS} of $\mathcal{G}$, 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 a 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 cost of a partition Π_{GPVS} is

$$cost(\Pi_{GPVS}) = \sum_{v_i \in \mathcal{V}_S} w_i. \tag{2.3}$$

In (2.3) each separator vertex contributes its weight to cost. Hence, the K-way GPVS problem can be defined as the task of finding a K-way vertex separator of minimum cost, while the balance criterion (2.1) on part weights is maintained. GPVS problem is also known to be NP-hard [12].

2.2 Hypergraph Partitioning (HP)

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices. Every net $n_j \in \mathcal{N}$ is a subset of vertices, i.e., $n_j \subseteq \mathcal{V}$. The vertices in a net n_j are called its *pins* and denoted as $pins[n_j]$. The size of a net is equal to the number of its pins, i.e., $s_j = |pins[n_j]|$. The set of nets connected to a vertex v_i is denoted as $nets[v_i]$. The degree of a vertex is equal to the number of nets it is connected to, i.e., $d_i = |nets[v_i]|$. Graph is a special instance of hypergraph such that each net has exactly two pins. Similar to graphs, let w_i and c_j denote the weight of vertex $v_i \in \mathcal{V}$ and the cost of net $n_j \in \mathcal{N}$, respectively.

Definition of K-way partition of hypergraphs is identical to that of GPES. In a partition Π of $\mathcal{H}$, a net that has at least one pin (vertex) in a part is said to *connect* that part. *Connectivity set* Λ_j of a net n_j is defined as the set of parts connected by n_j. *Connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* if it connects more than one part (i.e. $\lambda_j > 1$), and *uncut* otherwise (i.e. $\lambda_j = 1$). The cut and uncut nets are also referred to here as *external* and *internal* nets, respectively. The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. There are various [75, 19] *cutsize* definitions for representing the cost $\chi(\Pi)$ of a partition Π. Two relevant definitions are:

$$(a) \quad \chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} c_j \qquad and \qquad (b) \quad \chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} c_j(\lambda_j - 1). \tag{2.4}$$

In (2.4.a), the cutsize is equal to the sum of the costs of the cut nets. In (2.4.b), each cut net n_j contributes $c_j(\lambda_j - 1)$ to the cutsize. Hence, the hypergraph partitioning problem can be defined as the task of dividing a hypergraph into two or more parts such that the cutsize is minimized, while a given balance criterion (2.1) among the part weights is maintained. Here, part weight definition is identical to that of the graph model. The hypergraph partitioning problem is known to be NP-hard [54].

2.3 Graph Representation of Hypergraphs

As indicated in the excellent survey by Alpert and Kahng [2], hypergraphs are commonly used to represent circuit netlist connections in solving the circuit partitioning and placement problems in VLSI layout design. The circuit partitioning problem is to divide a system specification into clusters such that the number of inter-cluster connections is minimized. Other circuit representation models were also proposed and used in the VLSI literature including dual hypergraph, clique-net graph and net-intersection graph (NIG) [2]. Hypergraphs represent circuits in a natural way so that the circuit partitioning problem is directly described as an HP problem. Hence, these alternative circuit representation models can also be considered as alternative models for the HP problem so that the cutsize in a partitioning of these models relate to the cutsize of a partitioning of the hypergraph.

The dual of a given hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ is defined as a hypergraph $\mathcal{H}'$, where the nodes and nets of $\mathcal{H}$ become, respectively, the nets and nodes of $\mathcal{H}'$. That is, $\mathcal{H}' = (\mathcal{U}', \mathcal{N}')$ with $nets[u'_i] = pins[n_i]$ for each $u'_i \in \mathcal{U}'$ and $n_i \in \mathcal{N}$, and $pins[n'_j] = nets[u_j]$ for each $n'_j \in \mathcal{N}'$ and $u_j \in \mathcal{U}$.

In the clique-net transformation model, the vertex set of the target graph is equal to the vertex set of the given hypergraph with the same vertex weights. Each net of the given hypergraph is represented by a clique of vertices corresponding to its pins. That is, each net induces an edge between every pair of its pins. The multiple edges connecting each pair of vertices of the graph are contracted into a single edge of which cost is equal to the sum of the costs of the edges it represents. In the *standard* clique-net model [54], a uniform cost of $1/(s_i - 1)$ is assigned to every clique edge of net n_i with size s_i. Various other edge weighting functions are also proposed in the literature [2]. If an edge is in the cut set of a GPES then all nets represented by this edge are in the cut set of hypergraph partitioning, and vice versa. Ideally, no matter how vertices of a net are partitioned, the contribution of a cut net to the cutsize should always be one in a bipartition. However, the deficiency of the clique-net model is that it is impossible to achieve such a *perfect* clique-net model [40]. Furthermore, the transformation may result in very large graphs since the number of clique edges induced by the nets increase

quadratically with their sizes.

Recently, a randomized clique-net model implementation is proposed [1] which yields very promising results when used together with graph partitioning tool MeTiS. In this model, all nets of size larger than T are removed during the transformation. Furthermore, for each net n_i of size s_i, $F \times s_i$ random pairs of its pins (vertices) are selected and an edge with cost one is added to the graph for each selected pair of vertices. The multiple edges between each pair of vertices of the resulting graph are contracted into a single edge as mentioned earlier. In this scheme, the nets with size smaller than $2F+1$ (small nets) induce larger number of edges than the standard clique-net model, whereas the nets with size larger than $2F+1$ (large nets) induce smaller number of edges than the standard clique-net model. Considering the fact that MeTiS accepts integer edge costs for the input graph, this scheme has two nice features[1]. First, it simulates the uniform edge-weighting scheme of the standard clique-net model for small nets in a random manner since each clique edge (if induced) of a net n_i with size $s_i < 2F+1$ will be assigned an integer cost close to $2F/(s_i-1)$ on the average. Second, it prevents the quadratic increase in the number of clique edges induced by large nets in the standard model since the number of clique edges induced by a net in this scheme is linear in the size of the net. In our implementation, we use the parameters $T = 50$ and $F = 5$ in accordance with the recommendations given in [1].

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 $\mathcal{H}$. 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,

$$Adj(v_i) = \{v_j \ : \ n_j \in \mathcal{N} \ \ni \ pins[n_i] \cap pins[n_j] \neq \emptyset\}. \tag{2.5}$$

The NIG representation $\mathcal{G}$ for a hypergraph $\mathcal{H}$ can also be obtained by applying the clique-net model to the dual hypergraph of $\mathcal{H}$. Note that for a given hypergraph $\mathcal{H}$, NIG $\mathcal{G}$ is well-defined, however there is no unique reverse construction [2].

[1] private communication with Alpert.

Both dual hypergraph and NIG models view the HP problem in terms of partitioning nets instead of nodes. Kahng [42] and Cong, Hagen, and Kahng [20] exploited this perspective of the NIG model to formulate the hypergraph bipartitioning problem as a two-stage process. In the first stage, nets of $\mathcal{H}$ are bipartitioned through 2-way GPES of its NIG $\mathcal{G}$. The resulting net bipartition induces a partial node bipartition on $\mathcal{H}$, since the nodes (pins) that belong only to the nets on one side of the bipartition can be unambigiuosly assigned to that side. However, other nodes may belong to the nets on both sides of the bipartition. Thus, the second stage involves finding the best completion of the partial node bipartition; i.e., a part assignment for the shared nodes such that the cutsize (2.4.a) is minimized. This problem is known as the module (node) contention problem in the VLSI community. Kahng [42] used a winner-loser heuristic [32], whereas Cong et al. [20] used a matching-based (IG-match) algorithm for solving the 2-way module contention problem optimally. Cong, Labio, and Shivakumar [21] extended this approach to K-way hypergraph partitioning through using the dual hypergraph model. In the first stage, a K-way net partitioning is obtained through partitioning the dual hypergraph. For the second stage, they formulated the K-way module contention problem as a min-cost max-flow problem through defining binding factors between nodes and nets, and preference function between parts and nodes.

2.4 Graph/Hypergraph Partitioning Heuristics and Tools

Kernighan-Lin (KL) based heuristics are widely used for graph/hypergraph partitioning because of their short run-times and good quality results. The KL algorithm is an iterative improvement heuristic originally proposed for graph bipartitioning [48]. The KL algorithm, starting from an initial bipartition, performs a number of passes until it finds a locally minimum partition. Each pass consists of a sequence of vertex swaps. The same swap strategy was applied to the hypergraph bipartitioning problem by Schweikert-Kernighan [72]. Fiduccia-Mattheyses (FM) [25] introduced a faster implementation of the KL algorithm

for hypergraph partitioning. They proposed vertex move concept instead of vertex swap. This modification, as well as proper data structures, e.g., bucket lists, reduced the time complexity of a single pass of the KL algorithm to linear in the size of the graph and the hypergraph. Here, *size* refers to the number of edges and pins in a graph and hypergraph, respectively.

The performance of the FM algorithm deteriorates for large and very sparse graphs/hypergraphs. Here, sparsity of graphs and hypergraphs refer to their average vertex degrees. Furthermore, the solution quality of FM is not *stable* (*predictable*), i.e., average FM solution is significantly worse than the best FM solution, which is a common weakness of the move-based iterative improvement approaches. Random multi-start approach is used in VLSI layout design to alleviate this problem by running the FM algorithm many times starting from random initial partitions to return the best solution found [2]. However, this approach is not viable in parallel computing since decomposition is a preprocessing overhead introduced to increase the efficiency of the underlying parallel algorithm/program. Most users will rely on one run of the decomposition heuristic, so the quality of the decomposition tool depends equally on the worst and average decompositions than on just the best decomposition.

These considerations have motivated the *two–phase* application of the move-based algorithms in hypergraph partitioning [29]. In this approach, a clustering is performed on the original hypergraph $\mathcal{H}_0$ to induce a coarser hypergraph $\mathcal{H}_1$. Clustering corresponds to coalescing highly interacting vertices to supernodes as a preprocessing to FM. Then, FM is run on $\mathcal{H}_1$ to find a bipartition Π_1, and this bipartition is projected back to a bipartition Π_0 of $\mathcal{H}_0$. Finally, FM is re-run on $\mathcal{H}_0$ using Π_0 as an initial solution. Cong-Smith [22] introduced a clustering algorithm which works on the graphs. They convert the hypergraph to a graph by representing an r-pin net as a $r-clique$. Then they use a heuristic algorithm to construct the clusters. The clustered graph is given as input to the Fiduccia-Mattheyses algorithm. Shin-Kin [73] proposed a clustering algorithm which works on hypergraphs, then a KL based heuristic is used to partition the clustered hypergraph.

Recently, the two–phase approach has been extended to *multilevel* approaches [13, 35, 46] leading to successful graph partitioning tools Chaco [36]

and MeTiS [44]. These multilevel heuristics consist of 3 phases: *coarsening*, *initial partitioning* and *uncoarsening*. In the first phase, a multilevel clustering is applied starting from the original graph by adopting various matching heuristics until the number of vertices in the coarsened graph reduces below a predetermined threshold value. In the second phase, the coarsest graph is partitioned using various heuristics including FM. In the third phase, the partition found in the second phase is successively projected back towards the original graph by refining the projected partitions on the intermediate level uncoarser graphs using various heuristics including FM.

The success of multilevel algorithms both in runtime and solution quality makes them as a standard for the partitioning problem. The lack of a multilevel hypergraph partitioning tool at the time of this work was carried led us to develop a multilevel hypergraph partitioning tool PaToH for a fair experimental comparison of the hypergraph models with the graph models. The details of PaToH will be described in Chapter 6. Since multilevel graph partitioning tool MeTiS is accepted as the state-of-the-art partitioning tool, we have also used it for hypergraph partitioning problem with a hybrid approach using randomized clique-net.

2.5 Sparse Matrix Ordering Heuristics and Tools

As we mentioned earlier, the most popular ordering method is *Minimum Degree* (MD) heuristic [74]. The motivation of this method is simple. Since elimination of a vertex causes its adjacent vertices to become adjacent, MD selects a vertex of minimum degree to eliminate next. Success of the MD heuristic is followed by many variants of it. Very first implementations, such as Quotient Minimum Degree (QMD) [28] was too slow, although it is an in-place algorithm (that is no extra storage is required for fill-edges). A faster variant is Multiple Minimum Degree (MMD) [57]. It reduces the runtime of the heuristic by eliminating a set of vertex of minimum degree. By computing upper bound on a vertex's degree rather than the true degree, runtime of the heuristic even further reduced by

the recent variant Approximate Minimum Degree (AMD) [3]. Another recently proposed variant is Approximate Minimum Fill (AMF) [69]. This method uses the selection criteria that roughly approximate the amount of fill that would be generated by the elimination of a vertex instead of using the vertex degree.

As stated before, *Nested Dissection* (ND) is an alternative to MD algorithm. However, although good theoretical results are presented in [27], nested dissection has not been used until recently. Evolution of the graph partitioning tools have changed the situation and better methods for finding graph separators are available now, including Kernighan-Lin and Fiduccia-Mattheyses and their multilevel variants [48, 25, 12, 43, 35], vertex-separator Fiduccia-Mattheyses variants [6, 39] and spectral methods [66, 67].

The multilevel GPES approaches have been used in several multilevel nested dissection implementations based on indirect 2-way GPVS, e.g., *oemetis* ordering code of MeTiS [44]. Converting the solution of GPES to GPVS will be briefly described in the next section. Recently, direct 2-way GPVS approaches have been embedded into various multilevel nested dissection implementations [31, 38, 44]. In these implementations, a 2-way GPVS obtained on the coarsest graph is refined during the multilevel framework of the uncoarsening phase. Two distinct vertex-separator refinement schemes were proposed and used for the uncoarsening phase. The first one is the extension of the FM edge-separator refinement approach to vertex-separator refinement as proposed by Ashcraft and Liu [5]. This scheme considers vertex moves from vertex separator $\mathcal{V}_S$ to both $\mathcal{V}_1$ and $\mathcal{V}_2$ in $\Pi_{GPVS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$. This refinement scheme is adopted in the *onmetis* ordering code of MeTiS [44], ordering code of WGPP [31], and the ordering code BEND [38]. The second scheme is based on Liu's narrow separator refinement algorithm [58], which considers moving a set of vertices simultaneously from $\mathcal{V}_S$ at a time, in contrast to the FM-based refinement scheme [5], which moves only one vertex at a time. Liu's refinement algorithm [58] can be considered as repeatedly running the maximum-matching based vertex cover algorithm on the bipartite graphs induced by the edges between $\mathcal{V}_1$ and $\mathcal{V}_S$, and $\mathcal{V}_2$ and $\mathcal{V}_S$. That is, the wide vertex separator consisting of $\mathcal{V}_S$ and the boundary vertices of $\mathcal{V}_1$ ($\mathcal{V}_2$) is refined as in the GPES-based wide-to-narrow separator refinement scheme. The network-flow based minimum weighted vertex cover algorithms proposed by Ashcraft and

Liu [8], and Hendrickson and Rothberg [38] enabled the use of Liu's refinement approach [58] on the coarse graphs within the multilevel framework.

2.6 Solving GPVS Through GPES

Until recently, instead of solving the GPVS problem directly, it is solved through GPES. These indirect GPVS approaches first perform a GPES on the given graph to minimize the number of cut edges (i.e., $c_{ij} = 1$ in (2.2)) and then take the boundary vertices as the wide separator to be refined to a narrow separator. The wide-to-narrow refinement problem is described as a *minimum vertex cover* problem on the subgraph induced by the cut edges [66]. A minimum vertex cover is taken as a narrow separator for the whole graph, since each cut edge will be adjacent to a vertex in the vertex cover. That is, let $\mathcal{V}_{Bk} \subseteq \mathcal{V}_k$ denote the set of boundary vertices of part $\mathcal{V}_k$ in a partition $\Pi_{GPES} = \{\mathcal{V}_1, \ldots, \mathcal{V}_K\}$ of a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by edge separator $\mathcal{E}_S \subset \mathcal{E}$. Then, $\mathcal{K}(\mathcal{E}_S) = (\mathcal{V}_B = \cup_{k=1}^{K} \mathcal{V}_{Bk}, \mathcal{E}_S)$ denotes the K-partite subgraph of $\mathcal{G}$ induced by $\mathcal{E}_S$. A vertex cover $\mathcal{V}_S = \cup_{k=1}^{K} \mathcal{V}_{Sk}$ on $\mathcal{K}(\mathcal{E}_S)$ constitutes a K-way GPVS $\Pi_{GPVS} = \{\mathcal{V}_1 - \mathcal{V}_{S1}, \ldots, \mathcal{V}_K - \mathcal{V}_{SK}; \mathcal{V}_S\}$ of $\mathcal{G}$, where $\mathcal{V}_{Sk} \subseteq \mathcal{V}_{Bk}$ denotes the subset of boundary vertices of part $\mathcal{V}_K$ that belong to the vertex cover of $\mathcal{K}(\mathcal{E}_S)$. A minimum vertex cover $\mathcal{V}_S$ of $\mathcal{K}(\mathcal{E}_S)$ corresponds to an optimal refinement of the wide separator $\mathcal{V}_B$ into a narrow separator $\mathcal{V}_S$ under the assumption that each boundary vertex is adjacent to at least one non-boundary vertex in Π_{GPES} (see Section 2.7).

A minimum vertex cover of a bipartite graph can be computed optimally in polynomial time by finding a maximum cardinality matching, since these are dual concepts [52, 65, 66]. So, the wide-to-narrow separator refinement problem can easily be solved using this scheme for 2-way indirect GPVS, because the edge separator of a 2-way GPES induces a bipartite subgraph. This scheme has been widely exploited in a recursive manner in the nested-dissection based K-way indirect GPVS for ordering symmetric sparse matrices, because a 2-way GPES is adopted at each dissection step. However, the minimum vertex cover problem is known to be NP-hard on K-partite graphs at least for $K \geq 5$ [26], thus we need to resort to heuristics. Leiserson and Lewis [53] proposed two greedy heuristics for this purpose, namely minimum recovery (MR) and maximum inclusion (MI). The

MR heuristic is based on iteratively removing the vertex with minimum degree from the K-partite graph $\mathcal{K}(\mathcal{E}_S)$ and including all vertices adjacent to that vertex to the vertex cover $\mathcal{V}_S$. The MI heuristic is based on iteratively including the vertex with maximum degree into $\mathcal{V}_S$. In both heuristics, all edges incident to the vertices included into $\mathcal{V}_S$ are deleted from $\mathcal{K}(\mathcal{E}_S)$ so that the degrees of the remaining vertices in $\mathcal{K}(\mathcal{E}_S)$ are updated accordingly. Both heuristics continue the iterations until all edges are deleted from $\mathcal{K}(\mathcal{E}_S)$.

Here, we reveal the fact that the module contention problem encountered in the second stage of the NIG-based hypergraph bipartitioning approaches [20, 42] is similar to the wide-to-narrow separator refinement problem encountered in the second stage of the indirect GPVS approaches widely used in nested dissection based ordering. The module contention and separator refinement algorithms effectively work on the bipartite graph induced by the cut edges of a two-way GPES of the NIG representation of hypergraphs and the standard graph representation of sparse matrices, respectively. The winner-loser assignment heuristic [32, 42] used by Kahng [42] is very similar to the minimum-recovery heuristic proposed by Leiserson and Lewis [53] for separator refinement. Similarly, the IG-match algorithm proposed by Cong et al. [20] is similar to the maximum-matching based minimum vertex-cover algorithm [52, 65] used by Pothen, Simon, and Liou [66] for separator refinement. Despite not being stated in the literature, these net-bipartitioning based HP algorithms using the NIG model can be viewed as trying to solve the HP problem through an indirect GPVS of the NIG representation.

2.7 Vertex-Cover Model: On the Optimality of Separator Refinement

For 2-way GPES based GPVS, it was stated [65] that the minimum vertex cover $\mathcal{V}_S$ of the bipartite graph $\mathcal{K}(\mathcal{E}_S) = (\mathcal{V}_B = \mathcal{V}_{B1} \cup \mathcal{V}_{B2}, \mathcal{E}_S)$ induced by an edge separator $\mathcal{E}_S$ of GPES of $\Pi_{GPES} = \{\mathcal{V}_1, \mathcal{V}_2\}$ of $\mathcal{G}$ is a smallest vertex separator of $\mathcal{G}$ corresponding to $\mathcal{E}_S$. Recall that $\mathcal{V}_{Bk}$ denotes the set of boundary vertices of part $\mathcal{V}_k$. Here, we would like to discuss that this correspondence does not guarantee the optimality of the wide-to-narrow separator refinement. That is,

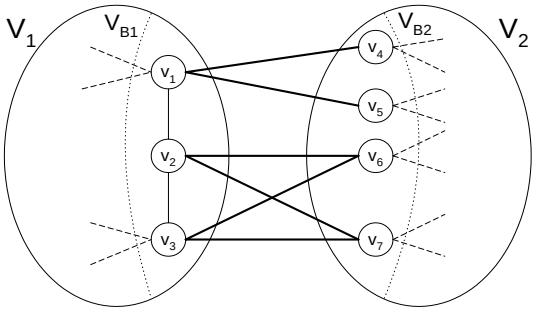

Figure 2.1: A sample 2-way GPES for wide-to-narrow separator refinement.

the minimum vertex cover $\mathcal{V}_S$ of $\mathcal{K}(\mathcal{E}_S)$ may not constitute a minimum vertex separator that can be obtained from the wide separator $\mathcal{V}_B$. We can classify the boundary vertices $\mathcal{V}_{Bk}$ of a part $\mathcal{V}_k$ as *loosely-bound* and *tightly-bound* vertices. A loosely-bound vertex v_i of $\mathcal{V}_{Bk}$ is not adjacent to any non-boundary vertex of $\mathcal{V}_k$, i.e., $Adj(v_i, \mathcal{V}_k) = Adj(v_i) \cap \mathcal{V}_k \subseteq \mathcal{V}_{Bk} - \{v_i\}$, whereas a tightly-bound vertex v_j of V_{Bk} is adjacent to at least one non-boundary vertex of $\mathcal{V}_k$, i.e., $Adj(v_j, \mathcal{V}_k - \mathcal{V}_{Bk}) \neq \emptyset$. Each cut edge between two tightly-bound vertices should always be covered by a vertex cover $\mathcal{V}_S$ of $\mathcal{K}(\mathcal{E}_S)$ for $\mathcal{V}_S$ to constitute a separator of $\mathcal{G}$. However, it is an unnecessarily severe measure to impose the same requirement for a cut edge incident to at least one loosely-bound vertex. If all vertices in $\mathcal{V}_{Bk}$ that are adjacent to a loosely-bound vertex $v_i \in \mathcal{V}_{Bk}$ are included into $\mathcal{V}_S$ then cut edges incident to v_i need not to be covered. For example, Fig. 2.1 illustrates a 2-way GPES, where $v_2 \in \mathcal{V}_{B1}$ is a loosely-bound vertex and all other vertices are tightly-bound vertices. Fig. 2.2 illustrates two optimal vertex covers $\mathcal{V}_S = \{v_1, v_2, v_3\}$ and $\mathcal{V}_S = \{v_1, v_6, v_7\}$, each of size 3, on bipartite graph $\mathcal{K}(\mathcal{E}_S)$. Vertices v_6 and v_2 are included into $\mathcal{V}_S$ in the former and latter solutions, respectively, to cover cut edge $(v_2, v6)$. However, in both solutions, $Adj(v_2, \mathcal{V}_1) = \{v_1, v_3\}$ remains in the optimal vertex cover so that there is no need to cover cut edge (v_2, v_6). Hence, there exists a wide-to-narrow separator refinement $\mathcal{V}_S = \{v_1, v_3\}$ of size 2 as shown in Fig. 2.3.

As mentioned in Section 2.5, Liu's narrow separator refinement algorithm [58] can also be considered as exploiting the vertex cover model on the bipartite graph induced by the edges between $\mathcal{V}_1$ and $\mathcal{V}_S$ ($\mathcal{V}_2$ and $\mathcal{V}_S$) of a GPVS $\Pi_{GPVS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$. So, the discussion given here also applies to Liu's narrow separator refinement algorithm, where loosely-bound vertices can only exist in the $\mathcal{V}_1$ ($\mathcal{V}_2$)

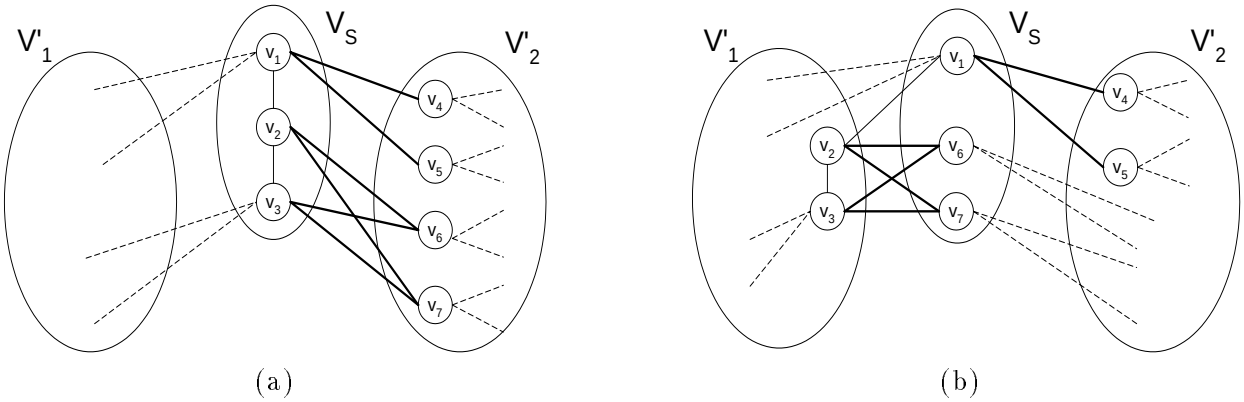

Figure 2.2: Two wide-to-narrow separator refinements induced by two optimal vertex covers.

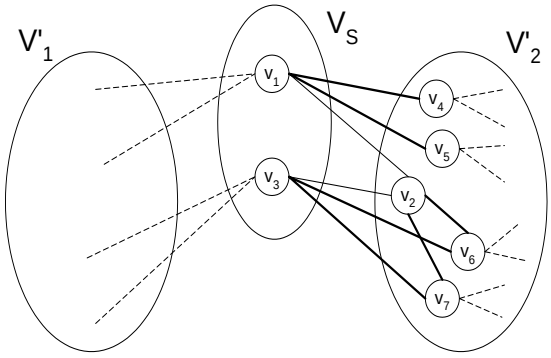

Figure 2.3: Optimal wide-to-narrow separator refinement.

part of the bipartite graph.

The non-optimality of the minimum vertex–cover model has been overlooked most probably because of the fact that loosely-bound vertices do not likely exist in the GPVS of graphs arising in finite difference and finite element applications.

Chapter 3

Hypergraph Models for 1D Decomposition

For parallel sparse-matrix vector product (SpMxV) using 1D decomposition, an $M \times M$ square sparse matrix $\mathbf{A}$ can be decomposed in two ways; *rowwise* or *columnwise*

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1^r \\ \vdots \\ \mathbf{A}_k^r \\ \vdots \\ \mathbf{A}_K^r \end{bmatrix} \qquad and \qquad \mathbf{A} = [\mathbf{A}_1^c \cdots \mathbf{A}_k^c \cdots \mathbf{A}_K^c],$$

where processor P_k owns row stripe $\mathbf{A}_k^r$ or column stripe $\mathbf{A}_k^c$, respectively, for a parallel system with K processors. As discussed in the introduction chapter, in order to avoid the communication of vector components during the linear vector operations, a symmetric partitioning scheme is adopted. That is, all vectors used in the solver are divided conformally with the row partitioning or the column partitioning in rowwise or columnwise decomposition schemes, respectively. In particular, the $\mathbf{x}$ and $\mathbf{y}$ vectors are divided as $[\mathbf{x}_1, \ldots, \mathbf{x}_K]^t$ and $[\mathbf{y}_1, \ldots, \mathbf{y}_K]^t$, respectively. In rowwise decomposition, processor P_k is responsible for computing $\mathbf{y}_k = \mathbf{A}_k^r \mathbf{x}$ and the linear operations on the k-th blocks of the vectors. In columnwise decomposition, processor P_k is responsible for computing $\mathbf{y}^k = \mathbf{A}_k^c \mathbf{x}_k$

(where $\mathbf{y} = \sum_{k=1}^{K} \mathbf{y}^k$) and the linear operations on the k-th blocks of the vectors. With these decomposition schemes, the linear vector operations can be easily and efficiently parallelized [10, 64], such that only the inner-product computations introduce global communication overhead of which its volume does not scale up with increasing problem size. In parallel SpMxV, the rowwise and columnwise decomposition schemes require communication before or after the local SpMxV computations, thus they can also be considered as *pre* and *post* communication schemes, respectively. Depending on the way in which the rows or columns of $\mathbf{A}$ are partitioned among the processors, entries in $\mathbf{x}$ or entries in $\mathbf{y}^k$ may need to be communicated among the processors. Unfortunately, the communication volume scales up with increasing problem size. Our goal is to find a rowwise or columnwise partition of $\mathbf{A}$ that minimizes the total volume of communication while maintaining the computational load balance.

The computational graph model is widely used in the representation of computational structures of various scientific applications, including repeated SpMxV computations, to decompose the computational domains for parallelization [14, 15, 41, 46, 50, 51, 60, 68]. In this model, the problem of 1D sparse matrix decomposition for minimizing the communication volume while maintaining the load balance is formulated as the well-known K-way graph partitioning problem. However, none of the graph models reflects the actual communication requirement as will be described in Section 3.2. In this work, we propose two *computational hypergraph* models which avoid all deficiencies of the graph model for 1D decomposition. The proposed hypergraph models reduce the decomposition problem to the well-known *K-way hypergraph partitioning* problem widely encountered in circuit partitioning in VLSI layout design.

Experimental results presented in Section 3.4 confirm the validity of our proposed hypergraph models. The hypergraph models using PaToH and hMeTiS produce 30%–38% better decompositions than the graph models using MeTiS, while the hypergraph models using PaToH are only 34%–130% slower than the graph models using the most recent version (Version 3.0) of MeTiS, on the average.

3.1 Graph Models for Sparse Matrix Decomposition

3.1.1 Standard Graph Model for Structurally Symmetric Matrices

A structurally symmetric sparse matrix $\mathbf{A}$ can be represented as an undirected graph $\mathcal{G}_A = (\mathcal{V}, \mathcal{E})$, where the sparsity pattern of $\mathbf{A}$ corresponds to the adjacency matrix representation of graph $\mathcal{G}_A$. That is, the vertices of $\mathcal{G}_A$ correspond to the rows/columns of matrix $\mathbf{A}$, and there exist an edge $e_{ij} \in \mathcal{E}$ for $i \neq j$ if and only if off-diagonal entries a_{ij} and a_{ji} of matrix $\mathbf{A}$ are nonzeros. In rowwise decomposition, each vertex $v_i \in \mathcal{V}$ corresponds to atomic task i of computing the inner product of row i with column vector $\mathbf{x}$. In columnwise decomposition, each vertex $v_i \in \mathcal{V}$ corresponds to atomic task i of computing the sparse SAXPY/DAXPY operation $\mathbf{y} = \mathbf{y} + x_i \mathbf{a}_{*i}$, where $\mathbf{a}_{*i}$ denotes column i of matrix $\mathbf{A}$. Hence, each nonzero entry in a row and column of $\mathbf{A}$ incurs a multiply-and-add operation during the local SpMxV computations in the pre and post communication schemes, respectively. Thus, computational load w_i of row/column i is the number of nonzero entries in row/column i. In graph theoretical notation, $w_i = d_i$ when $a_{ii} = 0$ and $w_i = d_i + 1$ when $a_{ii} \neq 0$. Note that the number of nonzeros in row i and column i are equal in a symmetric matrix.

This graph model displays a bidirectional computational interdependency view for SpMxV. Each edge $e_{ij} \in \mathcal{E}$ can be considered as incurring the computations $y_i \leftarrow y_i + a_{ij} \times x_j$ and $y_j \leftarrow y_j + a_{ji} \times x_i$. Hence, each edge represents the bidirectional interaction between the respective pair of vertices in both inner and outer product computation schemes for SpMxV. If rows (columns) i and j are assigned to the same processor in a rowwise (columnwise) decomposition, then edge e_{ij} does not incur any communication. However, in the pre-communication scheme, if rows i and j are assigned to different processors then cut edge e_{ij} necessitates the communication of two floating–point words because of the need of the exchange of updated x_i and x_j values between atomic tasks i and j just before the local SpMxV computations. In the post-communication scheme, if

columns i and j are assigned to different processors then cut edge e_{ij} necessitates the communication of two floating–point words because of the need of the exchange of partial y_i and y_j values between atomic tasks i and j just after the local SpMxV computations. Hence, by setting $c_{ij} = 2$ for each edge $e_{ij} \in \mathcal{E}$, both rowwise and columnwise decompositions of matrix $\mathbf{A}$ reduce to the K-way partitioning of its associated graph $\mathcal{G}_A$ according to the cutsize definition given in (2.2). Thus, minimizing the cutsize is an effort towards minimizing the total volume of interprocessor communication. Maintaining the balance criterion (2.1) corresponds to maintaining the computational load balance during local SpMxV computations.

Each vertex $v_i \in \mathcal{V}$ effectively represents both row i and column i in $\mathcal{G}_A$ although its atomic task definition differs in rowwise and columnwise decompositions. Hence, a partition Π of $\mathcal{G}_A$ automatically achieves a symmetric partitioning by inducing the same partition on the $\mathbf{y}$-vector and $\mathbf{x}$-vector components since a vertex $v_i \in \mathcal{P}_k$ corresponds to assigning row i (column i), y_i and x_i to the same part in rowwise (columnwise) decomposition.

In matrix theoretical view, the symmetric partitioning induced by a partition Π of $\mathcal{G}_A$ can also be considered as inducing a partial symmetric permutation on the rows and columns of $\mathbf{A}$. Here, the partial permutation corresponds to ordering the rows/columns assigned to part P_k before the rows/columns assigned to part P_{k+1}, for $k = 1, \ldots, K-1$, where the rows/columns within a part are ordered arbitrarily. Let $\mathbf{A}^{\Pi}$ denote the permuted version of $\mathbf{A}$ according to a partial symmetric permutation induced by Π. An internal edge e_{ij} of a part $\mathcal{P}_k$ corresponds to locating both a_{ij} and a_{ji} in diagonal block $\mathbf{A}^{\Pi}_{kk}$. An external edge e_{ij} of cost 2 between parts $\mathcal{P}_k$ and $\mathcal{P}_\ell$ corresponds to locating nonzero entry a_{ij} of $\mathbf{A}$ in off-diagonal block $\mathbf{A}^{\Pi}_{k\ell}$ and a_{ji} of $\mathbf{A}$ in off-diagonal block $\mathbf{A}^{\Pi}_{\ell k}$, or vice versa. Hence, minimizing the cutsize in the graph model can also be considered as permuting the rows and columns of the matrix to minimize the total number of nonzeros in the off-diagonal blocks.

Figure 3.1 illustrates a sample 10×10 symmetric sparse matrix $\mathbf{A}$ and its associated graph $\mathcal{G}_A$. The numbers inside the circles indicate the computational weights of the respective vertices (rows/columns). This figure also illustrates a

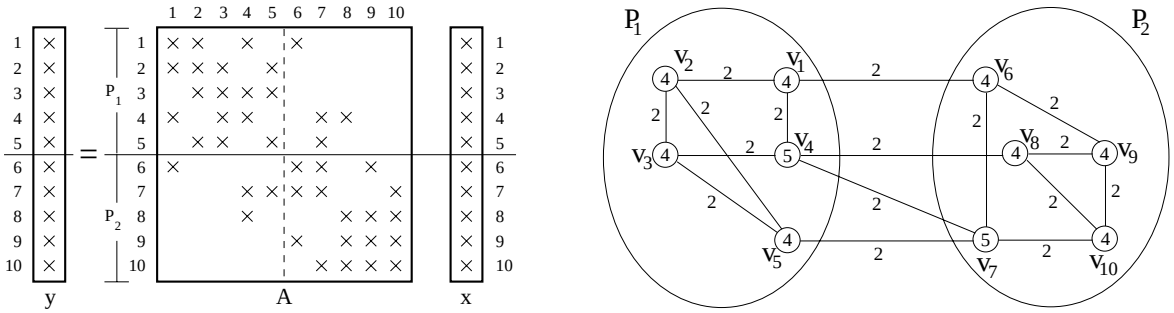

Figure 3.1: Two-way rowwise decomposition of a sample structurally symmetric matrix **A** and the corresponding bipartitioning of its associated graph $\mathcal{G}_A$.

rowwise decomposition of the symmetric **A** matrix and the corresponding bipartitioning of $\mathcal{G}_A$ for a two–processor system. As seen in Fig. 3.1, the cutsize in the given graph bipartitioning is 8 which is also equal to the total number of nonzero entries in the off-diagonal blocks. The bipartition illustrated in Fig. 3.1 achieves perfect load balance by assigning 21 nonzero entries to each row stripe. This number can also be obtained by adding the weights of the vertices in each part.

3.1.2 Bipartite Graph Model for Rectangular Matrices

The standard graph model is not suitable for the partitioning of nonsymmetric matrices. A recently proposed *bipartite* graph model [34, 49] enables the partitioning of rectangular as well as structurally symmetric/nonsymmetric square matrices. In this model, each row and column is represented by a vertex, and the sets of vertices representing the rows and columns form the bipartition, i.e. $\mathcal{V} = \mathcal{V}_\mathcal{R} \cup \mathcal{V}_\mathcal{C}$. There exists an edge between a row vertex $i \in \mathcal{V}_\mathcal{R}$ and a column vertex $j \in \mathcal{V}_\mathcal{C}$ if and only if the respective entry a_{ij} of matrix **A** is nonzero. Partitions $\Pi_\mathcal{R}$ and $\Pi_\mathcal{C}$ on $\mathcal{V}_\mathcal{R}$ and $\mathcal{V}_\mathcal{C}$, respectively, determine the overall partition $\Pi = \{\mathcal{P}_1, \ldots, \mathcal{P}_K\}$, where $\mathcal{P}_k = \mathcal{V}_{\mathcal{R}_k} \cup \mathcal{V}_{\mathcal{C}_k}$ for $k = 1, \ldots, K$. For rowwise (columnwise) decomposition, vertices in $\mathcal{V}_\mathcal{R}$ ($\mathcal{V}_\mathcal{C}$) are weighted with the number of nonzeros in the respective row (column) so that the balance criterion (2.1) is imposed only on the partitioning of $\mathcal{V}_\mathcal{R}$ ($\mathcal{V}_\mathcal{C}$). As in the standard graph model, minimizing the number of cut edges corresponds to minimizing the total number of nonzeros in the off-diagonal blocks.

This approach has the flexibility of achieving nonsymmetric partitioning. In the context of parallel SpMxV, the need for symmetric partitioning on square matrices is achieved by enforcing $\Pi_{\mathcal{R}} \equiv \Pi_{\mathcal{C}}$. Hendrickson and Kolda [34] propose several bipartite-graph partitioning algorithms that are adopted from the techniques for the standard graph model and one partitioning algorithm that is specific to bipartite graphs.

3.1.3 Proposed Generalized Graph Model for Structurally Symmetric/Nonsymmetric Square Matrices

In this work, we propose a simple yet effective graph model for symmetric partitioning of structurally nonsymmetric square matrices. The proposed model enables the use of the standard graph partitioning tools without any modification. In the proposed model, an nonsymmetric square matrix $\mathbf{A}$ is represented as an undirected graph $\mathcal{G}_{\mathcal{R}} = (\mathcal{V}_{\mathcal{R}}, \mathcal{E})$ and $\mathcal{G}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{E})$ for rowwise and columnwise decomposition schemes, respectively. Graphs $\mathcal{G}_{\mathcal{R}}$ and $\mathcal{G}_{\mathcal{C}}$ differ only in their vertex weight definitions. The vertex set and the corresponding atomic task definitions are identical to those of the symmetric matrices. Hence, computational weight w_i of a vertex $v_i \in \mathcal{V}_{\mathcal{R}}$ of $\mathcal{G}_{\mathcal{R}}$ and a vertex $v_i \in \mathcal{V}_{\mathcal{C}}$ of $\mathcal{G}_{\mathcal{C}}$ are equal to the total number of nonzeros in row i and column i in the pre and post communication schemes, respectively.

Both edge set and edge weight definitions are different than those of the symmetric matrices. In the edge set $\mathcal{E}$, $e_{ij} \in \mathcal{E}$ if and only if off-diagonal entries $a_{ij} \neq 0$ *or* $a_{ji} \neq 0$. That is, the vertices in the adjacency list of a vertex v_i denote the union of the column indices of the off-diagonal nonzeros at row i and the row indices of the off-diagonal nonzeros at column i. The cost c_{ij} of an edge e_{ij} is set to 1 if either $a_{ij} \neq 0$ or $a_{ji} \neq 0$, and it is set to 2 if both $a_{ij} \neq 0$ and $a_{ji} \neq 0$. Note that each row and column of matrix $\mathbf{A}$ are effectively represented by the same vertex as a simple means for enforcing symmetric permutation. The proposed scheme is referred to here as a generalized model since it automatically produces the existing graph representation for symmetric matrices by computing the same cost of 2 for every edge.

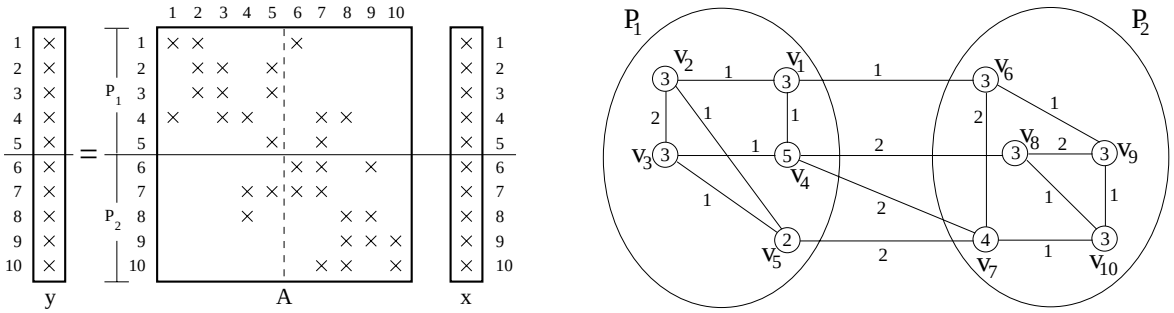

Figure 3.2: Two-way rowwise decomposition of a sample structurally nonsymmetric matrix **A** and the corresponding bipartitioning of its associated graph $\mathcal{G}_{\mathcal{R}}$.

In the proposed model, each edge with a cost of 2 represents the bidirectional interaction between the respective pair of vertices in an identical manner to that of the symmetric matrices. However, each edge with a cost of 1 represents an unidirectional interaction between the respective pair of vertices. That is, each edge $e_{ij} \in \mathcal{E}$ with $c_{ij} = 1$ incurs the computation of either $y_i \leftarrow y_i + a_{ij} \times x_j$ or $y_j \leftarrow y_j + a_{ji} \times x_i$, depending on whether $a_{ij} \neq 0$ or $a_{ji} \neq 0$, respectively. Hence, in inner–product computation scheme for SpMxV, an edge $e_{ij} \in \mathcal{E}$ with $c_{ij} = 1$ denotes the dependency relation of either atomic task i to x_j or atomic task j to x_i. A dual discussion holds for outer–product multiplication scheme. However, this ambiguity does not constitute any problem in the proposed model. If rows (columns) i and j are assigned to different processors in a rowwise (columnwise) decomposition, then cut edge e_{ij} with $c_{ij} = 1$ always necessitates the communication of a single floating–point word as follows. In rowwise decomposition, each cut edge $e_{ij} \in \mathcal{E}$ with $c_{ij} = 1$ incurs the communication of either updated x_i or x_j value just before the local SpMxV computations. In columnwise decomposition, each cut edge $e_{ij} \in \mathcal{E}$ with $c_{ij} = 1$ incurs the communication of either partial y_i or y_j result just after the local SpMxV computations.

Figure 3.2 illustrates a sample 10×10 nonsymmetric sparse matrix **A** and its associated graph $\mathcal{G}_{\mathcal{R}}$ for rowwise decomposition. The numbers inside the circles indicate the computational weights of the respective vertices (rows). This figure also illustrates rowwise decomposition of the matrix and the corresponding bipartitioning of its associated graph for a two–processor system. As seen in Fig. 3.2, the cutsize of the given graph bipartitioning is 7 which is also equal to the total number of nonzero entries in the off-diagonal blocks. Hence, similar to the

symmetric matrix case, minimizing cutsize in the proposed graph model can also be considered as permuting the rows and columns of the matrix to minimize the total number of off-block-diagonal nonzeros. As seen in Fig. 3.2, the bipartitioning achieves perfect load balance by assigning 16 nonzero entries to each row part. As mentioned earlier, $\mathcal{G}_{\mathcal{C}}$ model of sample matrix $\mathbf{A}$ for columnwise decomposition differs from $\mathcal{G}_{\mathcal{R}}$ only in vertex weights. Hence, the graph bipartitioning illustrated in Fig. 3.2 can also be considered as incurring a slightly imbalanced (15 versus 17 nonzeros) columnwise decomposition of sample matrix $\mathbf{A}$ (shown by vertical dash line) with identical communication requirement.

The storage requirement of the standard and proposed graph models is as follows. For an $M \times M$ square matrix with Z off-diagonal nonzero entries, the graph models contain $|\mathcal{V}| = M$ vertices for both symmetric and nonsymmetric matrices. The graph model contains exactly $|\mathcal{E}| = Z/2$ edges for symmetric matrices. However, the number of edges in the graph model for nonsymmetric matrices may vary between $Z/2$ and Z (i.e., $Z/2 < |\mathcal{E}| \leq Z$), because every symmetric pair off-diagonal nonzeros $a_{ij} \neq 0$ and $a_{ji} \neq 0$ in an nonsymmetric matrix decrease the number of edges by 1 from Z towards $Z/2$. In the graph models of both symmetric and nonsymmetric matrices, M words are required to store vertex weights, and M words are needed to store the starting indices of the adjacency lists. There is no need to store the edge costs for symmetric matrices since all edge costs are equal to 2, therefore Z words suffices to store $|\mathcal{E}| = Z/2$ edges as each edge has to be stored twice in the adjacency list representation. However, edge costs have to be stored in the graph model for nonsymmetric matrices because of different edge costs of 2 and 1. Therefore, the storage requirement of the graph models is $S_{\mathcal{G}} = 2M + Z$ words for symmetric matrices, whereas it may vary between $2M+2Z$ and $2M+4Z$ words for nonsymmetric matrices (i.e., $2M+2Z < S_{\mathcal{G}} \leq 2M+4Z$).

3.2 Flaws of the Graph Models

Consider the symmetric matrix decomposition given in Fig. 3.1. Assume that parts $\mathcal{P}_1$ and $\mathcal{P}_2$ are mapped to processors P_1 and P_2, respectively. The cutsize

of the bipartition shown in this figure is equal to $2\times4=8$, thus estimating the communication volume requirement as 8 words. In the pre-communication scheme, off-block-diagonal entries $a_{4,7}$ and $a_{5,7}$ assigned to processor P_1 display the same need for the nonlocal $\mathbf{x}$-vector component x_7 twice. However, it is clear that processor P_2 will send x_7 only once to processor P_1. Similarly, processor P_1 will send x_4 only once to processor P_2 because of the off-block-diagonal entries $a_{7,4}$ and $a_{8,4}$ assigned to processor P_2. In the post-communication scheme, the graph model treats the off-block-diagonal nonzeros $a_{7,4}$ and $a_{7,5}$ in $\mathcal{P}_1$ as if processor P_1 will send two multiplication results $a_{7,4}x_4$ and $a_{7,5}x_5$ to processor P_2. However, it is obvious that processor P_1 will compute the partial result for the nonlocal $\mathbf{y}$-vector component $y'_7=a_{7,4}x_4+a_{7,5}x_5$ during the local SpMxV phase and send this single value to processor P_2 during the post-communication phase. Similarly, processor P_2 will only compute and send the single value $y'_4=a_{4,7}x_7+a_{4,8}x_8$ to processor P_1. Hence, the actual communication volume is in fact 6 words instead of 8 in both pre and post communication schemes. A similar analysis of the rowwise decomposition of the nonsymmetric matrix given in Fig. 3.2 reveals the fact that the actual communication requirement is 5 words (x_4, x_5, x_6, x_7 and x_8) instead of 7 determined by the cutsize of the given bipartition of $\mathcal{G}_{\mathcal{R}}$.

In matrix theoretical view, the nonzero entries in the same column of an off-diagonal block incur the communication of a single x value in the rowwise decomposition (pre-communication) scheme. Similarly, the nonzero entries in the same row of an off-diagonal block incur the communication of a single y value in the columnwise decomposition (post-communication) scheme. However, as mentioned earlier, the graph models try to minimize the total number of off-block-diagonal nonzeros without considering the relative spatial locations of such nonzeros. In other words, the graph models treat all off-block-diagonal nonzeros in an identical manner by assuming that each off-block-diagonal nonzero will incur a distinct communication of a single word.

In graph theoretical view, the graph models treat all cut edges of equal cost in an identical manner while computing the cutsize. However, r cut edges, each of cost 2, stemming from a vertex v_{i_1} in part $\mathcal{P}_k$ to r vertices $v_{i_2}, v_{i_3}, \ldots, v_{i_{r+1}}$ in part $\mathcal{P}_\ell$ incur only $r+1$ communications instead of $2r$ in both pre and post communication schemes. In the pre-communication scheme, processor P_k sends x_{i_1} to

processor P_ℓ while P_ℓ sends $x_{i_2}, x_{i_3}, \ldots, x_{i_{r+1}}$ to P_k. In the post-communication scheme, processor P_ℓ sends $y'_{i_2}, y'_{i_3}, \ldots, y'_{i_{r+1}}$ to processor P_k while P_k sends y'_{i_1} to P_ℓ. Similarly, the amount of communication required by r cut edges, each of cost 1, stemming from a vertex v_{i_1} in part $\mathcal{P}_k$ to r vertices $v_{i_2}, v_{i_3}, \ldots, v_{i_{r+1}}$ in part $\mathcal{P}_\ell$ may vary between 1 and r words instead of exactly r words determined by the cutsize of the given graph partitioning.

3.3 Two Hypergraph Models for 1D Decomposition

We propose two computational hypergraph models for the decomposition of sparse matrices. These models are referred to here as the *column-net* and *row-net* models proposed for the rowwise decomposition (pre-communication) and columnwise decomposition (post-communication) schemes, respectively.

In the column-net model, matrix $\mathbf{A}$ is represented as a hypergraph $\mathcal{H}_\mathcal{R} = (\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ for rowwise decomposition. Vertex and net sets $\mathcal{V}_\mathcal{R}$ and $\mathcal{N}_\mathcal{C}$ correspond to the rows and columns of matrix $\mathbf{A}$, respectively. There exist one vertex v_i and one net n_j for each row i and column j, respectively. Net $n_j \subseteq \mathcal{V}_\mathcal{R}$ contains the vertices corresponding to the rows which have a nonzero entry in column j. That is, $v_i \in n_j$ if and only if $a_{ij} \neq 0$. Each vertex $v_i \in \mathcal{V}_\mathcal{R}$ corresponds to atomic task i of computing the inner product of row i with column vector $\mathbf{x}$. Hence, computational weight w_i of a vertex $v_i \in \mathcal{V}_\mathcal{R}$ is equal to the total number of nonzeros in row i. The nets of $\mathcal{H}_\mathcal{R}$ represent the *dependency* relations of the atomic tasks on the $\mathbf{x}$-vector components in rowwise decomposition. Each net n_j can be considered as incurring the computations $y_i \leftarrow y_i + a_{ij} \cdot x_j$ for each vertex (row) $v_i \in n_j$. Hence, each net n_j denotes the set of atomic tasks (vertices) that need x_j. Note that each pin v_i of a net n_j corresponds to a unique nonzero a_{ij} thus enabling the representation and decomposition of structurally nonsymmetric matrices as well as symmetric matrices without any extra effort. Figure 3.3(a) illustrates the dependency relation view of the column-net model. As seen in this figure, net $n_j = \{v_h, v_i, v_k\}$ represents the dependency of atomic tasks h, i, k to x_j because of the computations $y_h \leftarrow y_h + a_{hj} \cdot x_j$, $y_i \leftarrow y_i + a_{ij} \cdot x_j$

and $y_k \leftarrow y_k + a_{kj} \cdot x_j$. Figure 3.4(b) illustrates the column-net representation of the sample 16×16 nonsymmetric matrix given in Fig. 3.4(a). In Fig. 3.4(b), the pins of net $n_7 = \{v_7, v_{10}, v_{13}\}$ represent nonzeros $a_{7,7}$, $a_{10,7}$, and $a_{13,7}$. Net n_7 also represents the dependency of atomic tasks 7, 10 and 13 to x_7 because of the computations $y_7 \leftarrow y_7 + a_{7,7} \cdot x_7$, $y_{10} \leftarrow y_{10} + a_{10,7} \cdot x_7$ and $y_{13} \leftarrow y_{13} + a_{13,7} \cdot x_7$.

The row-net model can be considered as the dual of the column-net model. In this model, matrix $\mathbf{A}$ is represented as a hypergraph $\mathcal{H}_{\mathcal{C}} = (\mathcal{V}_{\mathcal{C}}, \mathcal{N}_{\mathcal{R}})$ for columnwise decomposition. Vertex and net sets $\mathcal{V}_{\mathcal{C}}$ and $\mathcal{N}_{\mathcal{R}}$ correspond to the columns and rows of matrix $\mathbf{A}$, respectively. There exist one vertex v_i and one net n_j for each column i and row j, respectively. Net $n_j \subseteq \mathcal{V}_{\mathcal{C}}$ contains the vertices corresponding to the columns which have a nonzero entry on row j. That is, $v_i \in n_j$ if and only if $a_{ji} \neq 0$. Each vertex $v_i \in \mathcal{V}_{\mathcal{C}}$ corresponds to atomic task i of computing the sparse SAXPY/DAXPY operation $\mathbf{y} = \mathbf{y} + x_i \mathbf{a}_{*i}$. Hence, computational weight w_i of a vertex $v_i \in \mathcal{V}_{\mathcal{C}}$ is equal to the total number of nonzeros in column i. The nets of $\mathcal{H}_{\mathcal{C}}$ represent the *dependency* relations of the computations of the $\mathbf{y}$-vector components to the atomic tasks represented by the vertices of $\mathcal{H}_{\mathcal{C}}$ in columnwise decomposition. Each net n_j can be considered as incurring the computation $y_j \leftarrow y_j + a_{ji} \cdot x_i$ for each vertex (column) $v_i \in n_j$. Hence, each net n_j denotes the set of atomic task results needed to accumulate y_j. Note that each pin v_i of a net n_j corresponds to a unique nonzero a_{ji} thus enabling the representation and decomposition of structurally nonsymmetric matrices as well as symmetric matrices without any extra effort. Figure 3.3(b) illustrates the dependency relation view of the row-net model. As seen in this figure, net $n_j = \{v_h, v_i, v_k\}$ represents the dependency of accumulating $y_j = y_j^h + y_j^i + y_j^k$ to the partial y_j results $y_j^h = a_{jh} \cdot x_h$, $y_j^i = a_{ji} \cdot x_i$ and $y_j^k = a_{jk} \cdot x_k$. Note that the row-net and column-net models become identical in structurally symmetric matrices.

By assigning unit costs to the nets (i.e. $c_j = 1$ for each net n_j), the proposed column-net and row-net models reduce the decomposition problem to the K-way hypergraph partitioning problem according to the cutsize definition given in (2.4.b) for the pre and post communication schemes, respectively. Consistency of the proposed hypergraph models for accurate representation of communication volume requirement while maintaining the symmetric partitioning restriction depends on the condition that "$v_j \in n_j$ for each net n_j". We first assume that this

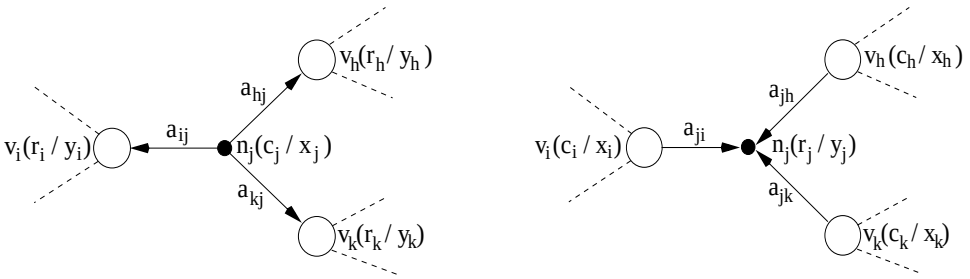

Figure 3.3: Dependency relation views of (a) column-net and (b) row-net models.

condition holds in the discussion throughout the following four paragraphs and then discuss the appropriateness of the assumption in the last paragraph of this section.

The validity of the proposed hypergraph models is discussed only for the column-net model. A dual discussion holds for the row-net model. Consider a partition Π of $\mathcal{H}_{\mathcal{R}}$ in the column-net model for rowwise decomposition of a matrix $\mathbf{A}$. Without loss of generality, we assume that part $\mathcal{P}_k$ is assigned to processor P_k for $k = 1, 2, \ldots, K$. As Π is defined as a partition on the vertex set of $\mathcal{H}_{\mathcal{R}}$, it induces a complete part (hence processor) assignment for the rows of matrix $\mathbf{A}$ and hence for the components of the $\mathbf{y}$ vector. That is, a vertex v_i assigned to part $\mathcal{P}_k$ in Π corresponds to assigning row i and y_i to part $\mathcal{P}_k$. However, partition Π does not induce any part assignment for the nets of $\mathcal{H}_{\mathcal{R}}$. Here, we consider partition Π as inducing an assignment for the internal nets of $\mathcal{H}_{\mathcal{R}}$ hence for the respective $\mathbf{x}$-vector components. Consider an internal net n_j of part $\mathcal{P}_k$ (i.e. $\Lambda_j = \{\mathcal{P}_k\}$) which corresponds to column j of $\mathbf{A}$. As all pins of net n_j lie in $\mathcal{P}_k$, all rows (including row j by the consistency condition) which need x_j for inner-product computations are already assigned to processor P_k. Hence, internal net n_j of $\mathcal{P}_k$, which does not contribute to the cutsize (2.4.b) of partition Π, does not necessitate any communication if x_j is assigned to processor P_k. The assignment of x_j to processor P_k can be considered as permuting column j to part $\mathcal{P}_k$, thus respecting the symmetric partitioning of $\mathbf{A}$ since row j is already assigned to $\mathcal{P}_k$. In the 4-way decomposition given in Fig. 3.4(b), internal nets n_1, n_{10}, n_{13} of part $\mathcal{P}_1$ induce the assignment of x_1, x_{10}, x_{13} and columns 1, 10, 13 to part $\mathcal{P}_1$. Note that part $\mathcal{P}_1$ already contains rows 1, 10, 13 thus respecting the symmetric partitioning of $\mathbf{A}$.

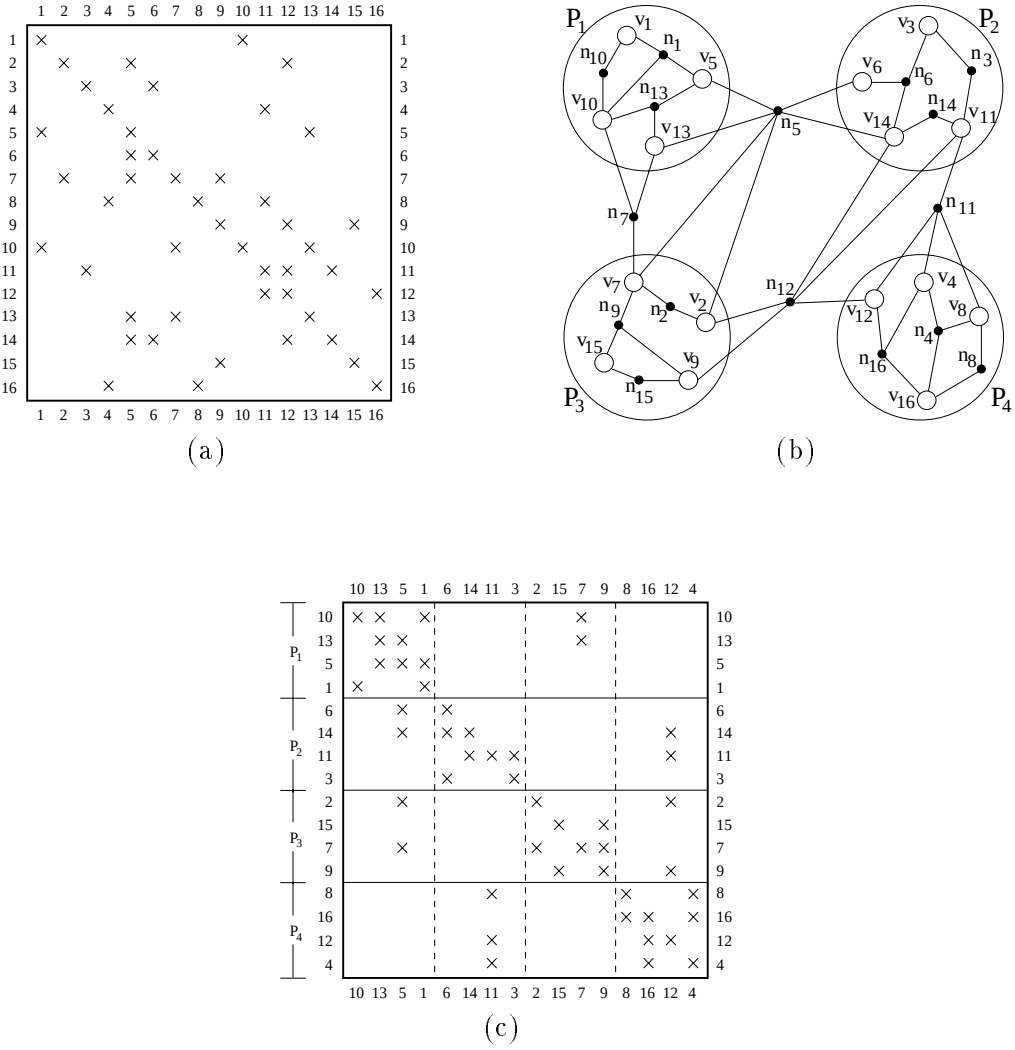

Figure 3.4: (a) A 16×16 structurally nonsymmetric matrix $\mathbf{A}$. (b) Column-net representation $\mathcal{H}_{\mathcal{R}}$ of matrix $\mathbf{A}$ and 4-way partitioning Π of $\mathcal{H}_{\mathcal{R}}$. (c) 4-way rowwise decomposition of matrix $\mathbf{A}^{\Pi}$ obtained by permuting $\mathbf{A}$ according to the symmetric partitioning induced by Π.

Consider an external net n_j with connectivity set Λ_j, where $\lambda_j = |\Lambda_j|$ and $\lambda_j > 1$. As all pins of net n_j lie in the parts in its connectivity set Λ_j, all rows (including row j by the consistency condition) which need x_j for inner-product computations are assigned to the parts (processors) in Λ_j. Hence, contribution $\lambda_j - 1$ of external net n_j to the cutsize according to (2.4.b) accurately models the amount of communication volume to incur during the parallel SpMxV computations because of x_j if x_j is assigned to any processor in Λ_j. Let $map[j] \in \Lambda_j$ denote the part and hence processor assignment for x_j corresponding to cut net n_j. In the column-net model together with the pre-communication scheme, cut net n_j indicates that processor $map[j]$ should send its local x_j to those processors in connectivity set Λ_j of net n_j except itself (i.e., to processors in the set $\Lambda_j - \{map[j]\}$). Hence, processor $map[j]$ should send its local x_j to $|\Lambda_j| - 1 = \lambda_j - 1$ distinct processors. As the consistency condition "$v_j \in n_j$" ensures that row j is already assigned to a part in Λ_j, symmetric partitioning of $\mathbf{A}$ can easily be maintained by assigning x_j hence permuting column j to the part which contains row j. In the 4-way decomposition shown in Fig. 3.4(b), external net n_5 (with $\Lambda_5 = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$) incurs the assignment of x_5 (hence permuting column 5) to part $\mathcal{P}_1$ since row 5 ($v_5 \in n_5$) is already assigned to part $\mathcal{P}_1$. The contribution $\lambda_5 - 1 = 2$ of net n_5 to the cutsize accurately models the communication volume to incur due to x_5, because processor P_1 should send x_5 to both processors P_2 and P_3 only once since $\Lambda_5 - \{map[5]\} = \Lambda_5 - \{P_1\} = \{P_2, P_3\}$.

In essence, in the column-net model, any partition Π of $\mathcal{H}_\mathcal{R}$ with $v_i \in \mathcal{P}_k$ can be safely decoded as assigning row i, y_i and x_i to processor P_k for rowwise decomposition. Similarly, in the row-net model, any partition Π of $\mathcal{H}_\mathcal{C}$ with $v_i \in \mathcal{P}_k$ can be safely decoded as assigning column i, x_i and y_i to processor P_k for columnwise decomposition. Thus, in the column-net and row-net models, minimizing the cutsize according to (2.4.b) corresponds to minimizing the actual volume of interprocessor communication during the pre and post communication phases, respectively. Maintaining the balance criterion (2.1) corresponds to maintaining the computational load balance during the local SpMxV computations. Figure 3.4(c) displays a permutation of the sample matrix given in Fig. 3.4(a) according to the symmetric partitioning induced by the 4-way decomposition shown in Fig. 3.4(b). As seen in Fig. 3.4(c), the actual communication volume for the given rowwise decomposition is 6 words since processor P_1 should send x_5 to

both P_2 and P_3, P_2 should send x_{11} to P_4, P_3 should send x_7 to P_1, and P_4 should send x_{12} to both P_2 and P_3. As seen in Fig. 3.4(b), external nets n_5, n_7, n_{11} and n_{12} contribute 2, 1, 1 and 2 to the cutsize since $\lambda_5 = 3$, $\lambda_7 = 2$, $\lambda_{11} = 2$ and $\lambda_{12} = 3$, respectively. Hence, the cutsize of the 4-way decomposition given in Fig. 3.4(b) is 6, thus leading to the accurate modeling of the communication requirement. Note that the graph model will estimate the total communication volume as 13 words for the 4-way decomposition given in Fig. 3.4(c) since the total number of nonzeros in the off-diagonal blocks is 13. As seen in Fig. 3.4(c), each processor is assigned 12 nonzeros thus achieving perfect computational load balance.

In matrix theoretical view, let $\mathbf{A}^{\Pi}$ denote a permuted version of matrix $\mathbf{A}$ according to the symmetric partitioning induced by a partition Π of $\mathcal{H}_{\mathcal{R}}$ in the column-net model. Each cut-net n_j with connectivity set Λ_j and $map[j] = \mathcal{P}_\ell$ corresponds to column j of $\mathbf{A}$ containing nonzeros in λ_j distinct blocks ($\mathbf{A}^{\Pi}_{k\ell}$, for $\mathcal{P}_k \in \Lambda_j$) of matrix $\mathbf{A}^{\Pi}$. Since connectivity set Λ_j of net n_j is guaranteed to contain part $map[j]$, column j contains nonzeros in $\lambda_j - 1$ distinct off-diagonal blocks of $\mathbf{A}^{\Pi}$. Note that multiple nonzeros of column j in a particular off-diagonal block contributes only one to connectivity λ_j of net n_j by definition of λ_j. So, the cutsize of a partition Π of $\mathcal{H}_R$ is equal to the number of nonzero column segments in the off-diagonal blocks of matrix $\mathbf{A}^{\Pi}$. For example, external net n_5 with $\Lambda_5 = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ and $map[5] = \mathcal{P}_1$ in Fig. 3.4(b) indicates that column 5 has nonzeros in two off-diagonal blocks $\mathbf{A}^{\Pi}_{2,1}$ and $\mathbf{A}^{\Pi}_{3,1}$ as seen in Fig. 3.4(c). As also seen in Fig. 3.4(c), the number of nonzero column segments in the off-diagonal blocks of matrix $\mathbf{A}^{\Pi}$ is 6 which is equal to the cutsize of partition Π shown in Fig. 3.4(b). Hence, the column-net model tries to achieve a symmetric permutation which minimizes the total number of nonzero column segments in the off-diagonal blocks for the pre-communication scheme. Similarly, the row-net model tries to achieve a symmetric permutation which minimizes the total number of nonzero row segments in the off-diagonal blocks for the post-communication scheme.

Nonzero diagonal entries automatically satisfy the condition "$v_j \in n_j$ for each net n_j", thus enabling both accurate representation of communication requirement and symmetric partitioning of $\mathbf{A}$. A nonzero diagonal entry a_{jj} already

implies that net n_j contains vertex v_j as its pin. If however some diagonal entries of the given matrix are zeros then the consistency of the proposed column-net model is easily maintained by simply adding rows, which do not contain diagonal entries, to the pin lists of the respective column nets. That is, if $a_{jj} = 0$ then vertex v_j (row j) is added to the pin list $pins[n_j]$ of net n_j and net n_j is added to the net list $nets[v_j]$ of vertex v_j. These pin additions do not affect the computational weight assignments of the vertices. That is, weight w_j of vertex v_j in $\mathcal{H}_{\mathcal{R}}$ becomes equal to either d_j or $d_j - 1$ depending on whether $a_{jj} \neq 0$ or $a_{jj} = 0$, respectively. The consistency of the row-net model is preserved in a dual manner.

3.4 Experimental Results

We have tested the validity of the proposed hypergraph models by running MeTiS on the graphs obtained by randomized clique-net transformation, and running PaToH and hMeTiS directly on the hypergraphs for the decompositions of various realistic sparse test matrices arising in different application domains. These decomposition results are compared with the decompositions obtained by running MeTiS using the standard and proposed graph models for the symmetric and nonsymmetric test matrices, respectively. The most recent version (Version 3.0) of MeTiS [44] was used in the experiments. As both hMeTiS and PaToH achieve K-way partitioning through recursive bisection, recursive MeTiS (pMeTiS) was used for the sake of a fair comparison. Another reason for using pMeTiS is that direct K-way partitioning version of MeTiS (kMeTiS) produces 9% worse partitions than pMeTiS in the decomposition of the nonsymmetric test matrices, although it is 2.5 times faster, on the average. pMeTiS was run with the default parameters: sorted heavy-edge matching, region growing and early-exit boundary FM refinement for coarsening, initial partitioning and uncoarsening phases, respectively. The current version (Version 1.0.2) of hMeTiS [47] was run with the parameters: greedy first-choice scheme (GFC) and early-exit FM refinement (EE-FM) for coarsening and uncoarsening phases, respectively. The V-cycle refinement scheme was not used, because in our experimentations it achieved at most 1% (much less on the average) better decompositions at the expense of approximately 3 times slower execution time (on the average) in the decomposition

of the test matrices. The GFC scheme was found to be 28% faster than the other clustering schemes while producing slightly (1%–2%) better decompositions on the average. The EE-FM scheme was observed to be 30% faster than the other refinement schemes without any difference in the decomposition quality on the average.

Table 3.1 illustrates the properties of the test matrices listed in the order of increasing number of nonzeros. In this table, the "description" column displays both the nature and the source of each test matrix. The sparsity patterns of the Linear Programming matrices used as symmetric test matrices are obtained by multiplying the respective rectangular constraint matrices with their transposes. In Table 3.1, the total number of nonzeros of a matrix also denotes the total number of pins in both column-net and row-net models. The minimum and maximum number of nonzeros per row (column) of a matrix correspond to the minimum and maximum vertex degree (net size) in the column-net model, respectively. Similarly, the standard deviation *std* and coefficient of variation *cov* values of nonzeros per row (column) of a matrix correspond to the *std* and *cov* values of vertex degree (net size) in the column-net model, respectively. Dual correspondences hold for the row-net model.

All experiments were carried out on a workstation equipped with a 133 MHz PowerPC processor with 512-Kbyte external cache and 64 Mbytes of memory. We have tested $K = 8$, 16, 32 and 64 way decompositions of every test matrix. For a specific K value, K-way decomposition of a test matrix constitutes a decomposition instance. pMeTiS, hMeTiS and PaToH were run 50 times starting from different random seeds for each decomposition instance. The average performance results are displayed in Tables 3.2–3.4 and Figs. 3.5–3.7 for each decomposition instance. The percent load imbalance values are below 3% for all decomposition results displayed in these figures, where percent imbalance ratio is defined as $100 \times (W_{max} - W_{avg})/W_{avg}$.

Table 3.2 displays the decomposition performance of the proposed hypergraph

models together with the standard graph model in the rowwise/columnwise decomposition of the symmetric test matrices. Note that the rowwise and columnwise decomposition problems become equivalent for symmetric matrices. Tables 3.3 and 3.4 display the decomposition performance of the proposed column-net and row-net hypergraph models together with the proposed graph models in the rowwise and columnwise decompositions of the nonsymmetric test matrices, respectively. Due to lack of space, the decomposition performance results for the clique-net approach are not displayed in Tables 3.2–3.4, instead they are summarized in Table 3.5. Although the main objective of this work is the minimization of the total communication volume, the results for the other performance metrics such as the maximum volume, average number and maximum number of messages handled by a single processor are also displayed in Tables 3.2–3.4. Note that the maximum volume and maximum number of messages determine the concurrent communication volume and concurrent number of messages, respectively, under the assumption that no congestion occurs in the network.

As seen in Tables 3.2–3.4, the proposed hypergraph models produce substantially better partitions than the graph model at each decomposition instance in terms of total communication volume cost. In the symmetric test matrices, the hypergraph model produces 7%–48% better partitions than the graph model (see Table 3.2). In the nonsymmetric test matrices, the hypergraph models produce 12%–63% and 9%–56% better partitions than the graph models in the rowwise (see Table 3.3) and columnwise (see Table 3.4) decompositions, respectively. As seen in Tables 3.2–3.4, there is no clear winner between hMeTiS and PaToH in terms of decomposition quality. In some matrices hMeTiS produces slightly better partitions than PaToH, whereas the situation is the other way round in some other matrices. As seen in Tables 3.2 and 3.3, there is also no clear winner between matching-based clustering scheme HCM and agglomerative clustering scheme HCC in PaToH (see Section 6.1 for detailed discussion of clustering schemes). However, as seen in Table 3.4, PaToH-HCC produces slightly better partitions than PaToH-HCM in all columnwise decomposition instances for the nonsymmetric test matrices.

Tables 3.2–3.4 show that the performance gap between the graph and hypergraph models in terms of the total communication volume costs is preserved by

almost the same amounts in terms of the concurrent communication volume costs. For example, in the decomposition of the symmetric test matrices, the hypergraph model using PaToH-HCM incurs 30% less total communication volume than the graph model while incurring 28% less concurrent communication volume, on the overall average. In the columnwise decomposition of the nonsymmetric test matrices, PaToH-HCM incurs 35% less total communication volume than the graph model while incurring 37% less concurrent communication volume, on the overall average.

Although the hypergraph models perform better than the graph models in terms of number of messages, the performance gap is not as large as in the communication volume metrics. However, the performance gap increases with increasing K. As seen in Table 3.2, in the 64-way decomposition of the symmetric test matrices, the hypergraph model using PaToH-HCC incurs 32% and 10% less total and concurrent number of messages than the graph model, respectively. As seen in Table 3.3, in the rowwise decomposition of the nonsymmetric test matrices, PaToH-HCC incurs 32% and 26% less total and concurrent number of messages than the graph model, respectively.

The performance comparison of the graph/hypergraph partitioning based 1D decomposition schemes with the conventional algorithms based on 1D and 2D [37, 56, 55] decomposition schemes is as follows. As mentioned earlier, in K-way decompositions of $M \times M$ matrices, the conventional 1D and 2D schemes incur the total communication volume of $(K-1)M$ and $2(\sqrt{K}-1)M$ words, respectively. For example, in 64-way decompositions, the conventional 1D and 2D schemes incur the total communication volumes of $63M$ and $14M$ words, respectively. As seen at the bottom of Tables 3.2 and 3.3, PaToH-HCC reduces the total communication volume to $1.91M$ and $0.90M$ words in the 1D 64-way decomposition of the symmetric and nonsymmetric test matrices, respectively, on the overall average. In 64-way decompositions, the conventional 1D and 2D schemes incur the concurrent communication volumes of approximately M and $0.22M$ words, respectively. As seen in Tables 3.2 and 3.3, PaToH-HCC reduces the concurrent communication volume to $0.052M$ and $0.025M$ words in the 1D 64-way decomposition of the symmetric and nonsymmetric test matrices, respectively, on the overall average.

Figure 3.5 illustrates the relative run-time performance of the proposed hypergraph model compared to the standard graph model in the rowwise/columnwise decomposition of the symmetric test matrices. Figures 3.6 and 3.7 display the relative run-time performance of the column-net and row-net hypergraph models compared to the proposed graph models in the rowwise and columnwise decompositions of the nonsymmetric test matrices, respectively. In Figs. 3.5–3.7, for each decomposition instance, we plot the ratios of the average execution times of the tools using the respective hypergraph model to that of pMeTiS using the respective graph model. The results displayed in Figs. 3.5–3.7 are obtained by assuming that the test matrix is given either in CSR or in CSC form which are commonly used for SpMxV computations. The standard graph model does not necessitate any preprocessing since CSR and CSC forms are equivalent in symmetric matrices and both of them correspond to the adjacency list representation of the standard graph model. However, in nonsymmetric matrices, construction of the proposed graph model requires some amount of preprocessing time, although we have implemented a very efficient construction code which totally avoids index search. Thus, the execution time averages of the graph models for the nonsymmetric test matrices include this preprocessing time. The preprocessing time constitutes approximately 3% of the total execution time on the overall average. In the clique-net model, transforming the hypergraph representation of the given matrices to graphs using the randomized clique-net model introduces considerable amount of preprocessing time, despite the efficient implementation scheme we have adopted. Hence, the execution time averages of the clique-net model include this transformation time. The transformation time constitutes approximately 23% of the total execution time on the overall average. As mentioned earlier, the PaToH and hMeTiS tools use both CSR and CSC forms such that the construction of the other form from the given one is performed within the respective tool.

As seen in Figs. 3.5–3.7, the tools using the hypergraph models run slower than pMeTiS using the the graph models in most of the instances. The comparison of Fig. 3.5 with Figs. 3.6 and 3.7 shows that the gap between the run-time performances of the graph and hypergraph models is much less in the decomposition of the nonsymmetric test matrices than that of the symmetric test matrices. These experimental findings were expected, because the execution times

of graph partitioning tool pMeTiS, and hypergraph partitioning tools hMeTiS and PaToH are proportional to the sizes of the graph and hypergraph, respectively. In the representation of an $M \times M$ square matrix with Z off-diagonal nonzeros, the graph models contain $|\mathcal{E}| = Z/2$ and $Z/2 < |\mathcal{E}| \leq Z$ edges for symmetric and nonsymmetric matrices, respectively. However, the hypergraph models contain $p = M + Z$ pins for both symmetric and nonsymmetric matrices. Hence, the size of the hypergraph representation of a matrix is always greater than the size of its graph representation, and this gap in the sizes decreases in favor of the hypergraph models in nonsymmetric matrices. Figure 3.7 displays an interesting behavior that pMeTiS using the clique-net model runs faster than pMeTiS using the graph model in the columnwise decomposition of 4 out of 9 nonsymmetric test matrices. In these 4 test matrices, the edge contractions during the hypergraph-to-graph transformation through randomized clique-net approach lead to less number of edges than the graph model.

As seen in Figs. 3.5–3.7, both PaToH-HCM and PaToH-HCC run considerably faster than hMeTiS in each decomposition instance. This situation can be most probably due to the design considerations of hMeTiS. hMeTiS mainly aims at partitioning VLSI circuits of which hypergraph representations are much more sparse than the hypergraph representations of the test matrices. In the comparison of the HCM and HCC clustering schemes of PaToH, PaToH-HCM runs slightly faster than PaToH-HCC in the decomposition of almost all test matrices except in the decomposition of symmetric matrices KEN-11 and KEN-13, and nonsymmetric matrices ONETONE1 and ONETONE2. As seen in Fig. 3.5, PaToH-HCM using the hypergraph model runs 1.47–2.93 times slower than pMeTiS using the graph model in the decomposition of the symmetric test matrices. As seen in Figs. 3.6 and 3.7, PaToH-HCM runs 1.04–1.63 times and 0.83–1.79 times slower than pMeTiS using the graph model in the rowwise and columnwise decomposition of the nonsymmetric test matrices, respectively. Note that PaToH-HCM runs 17%, 8% and 6% faster than pMeTiS using the graph model in the 8-way, 16-way and 32-way columnwise decompositions of nonsymmetric matrix LHR34, respectively. PaToH-HCM achieves 64-way rowwise decomposition of the largest test matrix BCSSTK32 containing 44.6K rows/columns and 1030K nonzeros in only 25.6 seconds, which is equal to the sequential execution time of multiplying matrix BCSSTK32 with a dense vector 73.5 times.

The relative performance results of the hypergraph models with respect to the graph models are summarized in Table 3.5 in terms of total communication volume and execution time by averaging over different K values. This table also displays the averages of the best and worst performance results of the tools using the hypergraph models. In Table 3.5, the performance results for the hypergraph models are normalized with respect to those of pMeTiS using the graph models. In the symmetric test matrices, direct approaches PaToH and hMeTiS produce 30%–32% better partitions than pMeTiS using the graph model, whereas the clique-net approach produces 16% better partitions, on the overall average. In the nonsymmetric test matrices, the direct approaches achieve 34%–38% better decomposition quality than pMeTiS using the graph model, whereas the clique-net approach achieves 21%–24% better decomposition quality. As seen in Table 3.5, the clique-net approach is faster than the direct approaches in the decomposition of the symmetric test matrices. However, PaToH-HCM achieves nearly equal run-time performance as pMeTiS using the clique-net approach in the decomposition of the nonsymmetric test matrices. It is interesting to note that the execution time of the clique-net approach relative to the graph model decreases with increasing number of processors K. This is because of the fact that the percent preprocessing overhead due to the hypergraph-to-graph transformation in the total execution time of pMeTiS using the clique-net approach decreases with increasing K.

As seen in Table 3.5, hMeTiS produces slightly (2%) better partitions at the expense of considerably larger execution time in the decomposition of the symmetric test matrices. However, PaToH-HCM achieves the same decomposition quality as hMeTiS for the nonsymmetric test matrices, whereas PaToH-HCC achieves slightly (2%–3%) better decomposition quality. In the decomposition of the nonsymmetric test matrices, although PaToH-HCC performs slightly better than PaToH-HCM in terms of decomposition quality, it is 13%–14% slower.

In the symmetric test matrices, the use of the proposed hypergraph model instead of the graph model achieves 30% decrease in the communication volume requirement of a single parallel SpMxV computation at the expense of 130% increase in the decomposition time by using PaToH-HCM for hypergraph partitioning. In the nonsymmetric test matrices, the use of the proposed hypergraph

Table 3.1: Properties of test matrices.

matrix name	description	number of rows/cols	number of nonzeros									
			total	avg. per row/col	per column				per row			
					min	max	std	cov	min	max	std	cov
Structurally Symmetric Matrices												
SHERMAN3	[24] 3D finite difference grid	5005	20033	4.00	1	7	2.66	0.67	1	7	2.66	0.67
KEN-11	[16] linear programming	14694	82454	5.61	2	243	14.54	2.59	2	243	14.54	2.59
NL	[18] linear programming	7039	105089	14.93	1	361	28.48	1.91	1	361	28.48	1.91
KEN-13	[16] linear programming	28632	161804	5.65	2	339	16.84	2.98	2	339	16.84	2.98
CQ9	[18] linear programming	9278	221590	23.88	1	702	54.46	2.28	1	702	54.46	2.28
CO9	[18] linear programming	10789	249205	23.10	1	707	52.17	2.26	1	707	52.17	2.26
CRE-D	[16] linear programming	8926	372266	41.71	1	845	76.46	1.83	1	845	76.46	1.83
CRE-B	[16] linear programming	9648	398806	41.34	1	904	74.69	1.81	1	904	74.69	1.81
FINAN512	[23] stochastic programming	74752	615774	8.24	3	1449	20.00	2.43	3	1449	20.00	2.43
Structurally Nonsymmetric Matrices												
GEMAT11	[24] optimal power flow	4929	38101	7.73	1	28	2.96	0.38	1	29	3.38	0.44
LHR07	[23] light hydrocarbon recovery	7337	163716	22.31	1	64	26.19	1.17	2	37	16.00	0.72
ONETONE2	[23] nonlinear analog circuit	36057	254595	7.06	2	34	5.13	0.73	2	66	6.67	0.94
LHR14	[23] light hydrocarbon recovery	14270	321988	22.56	1	64	26.26	1.16	2	37	15.98	0.71
ONETONE1	[23] nonlinear analog circuit	36057	368055	10.21	2	82	14.32	1.40	2	162	17.85	1.75
LHR17	[23] light hydrocarbon recovery	17576	399500	22.73	1	64	26.32	1.16	2	37	15.96	0.70
LHR34	[23] light hydrocarbon recovery	35152	799064	22.73	1	64	26.32	1.16	2	37	15.96	0.70
BCSSTK32	[24] 3D stiffness matrix	44609	1029655	23.08	1	141	10.10	0.44	1	192	10.45	0.45
BCSSTK30	[24] 3D stiffness matrix	28924	1036208	35.83	1	159	21.99	0.61	1	104	15.27	0.43

models instead of the graph model achieves 34%–35% decrease in the communication volume requirement of a single parallel SpMxV computation at the expense of only 34%–39% increase in the decomposition time by using PaToH-HCM.

Table 3.2: Average communication requirements for rowwise/columnwise decomposition of structurally symmetric test matrices.

name	K	Graph Model				Hypergraph Model: Column-net Model ≡ Row-net Model											
		pMeTiS				hMeTiS				PaToH-HCM				PaToH-HCC			
		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume	
		avg	max	tot	max	avg	max	tot	max	avg	max	tot	max	avg	max	tot	max
SHERMAN3	8	3.6	4.9	0.20	0.033	3.6	5.0	0.17	0.029	3.4	4.9	0.16	0.030	3.3	4.8	0.16	0.030
	16	5.3	8.2	0.31	0.028	5.2	7.8	0.27	0.024	4.5	7.4	0.25	0.024	4.7	7.8	0.25	0.025
	32	6.5	11.0	0.46	0.021	6.7	10.9	0.39	0.018	5.7	10.1	0.37	0.019	5.9	10.5	0.37	0.019
	64	7.5	13.6	0.64	0.016	7.9	13.6	0.55	0.013	7.0	13.1	0.53	0.014	7.0	13.4	0.53	0.014
KEN-11	8	7.0	7.0	0.70	0.116	6.9	7.0	0.47	0.078	6.9	7.0	0.51	0.083	7.0	7.0	0.55	0.094
	16	13.8	15.0	0.92	0.080	12.4	15.0	0.57	0.047	12.8	15.0	0.59	0.046	13.7	15.0	0.66	0.057
	32	26.1	30.5	1.16	0.055	19.8	30.3	0.70	0.032	21.2	31.0	0.73	0.033	22.1	30.5	0.79	0.034
	64	40.9	54.9	1.44	0.038	30.1	58.6	0.90	0.024	32.1	60.4	0.92	0.025	30.1	54.2	0.96	0.025
NL	8	7.0	7.0	1.33	0.192	6.8	7.0	0.72	0.110	6.8	7.0	0.76	0.124	7.0	7.0	0.79	0.135
	16	15.0	15.0	1.71	0.147	13.5	15.0	0.99	0.085	13.2	15.0	1.05	0.097	13.7	15.0	1.14	0.101
	32	28.1	31.0	2.26	0.101	19.5	26.5	1.40	0.060	20.0	27.6	1.52	0.068	20.3	27.5	1.57	0.070
	64	38.2	59.1	3.06	0.073	24.4	39.3	2.08	0.045	26.4	40.5	2.20	0.048	26.0	42.9	2.23	0.050
KEN-13	8	7.0	7.0	0.75	0.120	7.0	7.0	0.47	0.070	7.0	7.0	0.48	0.075	6.9	7.0	0.48	0.076
	16	14.8	15.0	0.94	0.078	13.2	15.0	0.54	0.043	14.0	15.0	0.55	0.041	13.4	15.0	0.55	0.042
	32	29.2	31.0	1.16	0.051	22.7	31.0	0.64	0.029	22.8	31.0	0.63	0.025	21.8	31.0	0.63	0.027
	64	51.0	62.2	1.41	0.034	35.9	62.8	0.80	0.022	35.8	63.0	0.79	0.020	34.7	63.0	0.78	0.019
CQ9	8	7.0	7.0	1.11	0.173	7.0	7.0	0.65	0.104	7.0	7.0	0.71	0.154	6.9	7.0	0.71	0.166
	16	14.9	15.0	1.69	0.172	12.7	15.0	0.88	0.097	12.9	15.0	0.99	0.120	12.7	14.9	0.96	0.112
	32	21.8	30.7	2.42	0.148	18.6	26.6	1.36	0.075	18.0	27.0	1.47	0.086	17.6	26.9	1.40	0.082
	64	32.1	56.4	3.71	0.115	23.7	38.4	2.27	0.061	22.7	41.0	2.34	0.065	22.7	39.5	2.31	0.064
CO9	8	7.0	7.0	0.96	0.156	7.0	7.0	0.67	0.110	7.0	7.0	0.68	0.133	7.0	7.0	0.67	0.139
	16	14.8	15.0	1.51	0.157	12.4	14.9	0.87	0.091	12.7	14.9	0.94	0.110	12.7	14.9	0.92	0.107
	32	19.5	29.7	2.08	0.120	17.6	26.6	1.33	0.079	17.6	26.3	1.37	0.077	18.1	26.7	1.34	0.079
	64	29.9	52.3	3.14	0.093	21.7	37.3	2.13	0.061	21.8	38.8	2.16	0.059	21.9	38.6	2.14	0.062
CRE-D	8	7.0	7.0	1.81	0.292	6.9	7.0	1.39	0.226	6.4	7.0	1.33	0.214	6.2	7.0	1.25	0.208
	16	14.9	15.0	2.81	0.238	13.0	15.0	2.09	0.177	11.8	15.0	2.00	0.176	11.2	15.0	1.89	0.163
	32	28.7	31.0	4.13	0.188	21.3	31.0	2.97	0.136	19.3	31.0	2.89	0.133	18.4	31.0	2.73	0.124
	64	47.9	63.0	6.01	0.142	31.2	61.3	4.16	0.104	29.7	60.8	4.19	0.104	27.9	60.5	3.96	0.098
CRE-B	8	7.0	7.0	1.70	0.267	6.9	7.0	1.40	0.224	6.7	7.0	1.33	0.213	6.6	7.0	1.28	0.212
	16	14.8	15.0	2.62	0.230	13.4	15.0	2.07	0.177	12.2	15.0	2.01	0.175	12.2	15.0	1.95	0.180
	32	28.5	31.0	3.89	0.179	21.5	30.9	2.90	0.138	20.0	31.0	2.88	0.148	19.3	31.0	2.75	0.154
	64	46.6	63.0	5.72	0.136	31.3	61.4	4.07	0.111	30.0	61.7	4.12	0.121	28.3	61.5	3.93	0.125
FINAN512	8	2.9	4.3	0.13	0.047	2.8	4.2	0.11	0.045	3.0	4.6	0.12	0.047	3.4	5.6	0.12	0.047
	16	4.3	7.2	0.20	0.034	3.0	6.7	0.14	0.024	3.3	7.2	0.16	0.025	4.0	9.4	0.17	0.027
	32	6.3	13.6	0.27	0.020	3.4	13.2	0.18	0.015	4.2	13.8	0.21	0.016	4.7	17.3	0.22	0.017
	64	8.8	26.5	0.38	0.013	4.2	25.8	0.28	0.010	5.5	26.4	0.31	0.011	5.9	31.0	0.32	0.012
Averages over K																	
	8	6.2	6.5	0.97	0.155	6.1	6.5	0.67	0.111	6.0	6.5	0.68	0.119	6.0	6.6	0.67	0.123
	16	12.5	13.4	1.41	0.129	11.0	13.3	0.93	0.085	10.8	13.3	0.95	0.091	10.9	13.6	0.94	0.090
	32	21.6	26.6	1.98	0.098	16.8	25.2	1.32	0.065	16.5	25.4	1.34	0.067	16.5	25.8	1.31	0.067
	64	33.6	50.1	2.83	0.073	23.4	44.3	1.92	0.050	23.4	45.1	1.95	0.052	22.7	45.0	1.91	0.052

In the "# of mssgs" column, "avg" and "max" denote the average and maximum number of messages, respectively, handled by a single processor. In the "comm. volume" column, "tot" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled by a single processor. Communication volume values (in terms of the number of words transmitted) are scaled by the number of rows/columns of the respective test matrices.

Table 3.3: Average communication requirement for rowwise decomposition of structurally nonsymmetric test matrices.

name	K	Graph Model				Hypergraph Model: Column-net Model											
		pMeTiS				hMeTiS				PaToH-HCM				PaToH-HCC			
		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume	
		avg	max	tot	max	avg	max	tot	max	avg	max	tot	max	avg	max	tot	max
GEMAT11	8	7.0	7.0	1.33	0.201	7.0	7.0	0.79	0.111	7.0	7.0	0.75	0.109	7.0	7.0	0.73	0.106
	16	15.0	15.0	1.85	0.144	14.8	15.0	1.00	0.071	14.7	15.0	0.96	0.070	14.6	15.0	0.93	0.067
	32	29.8	31.0	2.31	0.092	26.6	30.8	1.18	0.044	25.8	30.6	1.15	0.043	25.1	30.4	1.10	0.042
	64	47.7	58.8	2.71	0.056	34.3	46.7	1.33	0.026	33.5	46.2	1.32	0.026	31.9	44.2	1.27	0.025
LHR07	8	6.8	7.0	1.09	0.179	6.2	7.0	0.64	0.111	6.0	7.0	0.65	0.106	5.8	7.0	0.66	0.116
	16	13.0	15.0	1.52	0.130	10.3	13.9	0.93	0.089	9.7	13.8	0.91	0.081	9.2	13.1	0.90	0.083
	32	20.1	29.1	1.96	0.094	13.9	22.3	1.30	0.081	13.0	21.7	1.24	0.066	12.5	20.5	1.24	0.064
	64	24.4	44.8	2.49	0.079	16.8	33.5	1.84	0.077	15.6	30.0	1.65	0.056	15.9	30.7	1.64	0.059
ONETONE2	8	2.8	4.3	0.08	0.014	2.6	3.8	0.06	0.010	2.4	3.5	0.06	0.011	2.5	3.6	0.06	0.010
	16	4.9	7.5	0.17	0.015	4.9	7.3	0.11	0.010	4.7	6.9	0.12	0.011	4.7	6.8	0.12	0.011
	32	7.0	11.9	0.28	0.014	7.5	13.3	0.20	0.009	8.0	11.9	0.22	0.009	7.1	10.9	0.21	0.009
	64	9.4	18.6	0.39	0.011	10.1	20.1	0.29	0.007	10.7	17.2	0.31	0.008	9.4	15.8	0.31	0.008
LHR14	8	7.0	7.0	0.99	0.157	6.6	7.0	0.61	0.100	6.4	7.0	0.59	0.095	6.2	7.0	0.59	0.097
	16	14.0	15.0	1.33	0.116	11.4	14.6	0.84	0.074	10.3	13.5	0.81	0.071	10.0	13.6	0.82	0.072
	32	22.9	29.4	1.71	0.078	15.5	23.2	1.10	0.056	13.5	20.7	1.05	0.050	13.1	20.9	1.07	0.053
	64	29.9	48.6	2.14	0.054	18.1	31.5	1.44	0.048	15.4	27.5	1.34	0.040	15.6	29.0	1.36	0.041
ONETONE1	8	5.1	6.5	0.42	0.067	3.7	5.0	0.16	0.025	3.5	4.9	0.16	0.026	3.6	4.9	0.16	0.025
	16	8.5	11.8	0.59	0.050	7.9	10.4	0.29	0.023	7.6	9.8	0.30	0.026	7.8	10.1	0.29	0.024
	32	13.6	19.1	0.78	0.035	14.2	19.7	0.42	0.017	13.8	19.1	0.45	0.020	14.2	18.9	0.42	0.019
	64	18.7	28.9	0.97	0.025	22.0	33.0	0.57	0.013	19.3	29.2	0.61	0.016	19.8	29.7	0.56	0.015
LHR17	8	7.0	7.0	0.94	0.143	6.9	7.0	0.62	0.094	6.7	7.0	0.57	0.090	6.5	7.0	0.60	0.095
	16	14.3	15.0	1.28	0.110	12.4	14.8	0.82	0.068	11.0	13.8	0.77	0.066	10.8	13.7	0.80	0.068
	32	23.5	29.6	1.62	0.074	17.1	23.8	1.07	0.052	14.4	21.0	1.00	0.047	14.1	21.5	1.03	0.047
	64	30.3	46.9	2.04	0.048	19.6	33.0	1.38	0.041	16.4	29.4	1.29	0.036	16.0	30.3	1.30	0.036
LHR34	8	3.5	4.8	0.61	0.088	3.6	5.3	0.42	0.063	3.5	5.0	0.38	0.056	3.4	4.5	0.40	0.061
	16	7.3	9.5	0.95	0.075	7.3	10.1	0.62	0.049	7.0	9.7	0.57	0.046	6.8	8.8	0.60	0.050
	32	14.5	17.5	1.28	0.055	12.6	16.8	0.84	0.037	11.1	15.3	0.77	0.034	10.9	14.6	0.80	0.035
	64	23.7	30.6	1.63	0.038	17.2	24.9	1.08	0.027	14.6	22.7	1.00	0.025	14.3	22.5	1.03	0.025
BCSSTK32	8	3.5	5.4	0.07	0.015	3.7	5.7	0.05	0.012	3.5	5.4	0.05	0.013	3.6	5.5	0.05	0.012
	16	4.4	7.6	0.12	0.013	4.2	8.3	0.09	0.011	4.0	7.3	0.09	0.011	4.0	7.3	0.09	0.011
	32	5.1	9.4	0.20	0.011	4.7	10.6	0.14	0.008	4.7	9.6	0.15	0.009	4.6	9.7	0.14	0.008
	64	5.7	11.3	0.30	0.008	4.8	11.6	0.22	0.006	4.9	11.0	0.24	0.007	4.7	10.8	0.22	0.006
BCSSTK30	8	2.3	3.9	0.10	0.018	2.3	3.6	0.09	0.018	2.2	3.4	0.09	0.017	2.2	3.4	0.08	0.017
	16	3.7	6.3	0.21	0.022	3.3	5.4	0.18	0.018	3.3	5.6	0.18	0.018	3.3	5.6	0.16	0.017
	32	4.9	8.7	0.36	0.019	4.4	7.9	0.29	0.015	4.6	8.0	0.31	0.016	4.4	7.8	0.28	0.014
	64	5.8	11.3	0.57	0.016	5.3	10.6	0.45	0.013	5.6	10.3	0.48	0.013	5.3	10.0	0.45	0.012
Averages over K																	
	8	5.0	5.9	0.63	0.098	4.7	5.7	0.38	0.060	4.6	5.6	0.37	0.058	4.5	5.5	0.37	0.060
	16	9.5	11.4	0.89	0.075	8.5	11.1	0.54	0.046	8.0	10.6	0.53	0.045	7.9	10.4	0.52	0.045
	32	15.7	20.6	1.17	0.052	12.9	18.7	0.73	0.036	12.1	17.5	0.70	0.033	11.8	17.3	0.70	0.032
	64	21.7	33.3	1.47	0.037	16.5	27.2	0.96	0.029	15.1	24.8	0.92	0.025	14.8	24.8	0.90	0.025

In the "# of mssgs" column, "avg" and "max" denote the average and maximum number of messages, respectively, handled by a single processor. In the "comm. volume" column, "tot" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled by a single processor. Communication volume values (in terms of the number of words transmitted) are scaled by the number of rows/columns of the respective test matrices.

Table 3.4: Average communication requirements for columnwise decomposition of structurally nonsymmetric test matrices.

name	K	Graph Model				Hypergraph Model: Row-net Model											
		pMeTiS				hMeTiS				PaToH-HCM				PaToH-HCC			
		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume		# of mssgs per proc.		comm. volume	
		avg	max	tot	max	avg	max	tot	max	avg	max	tot	max	avg	max	tot	max
GEMAT11	8	7.0	7.0	1.44	0.213	7.0	7.0	0.75	0.108	7.0	7.0	0.76	0.110	7.0	7.0	0.72	0.108
	16	15.0	15.0	1.98	0.145	14.7	15.0	0.95	0.071	14.7	15.0	0.97	0.072	14.6	15.0	0.93	0.069
	32	29.9	31.0	2.46	0.091	25.6	30.0	1.13	0.043	25.9	30.3	1.15	0.043	25.0	29.9	1.10	0.042
	64	47.9	58.5	2.85	0.056	32.7	43.9	1.28	0.026	33.6	45.3	1.33	0.026	31.6	43.8	1.27	0.025
LHR07	8	6.9	7.0	1.10	0.188	6.5	7.0	0.75	0.123	6.4	7.0	0.67	0.107	6.4	7.0	0.66	0.105
	16	12.5	15.0	1.54	0.141	11.1	15.0	1.10	0.094	10.6	15.0	0.96	0.081	10.8	15.0	0.95	0.081
	32	19.3	30.3	2.05	0.112	16.4	28.7	1.52	0.068	15.1	29.5	1.32	0.059	15.6	29.0	1.31	0.059
	64	23.5	56.7	2.60	0.088	22.0	39.2	2.03	0.050	19.7	40.5	1.76	0.042	19.8	41.2	1.74	0.042
ONETONE2	8	2.6	3.8	0.09	0.017	2.4	3.2	0.07	0.012	2.2	3.1	0.08	0.013	3.1	4.5	0.08	0.013
	16	4.8	7.4	0.20	0.019	4.7	6.6	0.13	0.012	4.6	6.2	0.16	0.014	5.4	8.7	0.15	0.014
	32	7.5	12.7	0.34	0.016	7.6	11.2	0.24	0.010	7.6	11.1	0.27	0.011	8.3	14.8	0.25	0.011
	64	10.2	21.4	0.46	0.013	9.6	15.8	0.33	0.008	10.5	16.4	0.35	0.008	10.4	23.5	0.34	0.009
LHR14	8	7.0	7.0	1.05	0.168	6.6	7.0	0.67	0.109	6.6	7.0	0.61	0.096	6.7	7.0	0.61	0.096
	16	13.9	15.0	1.43	0.123	11.4	14.7	0.95	0.077	11.6	15.0	0.85	0.069	11.7	15.0	0.84	0.069
	32	22.9	30.4	1.85	0.087	16.8	27.9	1.26	0.054	16.4	29.6	1.11	0.047	16.5	30.5	1.11	0.049
	64	29.3	55.3	2.32	0.069	21.3	45.7	1.65	0.038	19.8	54.2	1.45	0.035	20.3	56.2	1.44	0.036
ONETONE1	8	5.1	6.5	0.44	0.067	3.7	5.0	0.19	0.031	3.5	4.7	0.21	0.033	3.5	4.9	0.20	0.034
	16	8.7	11.6	0.62	0.051	7.8	10.2	0.34	0.026	7.6	9.6	0.38	0.032	7.8	10.1	0.36	0.029
	32	14.4	20.0	0.81	0.035	13.3	18.6	0.49	0.021	13.4	18.6	0.54	0.026	14.0	19.1	0.51	0.024
	64	19.9	30.2	1.08	0.024	19.9	31.5	0.65	0.017	19.6	30.5	0.72	0.018	19.3	30.4	0.69	0.019
LHR17	8	7.0	7.0	1.02	0.164	6.8	7.0	0.66	0.100	6.8	7.0	0.59	0.087	6.9	7.0	0.58	0.087
	16	14.4	15.0	1.40	0.117	12.2	15.0	0.91	0.074	12.3	15.0	0.81	0.064	12.3	15.0	0.80	0.063
	32	24.2	30.6	1.78	0.080	18.0	30.0	1.22	0.052	17.1	30.6	1.06	0.044	17.2	30.8	1.05	0.044
	64	31.4	53.3	2.21	0.062	22.9	51.9	1.58	0.037	20.7	55.0	1.37	0.031	20.8	55.8	1.36	0.032
LHR34	8	3.4	4.5	0.67	0.103	3.4	4.1	0.43	0.065	3.4	4.1	0.39	0.056	3.4	4.1	0.39	0.055
	16	7.3	8.6	1.02	0.086	7.1	8.4	0.66	0.053	7.2	8.3	0.59	0.046	7.1	8.3	0.59	0.046
	32	14.7	16.8	1.40	0.061	12.4	15.9	0.92	0.040	12.4	15.6	0.81	0.033	12.5	15.7	0.80	0.033
	64	24.2	31.4	1.78	0.043	18.2	30.3	1.22	0.028	17.3	30.8	1.06	0.023	17.3	31.0	1.06	0.023
BCSSTK32	8	3.6	5.3	0.07	0.016	3.1	4.6	0.05	0.013	3.9	5.8	0.06	0.014	3.4	5.2	0.05	0.012
	16	4.3	7.3	0.12	0.014	3.9	7.0	0.08	0.010	4.4	7.9	0.10	0.012	4.1	7.7	0.08	0.011
	32	5.1	9.5	0.19	0.011	4.4	8.9	0.14	0.008	4.7	9.9	0.15	0.009	4.6	9.4	0.14	0.009
	64	5.5	11.6	0.29	0.009	4.5	10.1	0.21	0.007	4.9	11.4	0.23	0.008	4.7	11.2	0.21	0.007
BCSSTK30	8	2.5	4.0	0.08	0.017	2.8	4.6	0.08	0.017	2.2	3.4	0.07	0.014	2.4	4.2	0.06	0.013
	16	3.6	6.2	0.18	0.018	3.4	6.0	0.14	0.015	3.0	5.0	0.14	0.016	3.1	5.2	0.13	0.014
	32	4.7	8.2	0.31	0.015	4.0	8.0	0.22	0.012	4.0	6.9	0.24	0.013	3.9	7.1	0.21	0.012
	64	5.7	10.0	0.50	0.013	4.6	9.0	0.34	0.010	4.5	8.4	0.37	0.010	4.5	9.3	0.34	0.010
Averages over K																	
	8	5.0	5.8	0.66	0.106	4.7	5.5	0.40	0.064	4.7	5.5	0.38	0.059	4.8	5.7	0.37	0.058
	16	9.4	11.2	0.94	0.079	8.5	10.9	0.59	0.048	8.4	10.8	0.55	0.045	8.6	11.1	0.54	0.044
	32	15.8	21.1	1.24	0.057	13.2	19.9	0.79	0.034	13.0	20.2	0.74	0.032	13.1	20.7	0.72	0.031
	64	22.0	36.5	1.57	0.042	17.3	30.8	1.03	0.024	16.7	32.5	0.96	0.022	16.5	33.6	0.94	0.023

In the "# of mssgs" column, "avg" and "max" denote the average and maximum number of messages, respectively, handled by a single processor. In the "comm. volume" column, "tot" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled by a single processor. Communication volume values (in terms of the number of words transmitted) are scaled by the number of rows/columns of the respective test matrices.

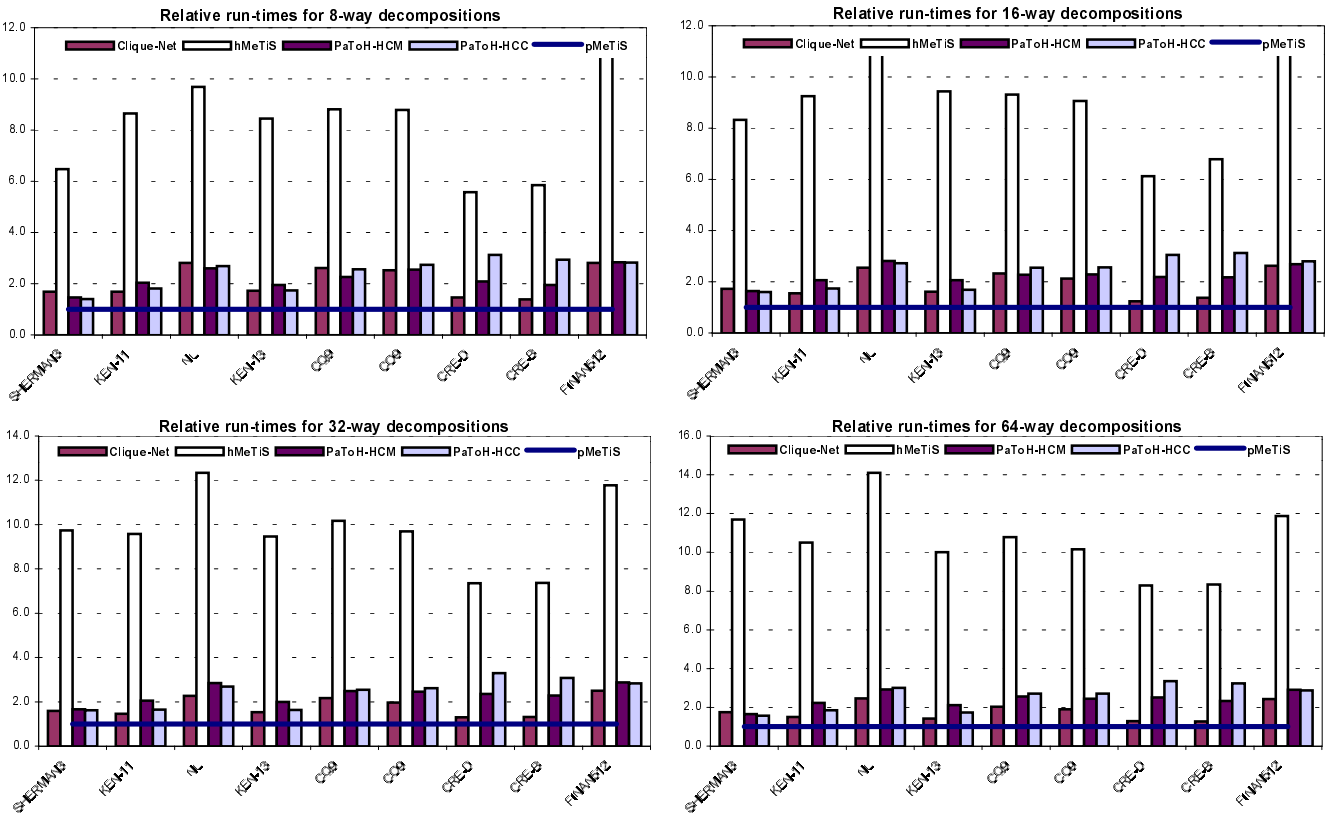

Figure 3.5: Relative run-time performance of the proposed column-net/row-net hypergraph model (Clique-net, hMeTiS, PaToH-HCM and PaToH-HCC) to the graph model (pMeTiS) in rowwise/columnwise decomposition of symmetric test matrices. Bars above 1.0 indicate that the hypergraph model leads to slower decomposition time than the graph model.

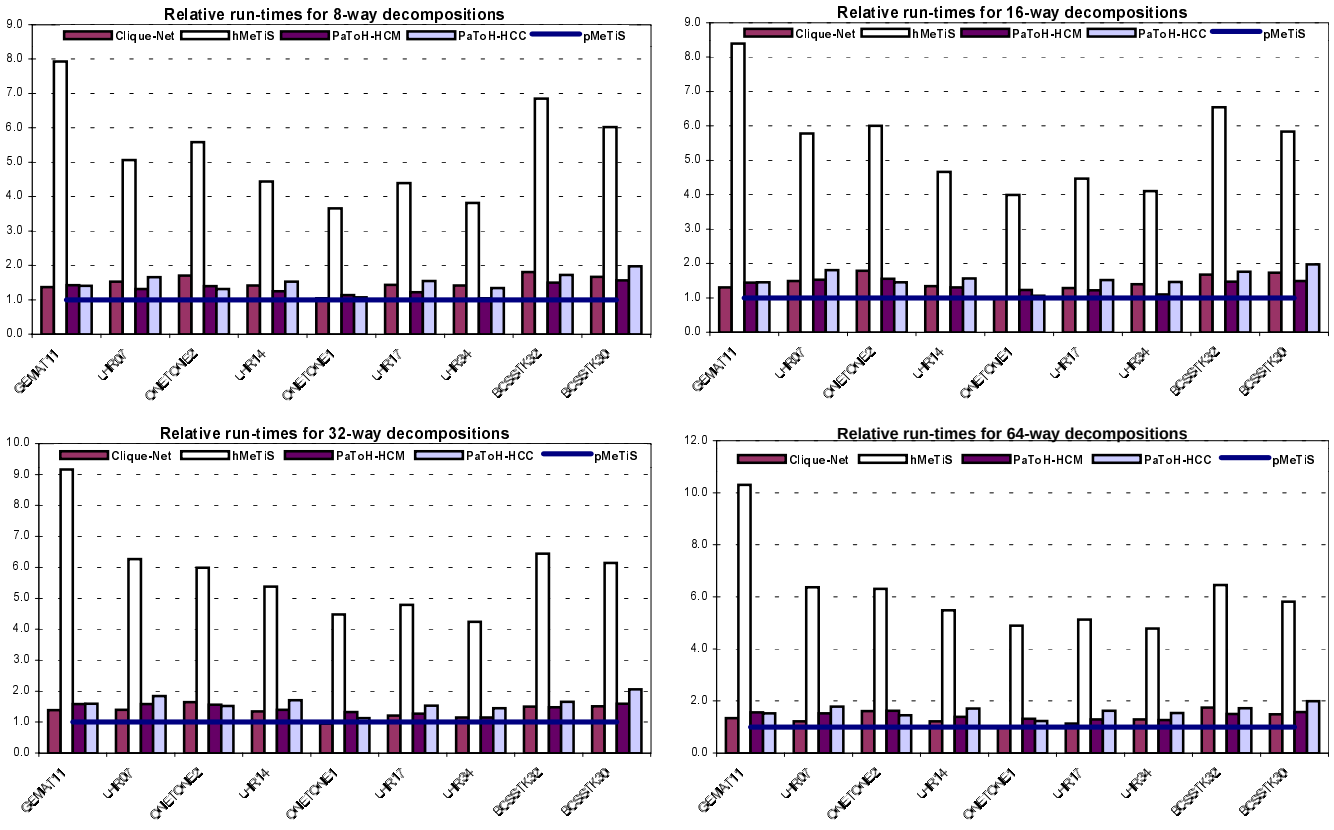

Figure 3.6: Relative run-time performance of the proposed column-net hypergraph model (Clique-net, hMeTiS, PaToH-HCM and PaToH-HCC) to the graph model (pMeTiS) in rowwise decomposition of symmetric test matrices. Bars above 1.0 indicate that the hypergraph model leads to slower decomposition time than the graph model.

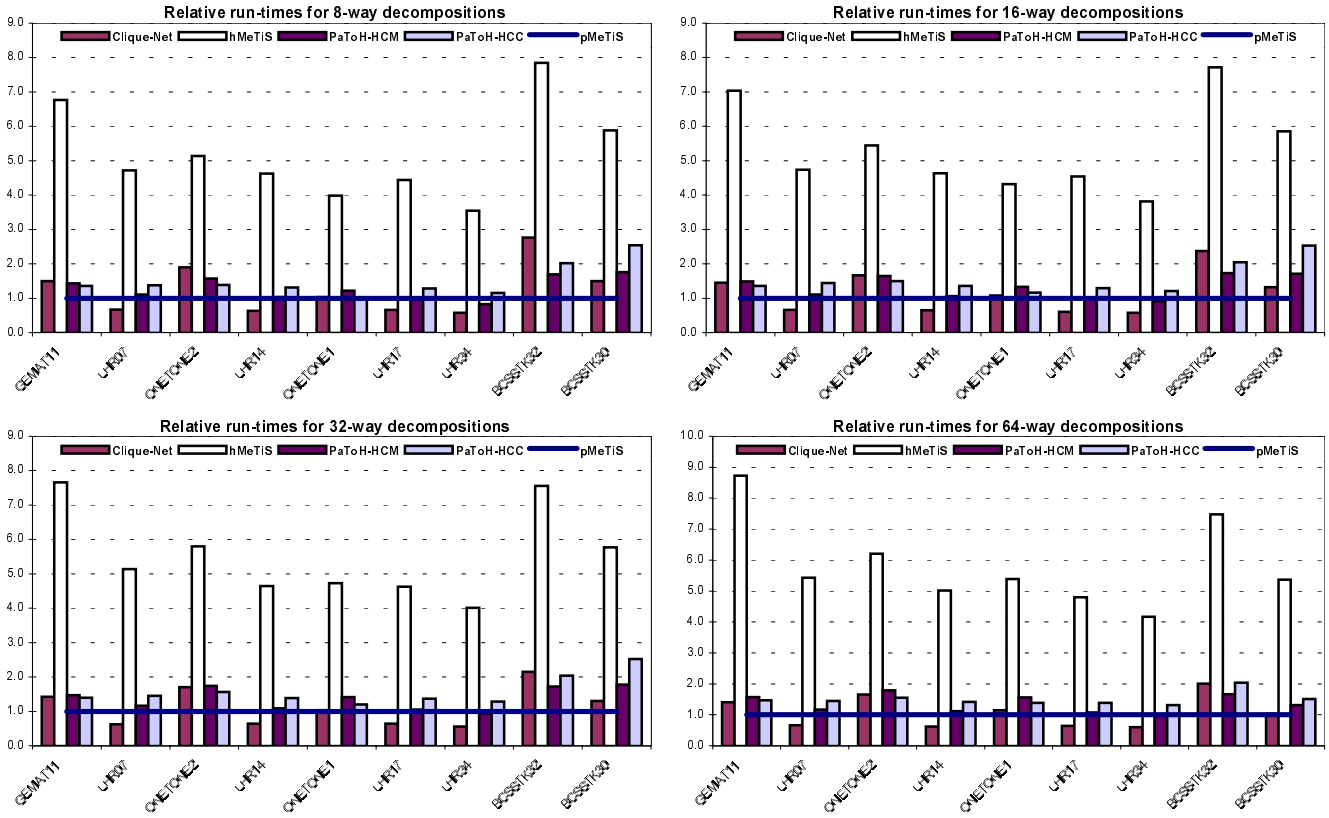

Figure 3.7: Relative run-time performance of the proposed row-net hypergraph model (Clique-net, hMeTiS, PaToH-HCM and PaToH-HCC) to the graph model (pMeTiS) in columnwise decomposition of symmetric test matrices. Bars above 1.0 indicate that the hypergraph model leads to slower decomposition time than the graph model.

Table 3.5: Overall performance averages of the proposed hypergraph models normalized with respect to those of the graph models using pMeTiS.

K	pMeTiS (clique-net model)				hMeTiS				PaToH-HCM				PaToH-HCC			
	Tot. Comm. Volume			Time	Tot. Comm. Volume			Time	Tot. Comm. Volume			Time	Tot. Comm. Volume			Time
	best	worst	avg		best	worst	avg		best	worst	avg		best	worst	avg	
Symmetric Matrices: Column-net Model ≡ Row-net Model																
8	0.86	0.84	0.85	2.08	0.73	0.70	0.71	8.13	0.73	0.73	0.73	2.19	0.73	0.73	0.73	2.42
16	0.86	0.84	0.83	1.90	0.70	0.66	0.66	8.95	0.70	0.69	0.68	2.25	0.71	0.69	0.69	2.43
32	0.85	0.84	0.84	1.79	0.68	0.65	0.66	9.72	0.69	0.68	0.68	2.33	0.69	0.68	0.68	2.44
64	0.85	0.84	0.84	1.78	0.71	0.68	0.69	10.64	0.72	0.69	0.70	2.41	0.72	0.69	0.70	2.56
avg	0.86	0.84	0.84	1.89	0.70	0.67	0.68	9.36	0.71	0.70	0.70	2.30	0.71	0.70	0.70	2.46
Nonsymmetric Matrices: Column-net Model																
8	0.78	0.78	0.78	1.48	0.68	0.63	0.64	5.31	0.67	0.64	0.64	1.32	0.66	0.62	0.63	1.50
16	0.80	0.78	0.78	1.44	0.66	0.63	0.64	5.53	0.67	0.64	0.65	1.37	0.65	0.62	0.63	1.56
32	0.79	0.78	0.78	1.34	0.66	0.64	0.66	5.88	0.67	0.65	0.66	1.44	0.65	0.63	0.64	1.61
64	0.80	0.79	0.79	1.34	0.69	0.68	0.68	6.17	0.69	0.68	0.68	1.45	0.67	0.66	0.66	1.62
avg	0.79	0.78	0.79	1.40	0.67	0.64	0.66	5.72	0.67	0.65	0.66	1.39	0.66	0.63	0.64	1.57
Nonsymmetric Matrices: Row-net Model																
8	0.75	0.74	0.76	1.25	0.64	0.62	0.63	5.22	0.64	0.63	0.63	1.29	0.62	0.60	0.61	1.50
16	0.75	0.74	0.75	1.15	0.65	0.63	0.64	5.34	0.65	0.63	0.65	1.33	0.62	0.61	0.62	1.54
32	0.75	0.75	0.75	1.12	0.67	0.65	0.66	5.55	0.66	0.64	0.66	1.38	0.63	0.62	0.63	1.58
64	0.76	0.77	0.76	1.09	0.67	0.67	0.67	5.84	0.66	0.65	0.66	1.36	0.64	0.63	0.63	1.50
avg	0.75	0.75	0.76	1.15	0.66	0.64	0.65	5.49	0.65	0.64	0.65	1.34	0.63	0.61	0.62	1.53

In total communication volume, a ratio smaller than 1.00 indicates that the hypergraph model produces better decompositions than the graph model. In execution time, a ratio greater than 1.00 indicates that the hypergraph model leads to slower decomposition time than the graph model.

Chapter 4

Hypergraph Models for 2D Decomposition

The atomic task definition in the 1D decomposition ensures that either row stripes or column stripes are distributed among the processors. That is computations for a row and column are considered as indivisible tasks in rowwise and columnwise decomposition, respectively. This atomic task definition can be unnecessarily restricted. Consider the sparse matrices which have some dense rows/columns. Load balancing problem becomes very hard for this kind of matrices. It is conjectured that columnwise decomposition can be more appropriate for the matrices with dense rows, and rowwise decomposition can be appropriate for the ones with dense columns. However, this precaution can be valuable for only nonsymmetric matrices. Furthermore, columnwise (rowwise) decomposition of matrices with dense rows (columns) is likely to induce high volume of communication during the post (pre) communication phase. The 2D decomposition approach is expected to yield better decomposition in terms of both load balancing and communication requirements since it has more degree of freedom.

Unfortunately, in the literature there is not too much work on 2D decomposition of matrices, and existing heuristics address only the load balancing problem [62, 56, 55, 37]. The matrix-vector multiplication algorithm proposed by Hendrickson et. al. [37] is based on 2D block checkerboard partitioning and minimizes the communication requirement implicitly. Lewis and Geijn [56] and

Lewis et.al. [55] proposed different parallel SpMxV computation approaches one of which eliminates the transpose operation required in method proposed by Hendrickson et. al. [37].

There is no work on 2D decomposition which directly aims at minimizing communication volume while maintaining the load balance. In this section, three different hypergraph models will be introduced for 2D decomposition of sparse matrices. Here, we propose a fine-grain hypergraph model which considers each multiply operation in SpMxV as atomic tasks during the decomposition. Two new coarse-grain hypergraph models are proposed for reducing the decomposition overhead. Another objective in the coarse-grain hypergraph models is an implicit effort towards reducing the amount of communication. The first hypergraph model produces jagged-like 2D decompositions of the sparse matrices. The second coarse-grain hypergraph model is specifically designed for checkerboard partitioning which is commonly used in the literature by the matrix-vector multiplication algorithms [62, 56, 55, 37]. Experimental results presented in Section 4.4 show that the fine-grain hypergraph model for 2D decomposition produces superior results over 1D decomposition results produced by both graph and hypergraph models, in terms of total communication volume. The coarse-grain models also produce better decompositions then the graph model in terms of total communication volume. In terms of number of messages, checkerboard decomposition displays its strength over all models.

As mentioned earlier, parallel SpMxV computations based on 2D decomposition schemes, necessitates both pre and post communication. That is, the entries in $\mathbf{x}$ vector need to be communicated just before the local SpMxV computations, and the result of partial $\mathbf{y}$ vector need to be communicated after local SpMxV computations. Here and after, we will use the term *expand* to denote the personalized communication of the entries in $\mathbf{x}$, and *fold* to denote the personalized communication of entries in $\mathbf{y}$.

4.1 A Fine-grain Hypergraph Model

In this model, an $M \times M$ matrix $\mathbf{A}$ with Z nonzero elements is represented as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ with $|\mathcal{V}| = Z$ vertices and $|\mathcal{N}| = 2 \times M$ nets for 2D decomposition which uses both pre and post communication. There exists one vertex $v_{ij} \in \mathcal{V}$ corresponding to each nonzero a_{ij} in matrix $\mathbf{A}$. For each row and for each column there exists a net in $\mathcal{N}$. For simplicity in the presentation let $\mathcal{N} = \mathcal{N}_{\mathcal{R}} \cup \mathcal{N}_{\mathcal{C}}$ such that $\mathcal{N}_{\mathcal{R}} = \{m_1, m_2, \ldots, m_M\}$ represents the set of nets corresponding to the rows and $\mathcal{N}_{\mathcal{C}} = \{n_1, n_2, \ldots, n_M\}$ represents the set of nets corresponding to the columns of the matrix $\mathbf{A}$. Net $m_j \subseteq \mathcal{V}$ contains the vertices corresponding to the nonzeros in row j, and net $n_j \subseteq \mathcal{V}$ contains the vertices corresponding to the nonzeros in column j. That is, $v_{ij} \in m_i$ and $v_{ij} \in n_j$ if and only if $a_{ij} \neq 0$. Note that each vertex v_{ij} is connected exactly two nets. Each vertex $v_{ij} \in \mathcal{V}$ corresponds to the atomic task of computing the scalar multiplication operation $y_i^j = a_{ij}x_j$. Hence, each vertex $v_{ij} \in \mathcal{V}$ has unit computational weight $w_{ij} = 1$. The nets in $\mathcal{N}_{\mathcal{C}}$ represent the dependency relations of the atomic tasks to the $\mathbf{x}$-vector components, that is, they model the expand operation in the pre communication phase. The nets in $\mathcal{N}_{\mathcal{R}}$ represent the dependency relations of the atomic tasks on the $\mathbf{y}$-vector components, in other words, they model the fold operation in the post communication phase. Hence, each column-net n_j denotes the set of atomic tasks (vertices) that need x_j during pre communication, and each row-net m_i denotes the set of atomic task results needed to accumulate y_i during the post communication. Figure 4.1 illustrates the dependency relation view of 2D fine-grain model. As seen in this figure, column-net $n_j = \{v_{ij}, v_{jj}, v_{lj}\}$ of size 3 represents the dependency of atomic tasks v_{ij}, v_{jj}, v_{lj} to x_j because of the 3 multiplication operations $y_i^j = a_{ij} \cdot x_j$, $y_j^j = a_{jj} \cdot x_j$ and $y_l^j = a_{lj} \cdot x_j$. In this figure, row-net $m_i = \{v_{ih}, v_{ii}, v_{ik}, v_{ij}\}$ of size 4 represents the dependency of accumulating $y_i = y_i^h + y_i^i + y_i^k + y_i^j$ to the 4 partial y_i results $y_i^h = a_{ih} \cdot x_h$, $y_i^i = a_{ii} \cdot x_i$, $y_i^k = a_{ik} \cdot x_k$ and $y_i^j = a_{ij} \cdot x_j$. Figure 4.3 displays the 2D fine-grain hypergraph representation of the sample 8×8 nonsymmetric matrix with 21 nonzero elements displayed in Figure 4.2. In Figure 4.3 pins of the row net $m_1 = \{v_{1,1}, v_{1,2}, v_{1,6}\}$ corresponding to row 1, represent the nonzeros $a_{1,1}$, $a_{1,2}$, and $a_{1,6}$ in that row. Net m_1 also represents the dependency of accumulating the $y_1 = y_1^1 + y_1^2 + y_1^6$ on the partial y_1 results $y_1^1 = a_{1,1}x_1$, $y_1^2 = a_{1,2}x_2$, and

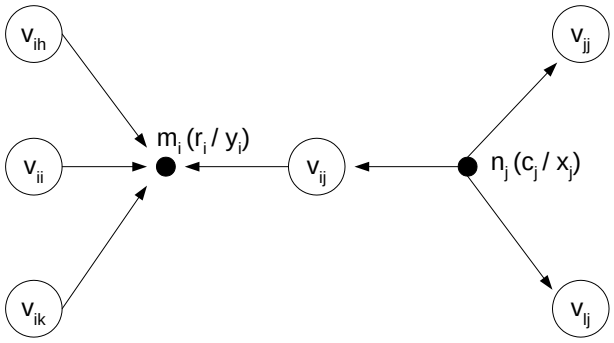

Figure 4.1: Dependency relation of 2D fine-grain hypergraph mode

$y_1^6 = a_{1,6}x_6$. Similarly, pins of the column net $n_7 = \{v_{4,7}, v_{5,7}, v_{7,7}\}$ corresponding to column 7, represents the nonzeros $a_{4,7}$, $a_{5,7}$, and $a_{7,7}$ in that column. Net n_7 is also represents the dependency of atomic tasks $v_{4,7}$, $v_{5,7}$ and $v_{7,7}$ to x_7 because of the computation $y_4^7 = a_{4,7}x_7$, $y_5^7 = a_{5,7}x_7$ and $y_7^7 = a_{7,7}x_7$.

By assigning unit costs to the nets (i.e. $c_j = 1$ for each net $n_j \in \mathcal{N}$), the proposed fine-grain hypergraph model reduces the decomposition problem to the K-way hypergraph partitioning problem according to the cutsize definition given in (2.4.b) for 2D schemes which requires both the pre and post communication. Nets corresponding to rows of matrix (i.e. nets in $\mathcal{N}_\mathcal{R}$) model the communication volume requirement of folds, and nets corresponding the columns of matrix (i.e. nets in $\mathcal{N}_\mathcal{C}$) model the communication volume requirement of expands. Consistency of the proposed hypergraph models for accurate representation of communication volume requirement while maintaining the symmetric partitioning depends on the condition that "$v_{ii} \in m_i$ and $v_{ii} \in n_i$ for each row-net m_i and column-net n_i". We first assume that this condition holds in the discussion throughout the following paragraphs and then discuss the appropriateness of the assumption in the last paragraph of this section.

Consider a partition Π of $\mathcal{H}$ in the fine-grain hypergraph model for 2D decomposition of a matrix $\mathbf{A}$. Without loss of generality, we assume that part $\mathcal{P}_k$ is assigned to processor P_k for $k = 1, 2, \ldots, K$. Recall that, Π is defined as a partition on the vertex set of $\mathcal{H}$, hence it does not induce any part assignment for the nets. Since column and row nets of $\mathcal{H}$ denotes the expand and fold operations on $\mathbf{x}$ and $\mathbf{y}$ vectors, we need to decode Π as inducing a partition on nets to formulate communication volume requirements. Let $\Lambda[n_j]$ and $\Lambda[m_j]$ denote the

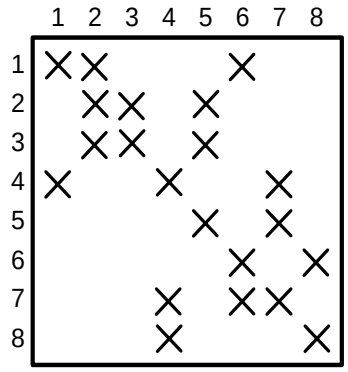

Figure 4.2: A 8×8 nonsymmetric matrix $\mathbf{A}$

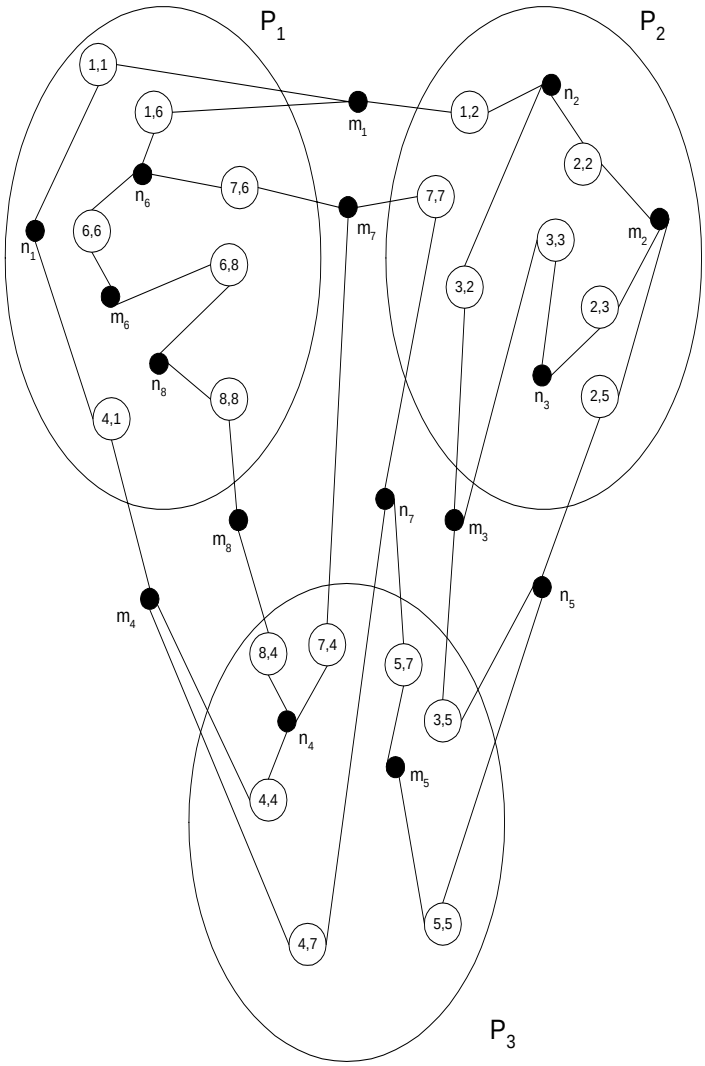

Figure 4.3: 2D fine-grain hypergraph representation $\mathcal{H}$ of the matrix $\mathbf{A}$ displayed in Figure 4.2 and 2-way partitioning Π of $\mathcal{H}$.

connectivity sets of column-net n_j and row-net m_j in Π, and $part[v_{jj}]$ denotes the part (hence processor) assignment for vertex v_{jj}.

Consider an internal column-net n_j of part $\mathcal{P}_k$ (i.e. $\Lambda[n_j] = \{\mathcal{P}_k\}$). As all pins of net n_j lie in $\mathcal{P}_k$, all nonzeros in column j (including a_{jj} by the consistency condition) which need x_j for their multiplication are already assigned to processor P_k. Hence, internal column-net n_j of $\mathcal{P}_k$, which does not contribute to the cutsize (2.4.b) of partition Π, does not necessitate any expand operation if x_j is assigned to processor P_k. Similarly, consider an internal row-net m_j of part $\mathcal{P}_k$. As all pins of row-net m_j lie in $\mathcal{P}_k$, all nonzeros in row j which will contribute in the accumulation of y_j are already assigned to processor P_k. Hence, internal row-net m_j of $\mathcal{P}_k$, which does not contribute to the cutsize (2.4.b) of partition Π, does not necessitate any fold operation if y_j is assigned to processor P_k.

Consider an external column-net n_j (i.e., $\lambda[n_j] > 1$). As all pins of net n_j lie in the parts in its connectivity set $\Lambda[n_j]$, all nonzeros (including a_{jj} by the consistency condition) which need x_j for multiplication are assigned to the parts (processors) in $\Lambda[n_j]$. Hence, contribution $\lambda[n_j]-1$ of external net n_j to the cutsize according to (2.4.b) accurately models the amount of communication volume to incur during the expand of x_j if x_j is assigned to any processor in $\Lambda[n_j]$. Let $map[n_j] \in \Lambda[n_j]$ denote the part and hence processor assignment for x_j corresponding to cut net n_j. Cut net n_j indicates that processor $map[n_j]$ should send its local x_j to those processors in connectivity set $\Lambda[n_j]$ except itself (i.e., to processors in the set $\Lambda[n_j]-\{map[n_j]\}$). Hence, processor $map[n_j]$ should send its local x_j to $|\Lambda[n_j]|-1 = \lambda[n_j]-1$ distinct processors. Similarly, consider an external row-net m_j. As all pins of net m_j lie in the parts in its connectivity set $\Lambda[m_j]$, all nonzeros which will contribute in the accumulation of y_j are already assigned to the parts (processors) in $\Lambda[m_j]$. Cut net m_j indicates that the processors in the connectivity set $\Lambda[m_j]$ except owner of m_j (i.e., processors in the set $\Lambda[m_j]-\{map[m_j]\}$) should send their partial $\mathbf{y}_j$ results to the processor $map[m_j]$. Hence, contribution $\lambda[m_j]-1$ of external row-net m_j to the cutsize according to (2.4.b) accurately models the amount of communication volume to incur during the fold of y_j if y_j is assigned to any processor in $\Lambda[m_j]$.

The connectivity sets $\Lambda[n_j]$ and $\Lambda[m_j]$ of column-net n_j and row-net m_j must

have at least one common part, since they share exactly one common pin, which is a_{jj} by the consistency condition, i.e., $\{part[v_{jj}]\} \subseteq (\Lambda[n_j] \cap \Lambda[m_j])$. There are four distinct cases to consider:

Case 1 Both row-net m_j and column-net n_j are internal to part $part[v_{jj}]$ (note that they cannot be internal to different parts, since both of them contains v_{jj}),

Case 2 Both row-net m_j and column-net n_j are external (cut) nets connected to part $part[v_{jj}]$,

Case 3 Row-net m_j is internal to part $part[v_{jj}]$, and column-net n_j is external net connected to part $part[v_{jj}]$,

Case 4 Column-net n_j is internal to part $part[v_{jj}]$, and row-net m_j is external net connected to part $part[v_{jj}]$

For "Case 1", using the discussion in the previous paragraph, we can safely assign internal nets m_i and n_i to part $part[v_{jj}]$. We know that external nets exactly model the communication requirement if their corresponding variable is also assigned to a part in connectivity set. Hence, for "Case 2", we can again safely assign external nets to part $part[v_{jj}]$, since it is already in the connectivity sets of both external nets. In cases 3 and 4 again since the part, which one of them is internal to, $(part[v_{jj}])$ is already in the connectivity set of the other one, we can also assign both nets to $part[v_{jj}]$.

In essence, in the fine-grain hypergraph model, any partition Π of $\mathcal{H}$ with $part[v_{ii}] = \mathcal{P}_k$ can be safely decoded as assigning row-net m_i (hence y_i) and column-net n_i (hence x_i) to part $\mathcal{P}_k$, i.e., $map[n_i] = map[m_i] = part[v_{ii}]$. With this assignment, both symmetric partitioning (in other words conformal partitioning) on $\mathbf{x}$ and $\mathbf{y}$ vectors is maintained and also total communication volume is exactly modeled. Thus, in the fine-grain model, minimizing the cutsize according to (2.4.b) corresponds to minimizing the actual volume of interprocessor communication during the pre and post communication phases.

Figure 4.3 displays a 3-way partition of the fine-grain hypergraph. The cost of this partition is 8. There are 6 cut nets with connectivity 2, hence their

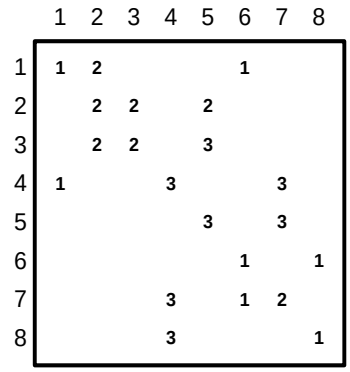

Figure 4.4: Decomposition result of the sample given in Figure 4.3

total contribution to the cost is $6 \times (2-1) = 6$. The connectivity set $\Lambda[m_7]$ of cut net m_7 is $\Lambda[m_7] = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$. Hence its contribution to the cost is $\lambda[m_7]-1 = 3-1 = 2$. Figure 4.4 displays the 3-way partitioning result obtained in Figure 4.3 in matrix view. Here we put the part number of each nonzero as its value. In this figure you can identify the row cutnets m_1, m_3, m_4, m_7 and m_8 as the rows containing different numbers. With this partition, processors P_3 and P_1 will send their partial y_7 results $y_7^4 = a_{7,4} \cdot x_4$ and $y_7^6 = a_{7,6} \cdot x_6$ to processor P_2, which already contains $a_{7,7}$, during the fold operation of y_7. Thus contribution $\lambda[m_7]-1 = 2$ of row-net m_7 to the cost exactly models volume of communication required in the fold of y_7.

Nonzero diagonal entries automatically satisfy the condition "$v_{ii} \in m_i$ and $v_{ii} \in n_i$ for each row-net m_i and column-net n_i" thus enabling both accurate representation of communication requirement and symmetric partitioning of $\mathbf{x}$ and $\mathbf{y}$ vectors. A nonzero diagonal entry a_{jj} already implies that both column-net n_j and row-net m_j contains vertex v_{jj} as their pin. If however some diagonal entries of the given matrix are zeros then the consistency of the proposed model is easily maintained by simply adding dummy vertex v_{jj} for each $a_{jj} = 0$ with $w_{jj} = 0$ to the vertex set $\mathcal{V}$ of $\mathcal{H}$. Vertex v_jj is also added to the both pin list $pins[n_j]$ of column-net n_j and $pins[m_j]$ of row-net m_j. The net list of this dummy vertex v_{jj} is simply set to $nets[v_{jj}] = \{n_j, m_j\}$. These vertex additions do not affect the weight computations, since we give zero as the weight of dummy vertices.

4.2 Hypergraph Model for Jagged-like Decomposition

In this section, we propose coarse-grain hypergraph model for jagged-like 2D decomposition of the sparse matrices for parallel SpMxV computations. As stated earlier, SpMxV algorithms that are based on 2D decomposition must use both pre and post communication schemes together. The proposed decomposition method is a two-phase method, in which each phase models either the pre communication cost or post communication cost. Therefore, we have two alternative schemes for this decomposition method. For the sake of simplicity in the presentation we will discuss only one scheme, the one which models the pre communication in the first phase and the post communication in the second phase. The dual discussion holds for the other scheme, that is the one which models the post communication in the first phase and the pre communication in the second phase.

In the jagged-like decomposition model, K-way 2D decomposition of a sparse matrix is achieved by first decomposing the matrix into $\sqrt{K}$ parts using the column-net model proposed in Section 3.3 (rowwise), then each part further decomposed into $\sqrt{K}$ parts using the row-net model (columnwise). Thus resulting decomposition is a 2D decomposition. Figures 4.5–4.9 display each step of this process on a sample 16×16 matrix. Let the input matrix $\mathbf{A}$ be an $M \times M$ matrix. In the first phase, $\mathbf{A}$ is represented by the column-net hypergraph $\mathcal{H}_{\mathcal{R}}$. For the sake of simplicity in the presentation, we assume that underlying parallel architecture is a $\sqrt{K} \times \sqrt{K}$ 2D mesh. Consider a $\sqrt{K}$-way partition Π of $\mathcal{H}_{\mathcal{R}}$. If we partially permute the matrix according to the row partitioning induced by the partition Π, we obtain a matrix $\mathbf{A}^{\Pi}$ which contains roughly $\frac{M}{\sqrt{K}} \times M$ submatrices. In fact, since column-net model tries the work load balance on local SpMxV computations, the resulting submatrices may not contain same number of rows but they will contain roughly equal number of nonzeros. We can assign each submatrix to a row group in 2D mesh. Clearly assignment of submatrices to row groups does not change the total communication volume, so we can safely assume that first submatrix is assigned to first row group and so on. For now, just assume that we will not assign the nonzeros in a column of each submatrix to more than

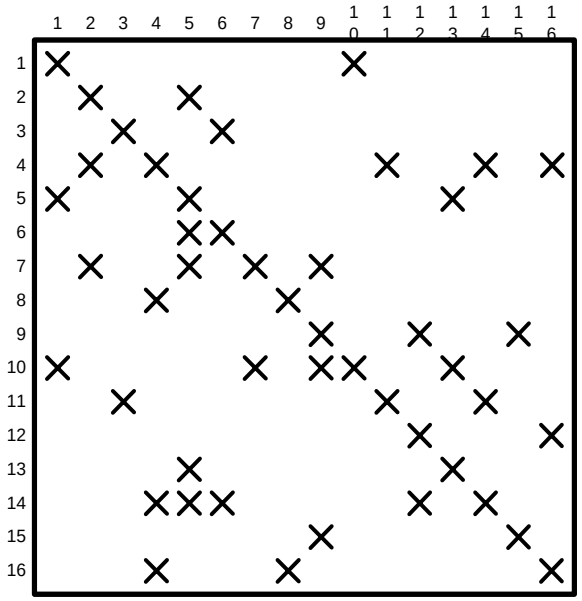

Figure 4.5: A 16×16 nonsymmetric matrix $\mathbf{A}$

one processor in each row processor group, i.e., columns of submatrices are indivisible. We will later explain the correctness of this assumption in our jagged-like decomposition model. The expand operation on the $\mathbf{x}$ vector components will require communication among the row processor groups, not between any pair of processors in a processor row of 2D mesh. Thus this phase minimizes the total volume of communication among the $\sqrt{K}$ row processor groups required during the pre communication step. Figure 4.6 illustrated the column-net representation of the sample hypergraph given in Figure 4.5. We labeled the vertices and nets of hypergraphs with letters "r" and "c" to denote row and column of matrix, for simplicity in the presentation. For a 4-way decomposition of the sample matrix we first decompose matrix into $\sqrt{4} = 2$ parts, to assign each part to a row group, namely to row groups $\{P_1, P_2\}$ and $\{P_3, P_4\}$. The resulting permuted matrix is displayed in Figure 4.7.

In the second phase, each submatrix of $\mathbf{A}^{\Pi}$ is independently decomposed into $\sqrt{K}$ column stripes using the row-net model described in Section 3.3. Since the vertices in the row-net hypergraph model correspond to the columns of the matrices, all nonzeros in a column of each submatrices will be assigned exactly to one processor. Hence, this verifies the assumption in the previous paragraph. That

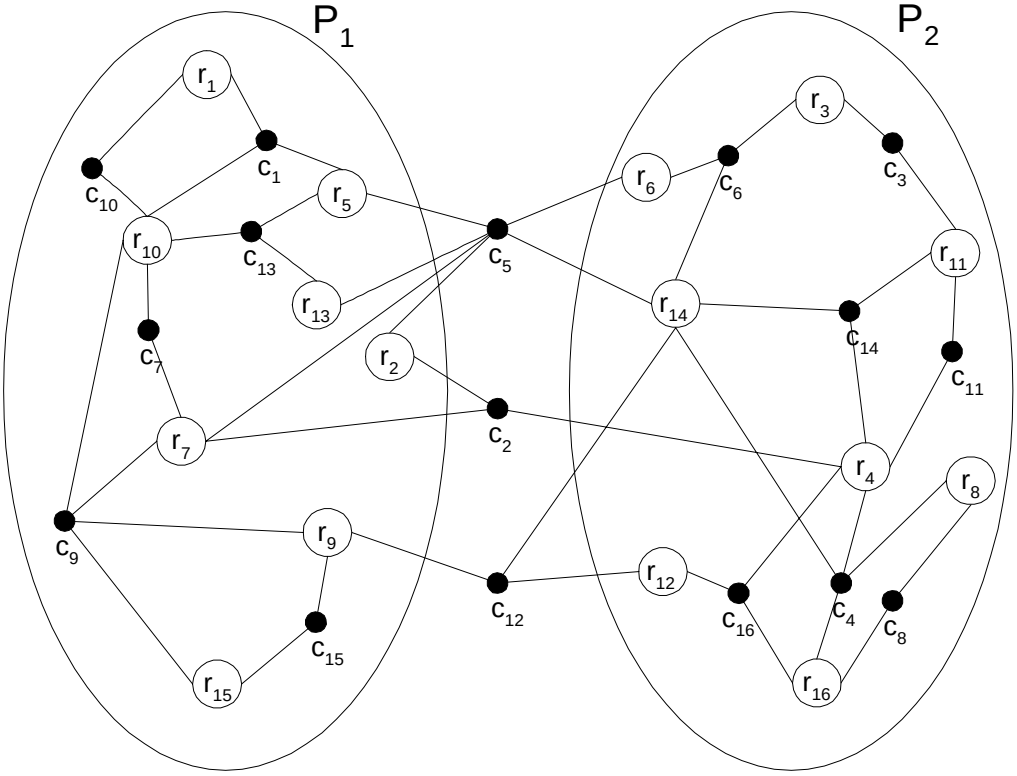

Figure 4.6: Jagged-like 4-way decomposition, Phase 1: Column-net representation $\mathcal{H}_{\mathcal{R}}$ of $\mathbf{A}$ and 2-way partitioning Π of the $\mathcal{H}_R$

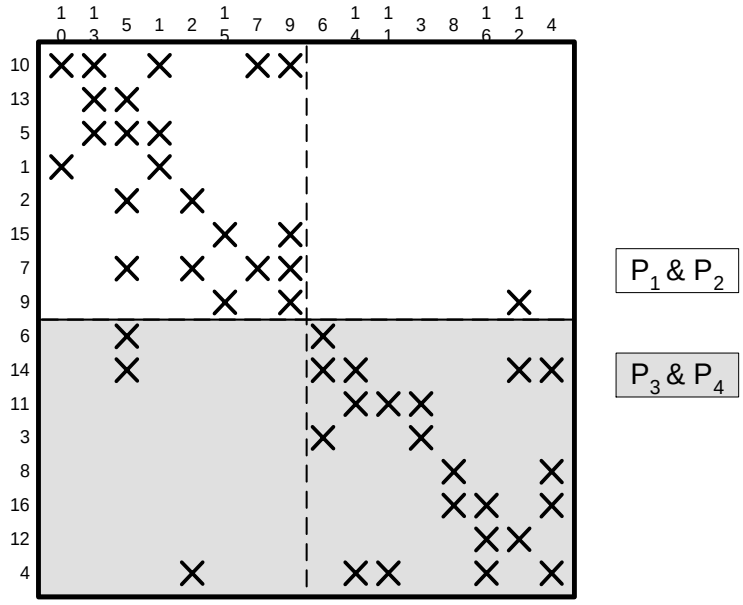

Figure 4.7: Jagged-like 4-way decomposition, Phase 1: 2-way rowwise decomposition of matrix $\mathbf{A}^{\Pi}$ obtained by permuting $\mathbf{A}$ according to the partitioning induced by Π

is, applying the row-net model in the second phase does not disturb the communication requirements of expand operation which is modeled in the first phase. Clearly, the columnwise decomposition in each row processor group, minimizes the total communication volume required during the post communication step, among the $\sqrt{K}$ processors in respective row of 2D mesh . Since each group of processors are assigned different rows of matrix $\mathbf{A}^{\Pi}$, only the processors in each group must communicate to obtain full $\mathbf{y}$ vector. Therefore, sum of the volume of communications of the each processor group exactly models the total volume of communication among the K processors required during the post communication step. Figure 4.8 displays the two row-net hypergraphs corresponding to each submatrix displayed in Figure 4.7. Each hypergraph is partitioned independently, sample partitions of these hypergraphs are also presented in this figure. The final permutation hence processor assignments is displayed in Figure 4.9.

Note that, in the second phase, some vertices may need to exist in more than one hypergraph. These vertices are the vertices corresponding to the columns which have nonzero in more than one row group of $\mathbf{A}^{\Pi}$. In other words, they are the cutnets of the first phase. In the second phase, we simply create a copy of each such column in the decomposition of each submatrix if there is at least one

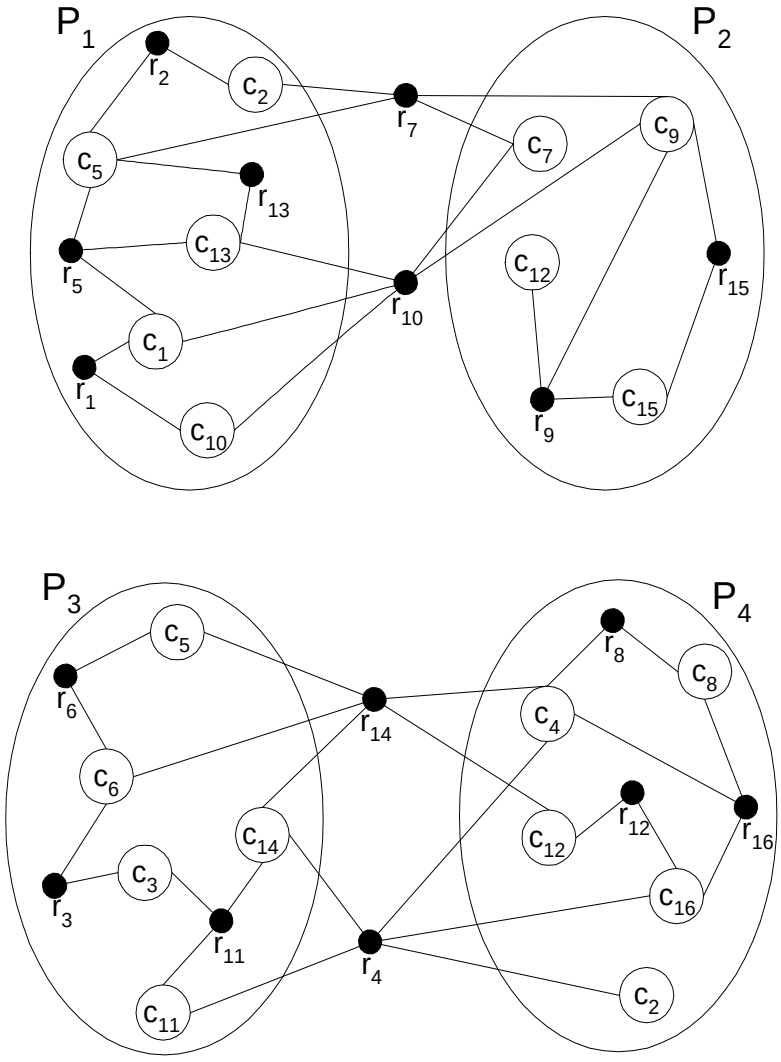

Figure 4.8: Jagged-like 4-way decomposition, Phase 2: Row-net representations of submatrices of **A** and 2-way partitionings

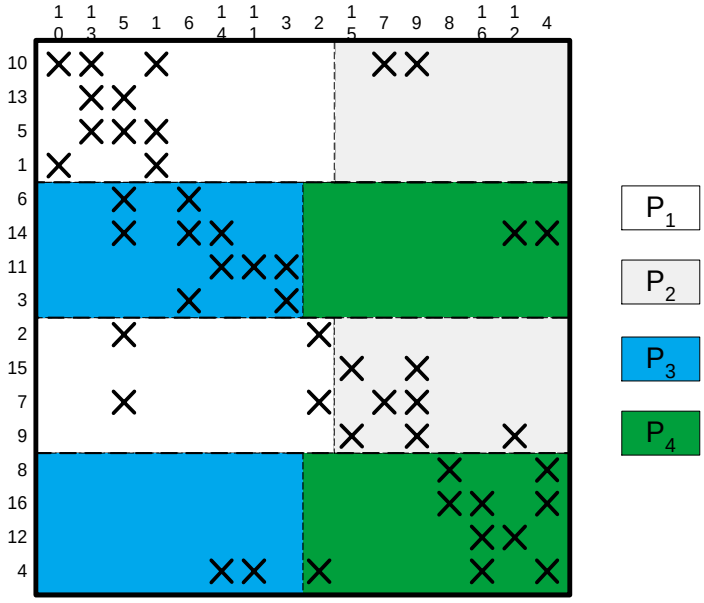

Figure 4.9: Jagged-like 4-way decomposition, Phase 2: Final permuted matrix.

nonzero in that column of submatrix. In other words, for each cutnet n_j in the first phase, we create exactly $\lambda[n_j]$ copies of vertex v_j in the second phase. For example, as seen in Figure 4.6 the column-net c_5 is a cutnet with $\lambda[c_5] = 2$, hence as displayed in Figure 4.8 each hypergraph contains a vertex for column 5, namely c_5. The computational weight of each vertex is just the number of nonzeros in the corresponding column of each submatrix. Hence, maintaining the balance criterion (2.1) corresponds to maintaining the computational load balance during the local SpMxV computations.

In terms of number of messages, the jagged-like decomposition has some nice features over 2D fine-grain hypergraph model. Recall that there is no restriction in the communication pattern of 2D fine-grain model, hence in both pre and post communication phases each processor can communicate with any processor. Thus the bound of total number of message is $2K(K-1)$. In jagged-like communication, in the pre communication phase, the maximum number of messages per processor is $K-\sqrt{K}$. Since the processors in the same row group of 2D mesh do not require communication of $\mathbf{x}$ vector components. In the post communication phase, the maximum number of communication for each processor is $\sqrt{K}-1$. Hence the bound of total number of messages in jagged-like decomposition is $K(K-1)$.

4.3 Hypergraph Model for Checkerboard Decomposition

Most of the SpMxV kernels in the literature [62, 56, 55, 37] work on the 2D checkerboard partitioning with the assumption that underlying interconnection topology is 2D processor mesh. The nice property of 2D checkerboard decomposition is that, all expand operations are only performed among the processors in the same column, and all fold operations are only performed among the processors in the same row of 2D mesh. This nice property is the result of maintaining both row and column *integrity*, that is, the nonzeros of each column (row) of the matrix is assigned to same column (row) of 2D processor mesh. As you may notice, the proposed jagged-like decomposition presented in the previous section already has some part of this nice property. Using the jagged-like decomposition, all fold operations are only performed among the processors in the same row of 2D mesh. However, for expand operations we should take care of extra precautions. In this section, we propose a hypergraph model for 2D checkerboard decomposition of sparse matrices for parallel SpMxV. In the second phase of jagged-like decomposition each column segment assigned to processor groups are represented by a vertex in the row-net model and decomposition in each processor group is done independently. That is, there is no restriction in the assignment of the column segments in the final decomposition. For example, as displayed in Figure 4.8 although both two copies of the vertex c_5 is assigned to first parts of two hypergraphs, the copies of the vertex c_2 are assigned to different parts in those hypergraphs. Hence as we can see in the matrix displayed in Figure 4.9, although the column integrity of column 5 is maintained, the integrity of the column 2 is not maintained. The simplest way to achieve column integrity, is to force the partitioner to put the copied vertices into same part in decomposition of the subsequent matrices. That is, the decomposition of the first submatrix can be done without any restriction, however, in the decomposition of the subsequent matrices the vertices corresponding to cutnet in the first phase are forced to be assigned to same part with the all previous decompositions in the second phase. As you may notice, this enforcement limits the search space of the decomposition of the subsequent submatrices. Furthermore, even the decomposition of first submatrix may blindly cause extra fold operations in the next decompositions.

Here we propose a new method which uses hypergraph model for 2D checkerboard decomposition of sparse matrices for parallel SpMxV computations. The proposed method is again a two phase method where the first phase is identical with the jagged-like decomposition. For the second phase we introduce new hypergraph partitioning problem; *Multi-Constraint Hypergraph Partitioning*. The notion of multi-constraint and multi-objective partitioning has recently become popular in graph partitioning [45, 71] for the parallelization of multi-physics and multi-phase applications. In these applications each constraint effectively corresponds to the computational load of the vertex in different phase of the target parallel algorithm. Hence maintaining balance on each constraint corresponds to maintaining load balance in each phase of the parallel algorithm. The intuition behind the new model for checkerboard decomposition is as follows. Since, the first decomposition in the second phase locks the vertices to the parts in the subsequent decompositions, the locked vertices may cause communication and there is no way to get rid of this communication in the subsequent decompositions. We should find a way to compute these extra communication before locking the vertices. Luckily, we can easily integrate the computation of this cost. That is, we can safely add the nets of hypergraphs of subsequent submatrices to the hypergraph of the first submatrix. Furthermore, we can add all the nets of all submatrices and solve the second phase just in one step. Recall that, in jagged-like decomposition the second phase contains $\sqrt{K}$ steps such that each of them is a $\sqrt{K}$-way decomposition.

The computational weight assignment is as follows. Since we have already decided that "which rows of the matrix will be assigned to which row of the 2D processor mesh", we have also decided computational weight of each column segment. In the new model, each vertex corresponding to columns of matrix will have $\sqrt{K}$ weights. Each weight of a vertex corresponds to the number of nonzeros of the corresponding column in the corresponding row processor group. Hence, maintaining the balance on each weight constraint corresponds to maintaining computational load balance among the processors of each row of 2D mesh. For our specific application, multiple weights of the vertices do not correspond to the weight of different phases. In fact they represent the load of computation that will be executed concurrently.

We can summarize the proposed checkerboard decomposition method as follows. First decompose matrix **A** rowwise into $\sqrt{K}$-way using column-net representation $\mathcal{H}_{\mathcal{R}}$. Let partition $\Pi = \{\mathcal{P}_1, \ldots, \mathcal{P}_K\}$ of $\mathcal{H}_{\mathcal{R}}$ be the partition obtained in the first phase. In the second phase decompose the matrix **A** columnwise into $\sqrt{K}$-way using row-net representation $\mathcal{H}_{\mathcal{C}}$ with multi-constraint on vertex weights. Let $w_{\mathcal{H}_{\mathcal{C}}}[i,j]$ denotes the jth weight of vertex v_i in hypergraph $\mathcal{H}_{\mathcal{C}}$, representing the number of nonzeros of the column i in the jth row group, i.e., $w_{\mathcal{H}_{\mathcal{C}}}[i,j] = |pins_{\mathcal{H}_{\mathcal{R}}}[v_i] \cap \mathcal{P}_j|$.

4.4 Experimental Results

We have tested the validity of the proposed hypergraph models for 2D decomposition by running PaToH on the hypergraphs for the decompositions of various realistic sparse test matrices arising in different application domains [24, 16, 18, 23]. Table 4.1 illustrates the properties of the test matrices listed in the order of increasing number of nonzeros. PaToH is modified to handle multi-constraints to present the checkerboard decomposition results. These 2D decomposition results are compared with the 1D decompositions obtained by running MeTiS using the standard graph models, and PaToH using the 1D column/row-net hypergraph model presented in Section 3.3 (Recall that column-net and row-net models become equivalent in symmetric matrices). As PaToH achieves K-way partitioning through recursive bisection, recursive MeTiS (pMeTiS) was used for the sake of a fair comparison. Another reason for using pMeTiS is that direct K-way partitioning version of MeTiS (kMeTiS) produces 3% worse partitions than pMeTiS in the decomposition of the test matrices, although it is approximately 2 times faster, on the average.

All experiments were carried out on a workstation equipped with a 133 MHz PowerPC processor with 512-Kbyte external cache and 64 Mbytes of memory. We have tested $K = 16$, 32 and 64 way decompositions of every test matrix. For a specific K value, K-way decomposition of a test matrix constitutes a decomposition instance. For jagged-like and checkerboard decompositions we assume that underlying architecture is 4×4, 4×8 and 8×8 2D processor mesh. pMeTiS and

Table 4.1: Properties of test matrices

name	number of rows/cols	number of nonzero			
		total	per row/col		
			min	max	avg
sherman3	5005	20033	1	7	4.00
bcspwr10	5300	21842	2	14	4.12
ken-11	14694	82454	2	243	5.61
nl	7039	105089	1	361	14.93
ken-13	28632	161804	2	339	5.65
cq9	9278	221590	1	702	23.88
co9	10789	249205	1	707	23.10
pltexpA4-6	26894	269736	5	204	10.03
vibrobox	12328	342828	9	121	27.81
cre-d	8926	372266	1	845	41.71
cre-b	9648	398806	1	904	41.34
world	34506	582064	1	972	16.87
mod2	34774	604910	1	941	17.40
finan512	74752	615774	3	1449	8.24

PaToH were run 50 times starting from different random seeds for each decomposition instance. The average performance results are displayed in Tables 4.2–4.4 for each decomposition instance. The percent load imbalance values are below 3% for all decomposition results displayed in these figures, where percent imbalance ratio is defined as $100 \times (W_{max} - W_{avg})/W_{avg}$.

Table 4.2 displays the decomposition performance of the proposed hypergraph models in 2D decomposition together with the standard graph model and 1D hypergraph model. Communication volume values (in terms of the number of words transmitted) are scaled by the number of rows/columns of the respective test matrices. As you can see average percent imbalance values are also displayed in this table. Since both MeTiS and PaToH use recursive bisection to achieve K-way partitioning, it is very hard to impose exact load balance for all instances in both of the tools. Although the main objective of this work is the minimization of the total communication volume, the results for the other performance metrics such as the maximum volume, average number and maximum number of messages handled by a single processor are also displayed in Table 4.3. Recall that, by its nature 2D checkerboard partitioning also minimizes these quantities implicitly. Note that the maximum volume and maximum number of messages determine

the concurrent communication volume and concurrent number of messages, respectively, under the assumption that no congestion occurs in the network.

As seen in Table 4.2, the proposed hypergraph models produce substantially better partitions than the graph model at each decomposition instance in terms of total communication volume. 2D fine-grain hypergraph model is clear winner in the communication volume cost as expected, since it has more degree of freedoms. On the overall average, 2D fine-grain hypergraph model produces 59%, 43% and 34% better decompositions than the 1D graph model, 1D hypergraph model and 2D jagged-like decomposition, respectively. As expected, when the limitations increase in the decomposition, the total volume of communication also increases. However, even the most restricted decomposition method checkerboard decomposition produces 26% better decompositions than the graph model, on the overall average.

Table 4.3 displays the average communication requirements of the proposed hypergraph models in terms of number of messages handled by a single processor. As seen in table, checkerboard decomposition result is shining. This result was expected since the theoretical bound on the maximum number of messages handled by a single processor is $2(\sqrt{K}-1)$. For example, for $K = 64$, the maximum number of messages is $2(\sqrt{(64)}-1) = 2(8-1) = 14$. Whereas, this number is $K-1 = 63$ for 1D graph and hypergraph models, $2(K-1) = 126$ for 2D fine-grain hypergraph model, and $K-1 = 63$ for jagged-like decomposition. Although theoretical bound on the number of messages in 1D graph and hypergraph models and 2D jagged-like decomposition are same, the hypergraph models produce 27% less number of messages than the 1D graph model.

Table 4.4 displays the average execution times of the MeTiS and PaToH for the standard graph and proposed hypergraph models. As seen in the table, 2D fine-grain model has the largest execution time. 2D fine-grain hypergraph model is approximately 2.4 times slower than the 1D hypergraph model. This was expected since 2D fine grain contains 2 times more pins and nets than the 1D hypergraph model, and also number of vertices in the 2D fine-grain model is equal to the number of nonzeros in the matrix, whereas it is the number of rows/columns in 1D hypergraph model. The execution time of jagged-like decomposition is 29% less than the 1D hypergraph decomposition, since it achieves the K-way

decomposition by $\sqrt{K}$ times $\sqrt{K}$-way decomposition. As also seen in Table 4.4, both 1D hypergraph decomposition and 2D checkerboard decomposition using PaToH is approximately 3 times slower than the standard graph model using MeTiS. Here, we should note that we have used PaToH without any modification (except multi-constraint code added for checkerboard), that is, current version of PaToH contains net weight variables, and is able to balance on nets, hence there are some variables for each cell and net which are maintained during coarse of partitioning. By modifying PaToH (i.e., removing the unnecessary code segments and variables) we may expect substantial reduce in running time of hypergraph models.

Table 4.2: Average communication volume requirements of the proposed hypergraph models and standard graph model. "tot" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled by a single processor. "bal" denotes the percent imbalance ratio found by the respective tool for each instance.

		Graph Model			1D Hypergraph			2D Fine-grain			2D Jagged-like			2D Checkerboard		
		comm. vol.			comm. vol.			comm. vol.			comm. vol.			comm. vol.		
name	K	tot	max	bal	tot	max	bal	tot	max	bal	tot	max	bal	tot	max	bal
sherman3	16	0.31	0.03	0.1	0.25	0.02	0.5	0.25	0.02	0.4	0.26	0.03	0.4	0.30	0.03	1.1
	32	0.46	0.02	0.3	0.37	0.02	1.0	0.36	0.02	0.6	0.38	0.02	1.0	0.45	0.02	4.0
	64	0.64	0.02	2.3	0.53	0.01	2.2	0.50	0.01	1.0	0.51	0.01	2.0	0.72	0.02	9.0
bcspwr10	16	0.09	0.01	0.2	0.08	0.01	1.0	0.07	0.01	0.9	0.08	0.01	1.6	0.10	0.01	1.1
	32	0.15	0.01	0.9	0.13	0.01	1.5	0.12	0.01	1.7	0.13	0.01	2.5	0.17	0.01	1.5
	64	0.23	0.01	2.7	0.22	0.01	2.5	0.19	0.01	2.2	0.21	0.01	3.1	0.28	0.01	1.7
ken-11	16	0.93	0.08	0.3	0.60	0.05	2.1	0.14	0.02	3.5	0.73	0.07	1.1	0.84	0.08	1.4
	32	1.17	0.06	4.8	0.74	0.03	2.6	0.29	0.02	3.6	0.88	0.05	2.1	0.98	0.06	2.7
	64	1.45	0.04	13.5	0.93	0.02	3.9	0.48	0.02	3.7	1.03	0.03	2.8	1.17	0.04	3.4
nl	16	1.70	0.15	0.5	1.06	0.10	0.3	0.74	0.08	0.1	1.00	0.09	0.5	1.15	0.10	0.1
	32	2.25	0.10	1.7	1.49	0.07	1.6	1.05	0.07	0.1	1.30	0.07	1.3	1.54	0.07	0.8
	64	3.04	0.07	7.7	2.20	0.05	4.5	1.38	0.05	0.3	1.63	0.05	2.0	2.11	0.05	1.1
ken-13	16	0.94	0.08	0.3	0.55	0.04	2.2	0.08	0.01	4.1	0.72	0.07	2.6	0.79	0.07	2.7
	32	1.17	0.05	1.9	0.63	0.03	3.1	0.17	0.02	5.2	0.81	0.05	3.7	0.89	0.06	3.9
	64	1.40	0.03	8.3	0.79	0.02	4.0	0.39	0.02	5.3	0.92	0.03	4.0	1.03	0.03	5.0
cq9	16	1.70	0.17	0.3	0.99	0.12	1.0	0.50	0.08	1.1	0.91	0.12	2.0	1.15	0.13	0.8
	32	2.43	0.15	1.2	1.45	0.08	1.8	0.79	0.09	1.6	1.27	0.08	2.4	1.62	0.10	1.5
	64	3.73	0.12	6.0	2.33	0.06	8.3	1.22	0.07	1.8	1.72	0.06	3.0	2.42	0.08	2.1
co9	16	1.50	0.16	0.3	0.94	0.11	0.9	0.47	0.07	0.9	0.88	0.11	1.3	1.12	0.12	0.4
	32	2.07	0.12	0.9	1.36	0.08	1.9	0.74	0.07	1.3	1.20	0.08	2.0	1.55	0.09	1.9
	64	3.10	0.09	3.4	2.17	0.06	3.8	1.09	0.06	1.8	1.63	0.05	3.0	2.24	0.07	1.8
pltexpA4-6	16	0.34	0.03	0.1	0.30	0.03	0.1	0.20	0.02	1.1	0.27	0.03	1.7	0.29	0.03	0.1
	32	0.55	0.03	0.3	0.51	0.02	0.2	0.29	0.01	1.3	0.47	0.02	3.1	0.53	0.02	0.6
	64	0.98	0.03	0.6	0.86	0.02	1.0	0.51	0.01	1.4	0.74	0.02	2.9	0.85	0.02	1.3
vibrobox	16	1.24	0.11	0.3	1.06	0.08	0.1	0.79	0.07	0.0	0.95	0.07	0.1	1.07	0.08	0.1
	32	1.73	0.08	0.8	1.53	0.06	0.4	1.06	0.06	0.0	1.31	0.05	1.1	1.49	0.06	0.2
	64	2.28	0.05	2.0	2.08	0.05	1.1	1.43	0.05	0.3	1.64	0.03	1.6	2.01	0.04	0.4
cre-d	16	2.82	0.24	0.9	2.00	0.17	1.3	1.15	0.12	0.0	1.63	0.19	1.3	1.81	0.20	1.4
	32	4.12	0.19	2.5	2.90	0.14	2.6	1.77	0.11	0.1	2.22	0.16	1.9	2.53	0.17	2.3
	64	5.95	0.14	5.6	4.14	0.10	6.6	2.55	0.10	0.2	2.72	0.10	2.7	3.44	0.10	4.5
cre-b	16	2.62	0.23	0.9	2.02	0.18	1.0	1.01	0.11	0.0	1.58	0.21	1.0	1.81	0.22	0.7
	32	3.90	0.18	2.2	2.88	0.15	1.5	1.55	0.11	0.0	2.15	0.18	1.8	2.55	0.20	1.9
	64	5.73	0.14	5.6	4.08	0.12	5.8	2.26	0.10	0.0	2.73	0.11	2.3	3.49	0.12	3.2
world	16	0.59	0.05	0.1	0.54	0.06	0.6	0.23	0.05	1.5	0.63	0.08	1.5	0.70	0.09	1.5
	32	0.84	0.04	0.3	0.76	0.05	1.1	0.41	0.04	1.8	0.86	0.06	2.1	0.96	0.07	1.7
	64	1.19	0.03	0.7	1.06	0.04	1.7	0.62	0.04	1.9	1.07	0.04	2.9	1.30	0.04	2.1
mod2	16	0.57	0.05	0.1	0.52	0.06	0.8	0.24	0.05	1.8	0.60	0.08	1.7	0.67	0.09	1.5
	32	0.79	0.04	0.3	0.72	0.04	1.2	0.41	0.05	2.1	0.82	0.06	2.1	0.91	0.07	1.6
	64	1.14	0.03	0.8	1.02	0.04	1.8	0.62	0.04	1.8	1.03	0.04	3.1	1.27	0.04	2.3
finan512	16	0.20	0.03	0.0	0.16	0.03	2.8	0.07	0.02	3.5	0.20	0.06	5.2	0.21	0.07	4.5
	32	0.27	0.02	1.0	0.21	0.02	3.2	0.10	0.02	3.8	0.25	0.07	5.4	0.28	0.08	5.3
	64	0.38	0.01	1.7	0.31	0.01	4.3	0.20	0.02	4.1	0.38	0.05	6.2	0.46	0.05	4.9
Averages over K																
average	16	1.11	0.10	0.3	0.79	0.08	1.1	0.42	0.05	1.4	0.74	0.09	1.6	0.86	0.09	1.2
	32	1.56	0.08	1.4	1.12	0.06	1.7	0.65	0.05	1.7	1.00	0.07	2.3	1.17	0.08	2.1
	64	2.23	0.06	4.4	1.62	0.04	3.7	0.96	0.04	1.8	1.28	0.04	3.0	1.63	0.05	3.1
overall average		1.63	0.08	2.0	1.18	0.06	2.1	0.68	0.05	1.6	1.01	0.07	2.3	1.22	0.07	2.1

Table 4.3: Average communication requirements of the proposed hypergraph models and standard graph model. "avg" and "max" denote the average and maximum number of messages handled by a single processor

		Graph Model		1D Hypergraph		2D Fine-grain		2D Jagged-like		2D Checkerboard	
name	K	avg	max	avg	max	avg	max	avg	max	avg	max
sherman3	16	5.30	8.10	4.46	7.22	8.38	13.90	5.16	8.36	4.09	5.34
	32	6.48	10.94	5.81	10.44	10.07	17.60	6.34	11.00	5.83	8.92
	64	7.42	13.40	6.94	13.40	11.01	20.78	7.20	13.00	7.26	11.14
bcspwr10	16	4.21	7.28	4.29	7.30	7.14	12.04	4.31	7.20	3.99	5.58
	32	4.79	9.30	4.65	8.80	7.49	13.86	4.70	9.18	4.94	8.04
	64	5.20	10.24	4.93	9.56	7.32	13.80	4.94	9.70	5.52	9.50
ken-11	16	13.99	15.00	12.91	15.00	10.79	21.16	13.69	15.00	5.98	6.00
	32	26.00	30.48	21.19	30.96	18.85	40.90	22.84	28.88	9.62	10.00
	64	40.48	55.14	32.22	60.80	28.23	76.28	28.93	45.04	13.21	14.00
nl	16	14.99	15.00	13.30	15.00	23.87	28.56	13.75	15.00	6.00	6.00
	32	27.88	31.00	20.39	27.58	35.98	50.48	21.61	27.80	9.95	10.00
	64	38.35	58.98	26.13	41.32	42.43	75.94	25.67	40.68	13.39	14.00
ken-13	16	14.77	15.00	13.87	15.00	9.39	19.28	12.52	15.00	6.00	6.00
	32	29.02	31.00	22.79	31.00	11.22	35.62	21.07	29.92	9.81	10.00
	64	50.81	61.92	35.93	63.00	20.51	71.54	29.29	47.96	13.28	14.00
cq9	16	14.88	15.00	12.62	14.92	18.03	26.08	13.36	14.96	6.00	6.00
	32	21.96	30.60	17.87	26.78	24.54	45.38	18.37	28.00	9.75	10.00
	64	32.27	56.58	22.67	41.12	30.72	75.26	21.27	42.32	12.89	14.00
co9	16	14.81	15.00	12.82	14.92	20.00	26.40	13.47	15.00	6.00	6.00
	32	19.62	29.46	17.55	26.20	26.84	45.57	17.93	27.68	9.66	10.00
	64	29.99	53.04	21.85	39.52	31.13	73.50	20.37	40.04	12.77	14.00
pltexpA4-6	16	10.05	13.62	10.11	13.62	14.78	22.80	7.53	10.84	5.47	6.00
	32	15.86	25.40	14.73	25.38	20.51	36.96	11.23	19.54	8.43	10.00
	64	20.48	45.20	17.35	38.12	21.40	52.88	14.86	32.64	9.95	12.58
vibrobox	16	12.84	14.86	10.14	12.42	23.27	28.32	10.64	13.20	5.82	6.00
	32	20.85	27.20	14.77	20.14	31.28	47.88	15.24	20.44	9.26	10.00
	64	28.85	40.48	19.58	30.84	35.38	80.68	19.74	27.38	11.53	13.04
cre-d	16	14.90	15.00	11.78	15.00	26.05	29.67	12.26	15.00	5.80	6.00
	32	28.59	31.00	19.49	31.00	41.37	54.87	18.84	28.44	9.19	10.00
	64	47.36	63.00	29.73	61.28	55.76	92.27	24.86	51.48	11.78	14.00
cre-b	16	14.78	15.00	12.13	15.00	25.91	29.60	12.87	15.00	5.91	6.00
	32	28.57	31.00	19.97	31.00	40.33	55.47	19.49	28.44	9.51	10.00
	64	46.42	63.00	29.98	61.34	52.72	89.80	25.10	50.32	12.29	14.00
world	16	11.78	15.00	6.09	15.00	16.57	27.68	9.29	14.38	5.12	6.00
	32	18.00	30.94	8.19	30.94	23.14	51.36	13.79	25.68	7.46	10.00
	64	20.58	57.58	11.58	58.08	27.42	87.52	16.37	41.78	9.47	14.00
mod2	16	10.95	15.00	5.59	14.92	13.02	27.12	8.71	14.16	4.92	6.00
	32	14.59	29.72	7.42	27.84	18.68	48.44	12.10	24.24	7.12	10.00
	64	17.84	50.84	10.51	46.42	24.44	80.72	14.56	37.96	8.92	14.00
finan512	16	4.35	7.40	3.48	7.40	9.24	19.53	4.50	9.20	4.08	5.90
	32	6.39	13.64	4.15	13.58	10.75	34.47	5.33	14.04	5.12	9.46
	64	8.80	26.40	5.37	26.40	14.90	62.33	5.82	20.36	6.12	11.80
Averages over K											
average	16	11.61	13.30	9.54	13.05	16.17	23.72	10.15	13.02	5.37	5.92
	32	19.19	25.83	14.21	24.40	22.93	41.35	14.92	23.09	8.26	9.74
	64	28.20	46.84	19.63	42.23	28.81	68.09	18.50	35.76	10.60	13.15
overall average		19.67	28.66	14.46	26.56	22.64	44.39	14.52	23.96	8.08	9.60

Table 4.4: Average execution times, in seconds, of the MeTiS and PaToH for the standard graph model and proposed hypergraph models. Numbers in the parentheses are the normalized execution times with respect to Graph Model using MeTiS.

		Graph Model		1D Hypergraph		2D Fine-grain		2D Jagged-like		2D Checkerboard	
name	K	exec. time		exec. time		exec. time		exec. time		exec. time	
sherman3	16	0.53	(1.00)	0.94	(1.77)	1.60	(3.03)	0.60	(1.13)	0.85	(1.61)
	32	0.61	(1.00)	1.10	(1.79)	2.05	(3.34)	0.65	(1.06)	1.07	(1.75)
	64	0.71	(1.00)	1.22	(1.71)	2.42	(3.39)	0.82	(1.15)	1.29	(1.80)
bcspwr10	16	0.28	(1.00)	1.01	(3.62)	2.04	(7.28)	0.66	(2.35)	0.86	(3.06)
	32	0.34	(1.00)	1.24	(3.63)	2.47	(7.25)	0.70	(2.05)	1.02	(3.01)
	64	0.42	(1.00)	1.39	(3.34)	2.86	(6.86)	0.85	(2.03)	1.30	(3.13)
ken-11	16	1.77	(1.00)	3.86	(2.19)	6.47	(3.66)	2.51	(1.42)	3.21	(1.82)
	32	1.98	(1.00)	4.74	(2.39)	8.10	(4.09)	2.78	(1.40)	3.73	(1.88)
	64	2.35	(1.00)	5.31	(2.26)	9.87	(4.20)	3.19	(1.36)	4.39	(1.87)
nl	16	1.21	(1.00)	3.75	(3.09)	8.58	(7.07)	2.54	(2.09)	3.39	(2.79)
	32	1.43	(1.00)	4.46	(3.12)	10.56	(7.39)	2.59	(1.81)	3.84	(2.68)
	64	1.54	(1.00)	5.13	(3.34)	12.33	(8.03)	3.13	(2.04)	4.48	(2.92)
ken-13	16	3.84	(1.00)	8.33	(2.17)	12.81	(3.33)	5.20	(1.35)	6.69	(1.74)
	32	4.50	(1.00)	9.81	(2.18)	16.39	(3.64)	5.80	(1.29)	7.77	(1.73)
	64	4.78	(1.00)	10.99	(2.30)	20.71	(4.33)	6.67	(1.40)	9.16	(1.92)
cq9	16	2.12	(1.00)	5.58	(2.64)	14.41	(6.81)	4.15	(1.96)	5.42	(2.56)
	32	2.46	(1.00)	6.43	(2.61)	17.13	(6.96)	4.47	(1.82)	6.37	(2.59)
	64	2.80	(1.00)	7.90	(2.82)	20.49	(7.31)	5.16	(1.84)	7.20	(2.57)
co9	16	2.42	(1.00)	6.58	(2.72)	16.01	(6.63)	4.78	(1.98)	6.21	(2.57)
	32	2.84	(1.00)	7.89	(2.78)	20.29	(7.14)	5.10	(1.80)	7.52	(2.65)
	64	3.07	(1.00)	9.15	(2.99)	24.54	(8.01)	6.17	(2.01)	8.72	(2.84)
pltexpA4-6	16	3.22	(1.00)	12.26	(3.81)	28.69	(8.92)	8.78	(2.73)	11.27	(3.50)
	32	3.84	(1.00)	15.87	(4.13)	36.92	(9.61)	9.02	(2.35)	13.67	(3.56)
	64	4.32	(1.00)	18.20	(4.21)	42.06	(9.73)	11.41	(2.64)	17.09	(3.95)
vibrobox	16	2.77	(1.00)	12.64	(4.56)	28.83	(10.40)	10.92	(3.94)	15.88	(5.73)
	32	3.25	(1.00)	15.11	(4.65)	35.43	(10.90)	11.52	(3.54)	18.86	(5.80)
	64	3.49	(1.00)	17.35	(4.97)	41.50	(11.88)	13.27	(3.80)	21.81	(6.24)
cre-d	16	4.18	(1.00)	9.76	(2.34)	31.30	(7.49)	11.14	(2.67)	13.27	(3.18)
	32	4.80	(1.00)	11.71	(2.44)	38.77	(8.08)	12.88	(2.69)	14.92	(3.11)
	64	5.03	(1.00)	13.66	(2.72)	45.50	(9.05)	14.10	(2.80)	17.48	(3.48)
cre-b	16	4.41	(1.00)	10.47	(2.38)	32.05	(7.27)	11.04	(2.50)	14.06	(3.19)
	32	5.01	(1.00)	12.13	(2.42)	39.88	(7.96)	11.77	(2.35)	15.73	(3.14)
	64	5.42	(1.00)	14.20	(2.62)	46.92	(8.66)	13.83	(2.55)	18.63	(3.44)
world	16	5.76	(1.00)	19.37	(3.36)	48.24	(8.37)	15.28	(2.65)	20.88	(3.62)
	32	7.04	(1.00)	23.52	(3.34)	63.34	(9.00)	17.13	(2.43)	25.10	(3.57)
	64	8.16	(1.00)	28.89	(3.54)	77.90	(9.54)	19.59	(2.40)	29.79	(3.65)
mod2	16	5.85	(1.00)	20.51	(3.51)	52.13	(8.92)	16.22	(2.77)	20.57	(3.52)
	32	7.19	(1.00)	23.85	(3.32)	66.18	(9.20)	17.42	(2.42)	25.72	(3.58)
	64	7.96	(1.00)	29.30	(3.68)	74.27	(9.33)	20.93	(2.63)	30.32	(3.81)
finan512	16	7.84	(1.00)	25.72	(3.28)	55.13	(7.03)	16.49	(2.10)	20.05	(2.56)
	32	9.56	(1.00)	31.49	(3.30)	67.26	(7.04)	17.01	(1.78)	25.62	(2.68)
	64	11.17	(1.00)	37.29	(3.34)	79.71	(7.13)	21.69	(1.94)	31.12	(2.78)
Averages over K											
average	16	-	(1.00)	-	(2.96)	-	(6.87)	-	(2.26)	-	(2.96)
	32	-	(1.00)	-	(3.01)	-	(7.26)	-	(2.06)	-	(2.98)
	64	-	(1.00)	-	(3.13)	-	(7.68)	-	(2.18)	-	(3.17)
overall average		-	(1.00)	-	(3.03)	-	(7.27)	-	(2.17)	-	(3.04)

Chapter 5

Hypergraph Partitioning-Based Sparse Matrix Ordering

The first step of a direct method to solve linear system $Zx = b$ is a heuristic reordering of the rows and columns of Z to reduce *fill* in the factor matrices. The fill is the set of zero entries in Z that become nonzero in the factor matrices. Reducing the fill usually causes a faster and less memory intensive factorization. Minimum degree [74] algorithm (MD) is the most commonly used heuristic for reordering. An alternative for reordering is nested dissection [27]. Although nested dissection has some nice theoretical results [27], it has not been used widely until the development of recent multilevel graph partitioning tools. Here, we will demonstrate the flaw of the graph model for sparse matrix ordering in multilevel framework. We will propose a novel hypergraph partitioning-based nested dissection ordering for matrices arising in the solution of LP problems using an interior point method. Furthermore, we will generalize the proposed method to order any symmetric matrices.

5.1 Flaws of the Graph Model in Multilevel Framework

As discussed in Sections 2.4 and 2.5, most of the nested dissection tools [31, 38, 44] are based on successful multilevel graph partitioning tools [31, 36, 44] with some extra initial partitioning and refinement strategies specific to the solution of the GPVS problem. As also discussed in Section 2.4, a multilevel partitioning tool basically contains three phases; coarsening, initial partitioning and uncoarsening. During the coarsening phase, vertices are visited in some order and usually two (or more) of them selected according to a some criteria to construct the vertices of coarsened graph. Consider the two examples displayed in Figure 5.1 as partial illustration of two different GPVS partitioning results at some level m of multilevel GPVS tool. In the first one, $\ell+1$ vertices $\{v_i, v_{i+1}, \ldots, v_{i+\ell}\}$ are coalesced to construct vertex $v_{i-\ell}$ as a result of one or more levels of coarsening. This is a valid and narrow separator for level m. GPVS tool computes the cost of this separator as $\ell+1$ at this level. However, obviously this separator is not a narrow separator in the original graph, it is a wide separator in the original graph. In other words, there is a subset of those vertices which is a valid narrow separator of the original graph. In fact anyone of the vertices is a valid separator of cost 1 in the original graph. Similarly, for the second example, GPVS tool computes the cost of the separator as 3, however, there is a subset of constituent vertices of $v_{ijk} = \{v_i, v_j, v_k\}$ which is a valid narrow separator of cost 1 in the original graph (i.e., either $\{v_i\}$ or $\{v_k\}$).

In GPES, the multilevel framework does not have this kind of flaw. That is, for an edge separator $\mathcal{E}_S$ at level m, there is no subset of $\mathcal{E}_S$ which is a valid edge separator of the original graph.

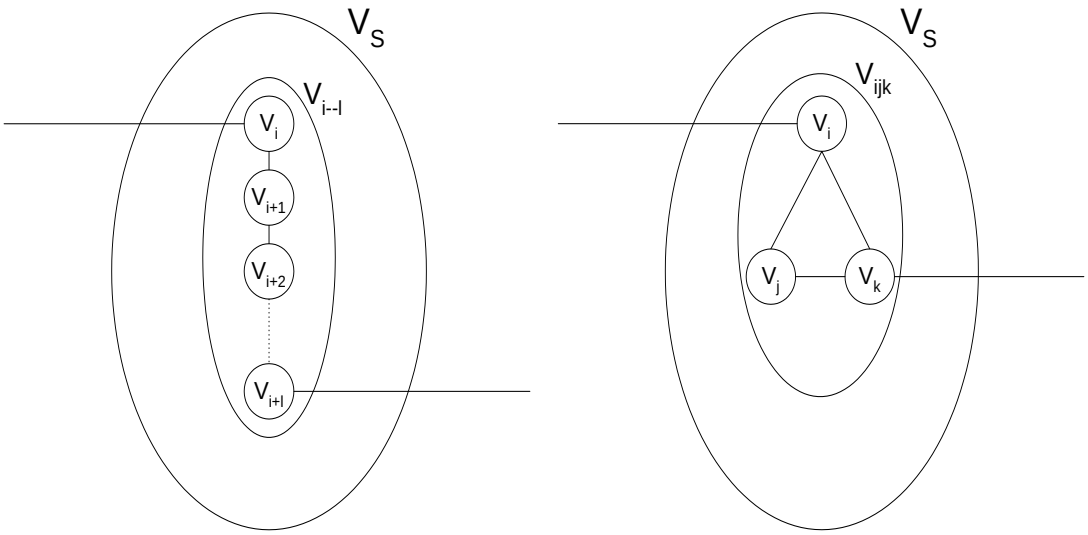

Figure 5.1: Partial illustration of two sample GPVS result to demonstrate the flaw of the graph model in multilevel framework.

5.2 Describing GPVS Problem as a HP Problem

Consider a hypergraph $\mathcal{H} = (\mathcal{U}, \mathcal{N})$ and its NIG representation $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ as discussed in Section 2.3. A K-way vertex partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_K\}$ of $\mathcal{H}$ can be decoded as $(K+1)$-way net partitioning $\Pi_{HP} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_K; \mathcal{N}_S\}$ of $\mathcal{H}$ as follows. Here, $\mathcal{N}_k$ corresponds to the internal nets of part $\mathcal{U}_k$, i.e., for $1 \leq k \leq K$, $\mathcal{N}_k = \{n_j | pins[n_j] \cap \mathcal{U}_k = pins[n_j]\}$. $\mathcal{N}_S$ corresponds to the external nets. In particular, a 2-way vertex partition $\Pi_{HP} = \{\mathcal{U}_1, \mathcal{U}_2\}$ of $\mathcal{H}$ can be decoded as 3-way net partitioning $\Pi_{HP} = \{\mathcal{N}_1, \mathcal{N}_2; \mathcal{N}_S\}$ of $\mathcal{H}$. Here, we consider net-partition $\Pi_{HP} = \{\mathcal{N}_1, \mathcal{N}_2; \mathcal{N}_S\}$ of $\mathcal{H}$ as inducing a GPVS $\Pi_{GPVS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$ on its NIG representation $\mathcal{G}$, where $\mathcal{V}_1 \equiv \mathcal{N}_1$, $\mathcal{V}_2 \equiv \mathcal{N}_2$, $\mathcal{V}_S \equiv \mathcal{N}_S$. Let $Adj_{\mathcal{H}}(n_i)$ denote the set of nets that share pin(s) with net n_i. Consider an internal net n_i of part $\mathcal{U}_1$,i.e., $n_i \in \mathcal{U}_1$. It is clear that we have either $Adj_{\mathcal{H}}(n_i) \subseteq \mathcal{N}_1$ or $Adj_{\mathcal{H}}(n_i) \subseteq \mathcal{N}_1 \cup \mathcal{N}_S$. Recall that NIG $\mathcal{G}$ contains a vertex v_i for each net n_i of $\mathcal{H}$. So we have either $Adj_{\mathcal{G}}(v_i) \subseteq \mathcal{V}_1$ or $Adj_{\mathcal{G}}(v_i) \subseteq \mathcal{V}_1 \cup \mathcal{V}_S$ in NIG $\mathcal{G}$. In other words, $Adj_{\mathcal{G}}(v_i) \cap \mathcal{V}_2 = \emptyset$. In the respective Π_{GPVS}, this corresponds to $Adj_{\mathcal{G}}(\mathcal{V}_1) \cap V_2 = Adj_{\mathcal{G}}(V_2) \cap \mathcal{V}_1 = \emptyset$ which in turn corresponds to $Adj_{\mathcal{G}}(V_1) \subseteq \mathcal{V}_S$ and $Adj_{\mathcal{G}}(V_2) \subseteq \mathcal{V}_S$. Thus, $\mathcal{V}_S$ of Π_{GPVS} constitutes a valid separator of size $|\mathcal{V}_S| = |\mathcal{N}_S|$. Recall that in the GPVS problem, balancing is defined on the

vertex counts of parts $\mathcal{V}_1$ and $\mathcal{V}_2$. Hence, the GPVS problem on NIG $\mathcal{G}$ can be described as an HP problem according to the net-cut metric (Equation (2.4.a) with $c_j = 1$) with balancing on the internal nets of parts $\mathcal{U}_1$ and $\mathcal{U}_2$.

From a matrix theoretical point of view, let A be a matrix and $\mathcal{H}$ be its row-net hypergraph representation, the NIG $\mathcal{G}$ would be the standard graph representation of matrix AA^T. Hence, finding a doubly-bordered form of matrix AA^T (finding GPVS on $\mathcal{G}$) is equivalent to finding a singly-bordered form of matrix A (finding a net partition on $\mathcal{H}$). Although this finding looks very impressive, it is not very useful on itself. For a general GPVS problem on $\mathcal{G}$, which is equivalent to finding a doubly-bordered form of associated matrix (say Z) of $\mathcal{G}$, we should know the decomposition of matrix Z as $Z = AA^T$.

5.3 Ordering for LP Problems

The interior point methods for solving linear programming (LP) problems require the solution of $Zx = b$ repeatedly, where $Z = ADA^T$. Here, D is a diagonal matrix whose numerical values change in each iteration, however constraint matrix A remains unchanged. The linear systems are usually solved by factoring matrix Z. As discussed earlier, factorization introduces fills, and hence, the fill-reducing reordering heuristics are used just before the factorization.

Here, we propose a hypergraph-partitioning-based nested dissection ordering for the ordering of matrix $Z = ADA^T$. Nested dissection ordering requires finding a doubly-bordered (DB) form of the matrix. In DB form, borders correspond to separator S, and block-diagonals correspond to X and Y parts of nested dissection as mentioned earlier. Nested dissection simply orders rows/columns of S after the rows/columns of X and Y. Together with the formulation of GPVS problem as an HP problem, described in the previous section, we can construct an ordering of Z by just recursively dissecting A. That is, in each bisection of A cutnets in $\mathcal{N}_S$ correspond to separator vertices in S in the nested dissection. Figure 5.2 and 5.3 illustrate this finding in a two level incomplete nested dissection.

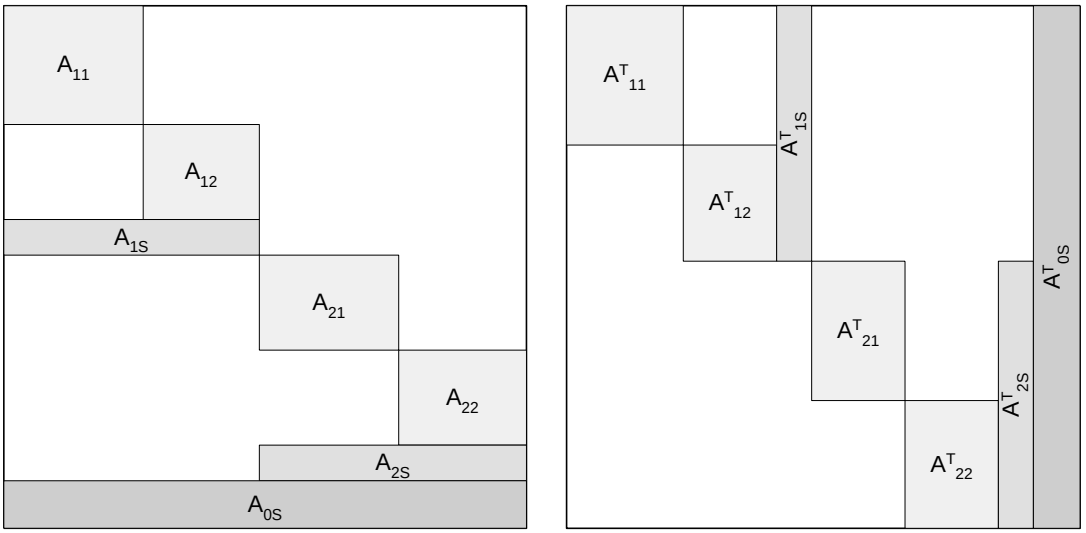

Figure 5.2: 2 level recursive partitioning of A and its transpose A^T

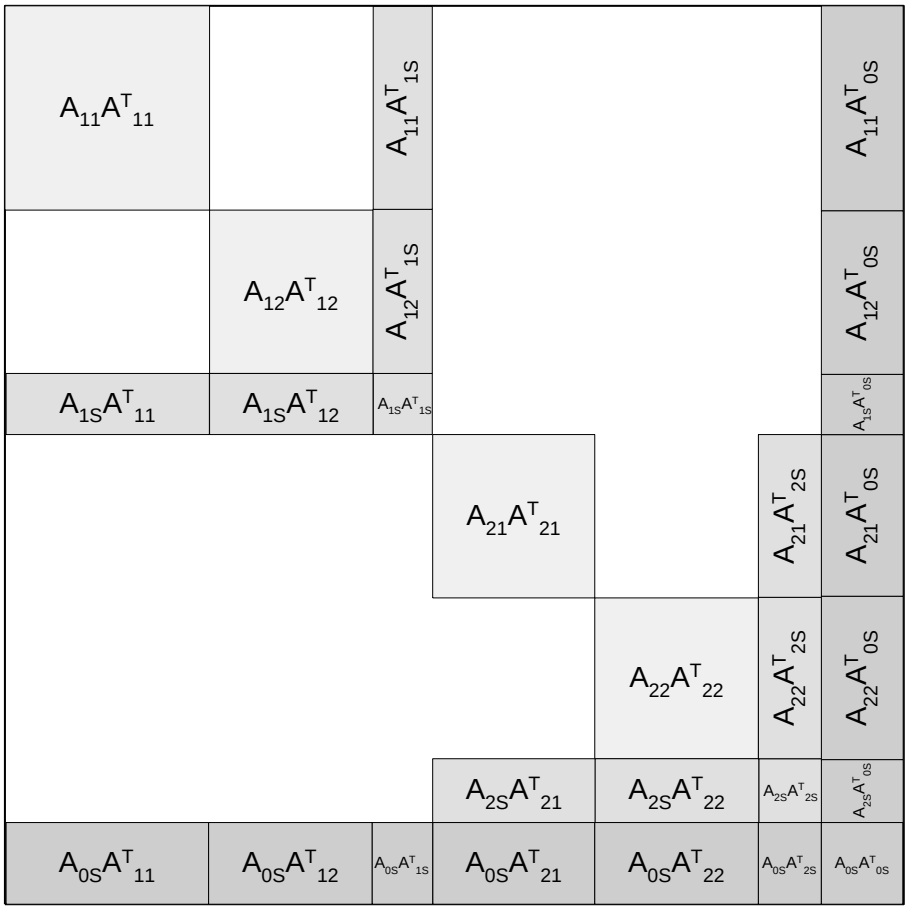

Figure 5.3: Resulting DB form of AA^T, for matrix A displayed in Figure 5.2

```
initialize delete[u_i] ← FALSE for u_i ∈ U
for node u_i ∈ U in non-increasing degree order
    if delete[u_i] = FALSE then
        for each n_j ∈ nets[u_i] do
            if deg_G(v_j) = deg_H(u_i) − 1 then
                for each u_k ∈ pins[n_j] do
                    if u_k ≠ u_i and delete[u_k] = FALSE then
                        delete[u_k] ← TRUE
delete all nodes u_i of H with delete[u_i] = TRUE
```

Figure 5.4: Clique discarding algorithm for $\mathcal{H} = (\mathcal{U}, \mathcal{N})$. Here, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the NIG representation of $\mathcal{H}$

Since our main aim is to achieve a GPVS on NIG $\mathcal{G}$ through a partitioning on $\mathcal{H}$, we may simplify $\mathcal{H}$ without disturbing its NIG representation $\mathcal{G}$. That is, let $\mathcal{H}'$ be the simplified version of $\mathcal{H}$ such that the NIG representation of both of them is exactly the same (i.e., $\mathcal{G}$), then we can safely use $\mathcal{H}'$ instead of $\mathcal{H}$ to find a GPVS partition on $\mathcal{G}$. Here, we propose two simplification methods.

5.3.1 Clique Discarding

Let $\mathcal{H}$ be the row-net hypergraph representation of matrix A, clearly its NIG $\mathcal{G}$ is the graph representation of matrix AA^T. As mentioned in Section 2.3, the NIG representation $\mathcal{G}$ for a hypergraph $\mathcal{H}$ can also be obtained by applying the clique-net model to the dual hypergraph of $\mathcal{H}$. In other words, each node of $\mathcal{H}$ (columns of A) induces a clique among the vertices of $\mathcal{G}$ that correspond to nets incident on that node in $\mathcal{H}$ (rows with nonzero at that column). Hence, if the two columns have exactly the same sparsity pattern (i.e., have nonzeros in the same rows) they induce the same clique in $\mathcal{G}$. Furthermore, if the sparsity pattern of a vertex, say v_i, is a subset of another vertex, say v_j, then clique edges which are induced by v_i are a subset of clique edges which are induced by v_j, so v_i become redundant in the partitioning of $\mathcal{H}$ to find a GPVS partition on NIG $\mathcal{G}$.

Here we present a simple yet effective algorithm to find the redundant nodes

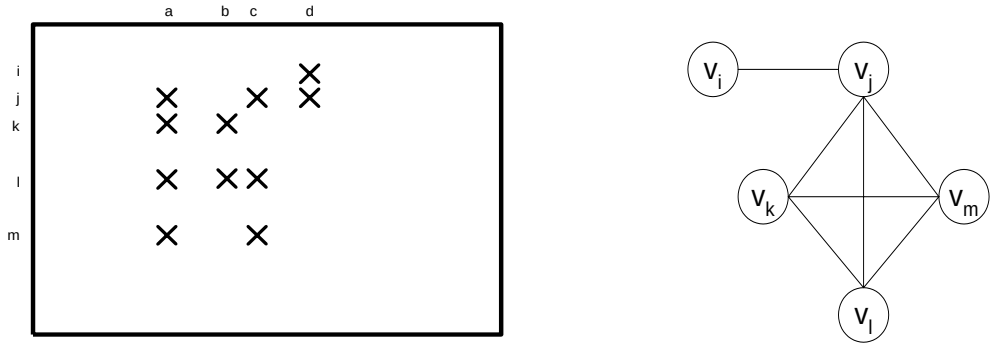

Figure 5.5: A sample partial matrix and NIG representation of associated hypergraph to illustrate the clique discarding algorithm

of hypergraph in the solution of GPVS problem through hypergraph partitioning methods. Figure 5.4 displays the proposed algorithm. The algorithm works as follows; nodes of hypergraphs are visited in the non-increasing degree order. If the currently visited node u_i in $\mathcal{H}$ is not marked for deletion yet, we check the degree of vertex v_j corresponding to the incident net n_j of u_i. If the degree of vertex v_j in $\mathcal{G}$ is equal to the degree of u_i minus one, this means that u_i is the node that induced the largest clique which includes v_j. In other words, all other nodes connected to n_j in $\mathcal{H}$ will induce cliques whose edges are subset of the clique edges induced by u_i. So we can safely delete all other nodes connected to n_j in $\mathcal{H}$. Consider the example sketched in Figure 5.5. Our algorithm works as follows. The columns of the sample matrix are visited in the order a, c, b, d. For the first column a with 4 nonzeros, we check the degree of vertices v_j, v_k, v_l, v_m. Since degree of v_j is 4 in $\mathcal{G}$ (not equal to $deg_{\mathcal{H}}(u_a) - 1 = 4 - 1 = 3$) we just skip it. The degree of v_k is 3 in $\mathcal{G}$, therefore all the nodes, except node u_a, incident to net n_j will be marked for deletion. Hence, u_b is marked for deletion. Since the degree of v_l is also 3 in $\mathcal{G}$, this cause to mark u_c for deletion. Although the degree of v_m is also 3 in $\mathcal{G}$, since the only node u_c incident to n_m already marked for deletion, no extra vertex is marked. In the outer-most loop, we will skip nodes u_c and u_b since they are marked for deletion. For node u_d, no other node will be marked. Although degree of v_i is 1 which is equal to $deg_{\mathcal{H}}(u_d) - 1 = 2 - 1 = 1$, there is no other node in the hypergraph (except u_d) which is connected to n_i. Since the degree of v_j is not 1, it will be skipped. At the end of the execution nodes v_b and v_c is marked for deletion, so we can safely discard those nodes in the hypergraph.

5.3.2 Sparsening

Here, we propose a second hypergraph simplification algorithm for solving GPVS problem through hypergraph partitioning. Recall from Section 2.3 that, two vertices of NIG are adjacent if the respective nets share at least one pin in the hypergraph. However, if they share more than one pin, only one of them suffices in our application, because our goal is to achieve a GPVS partitioning on NIG through hypergraph partitioning. Here we present a simple yet effective algorithm for pin deletion based sparsening. We need to first identify the pins that can be deleted. Let $W[i,j]$ denotes the number of common pins of nets n_i and n_j. We have the following lemma for pin deletion:

Lemma 1 *For each* $u \in pins[n_i]$, *pin* (n_i, u) *can be deleted if* $W[i,j] > 1$ *for all* $n_j \in nets[u] - \{n_i\}$.

Obviously, $W[i,j]$ must be greater than or equal to 1, since u is common a pin of both n_i and n_j. If $W[i,j] = 1$ for a net n_j, this means that u is the only common pin between n_i and n_j, so we cannot delete it, since we loose edge $\{v_i, v_j\}$ in NIG. If $W[i,j] > 1$ for all n_j, this means that n_i and n_j share more than one pin, including u, so we can safely delete pin (n_i, u). Consider the example given in Figure 5.6. In this figure NIG edges are labeled both with $W[i,j]$ values and the set of common nodes for the sake of simplicity of presentation. Consider the possible deletion of pins of net n_1. Pin (n_1, u_1) cannot be deleted since $W[1,3] = 1$, that is u_1 is the only common node in the pin lists of nets n_1 and n_3. Pin (n_1, u_2) can be deleted since both $W[1,2] = 2$ and $W[1,4] = 4$. Pin (n_1, u_3) can also be deleted since $W[1,4] = 2$. However, pins (n_1, u_2) and (n_1, u_3) cannot be deleted together, since deleting both of them makes $W[1,4] = 0$.

The proposed pin deletion-based sparsening algorithm is displayed in Figure 5.7. The algorithm does not require the NIG $\mathcal{G}$ as input. Edge weight values $W[i,j]$ of $\mathcal{G}$ are recomputed for each net n_j. When pin (n_i, u) is identified for deletion, since pin (n_i, u) stored both in the net list of node u and in the pin list of net n_i, we delete both n_i from $nets[u]$ and u from $pins[n_i]$ to effectively delete (n_i, u) in $\mathcal{H}$. Note that when pin (n_i, u) is deleted, weights of edges between n_i

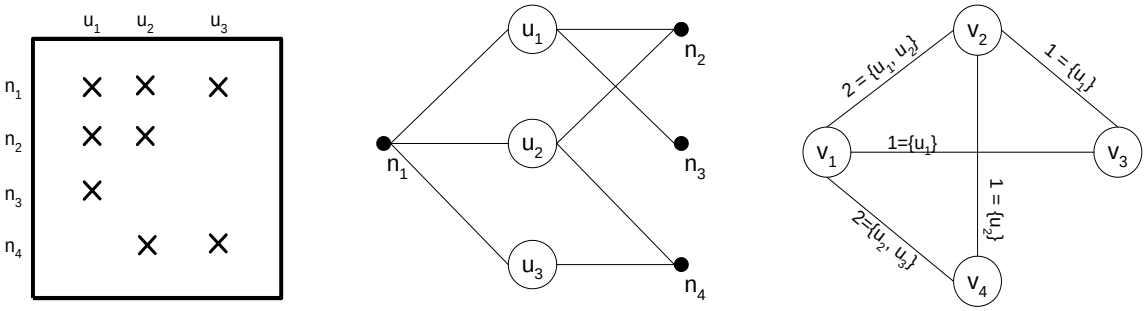

Figure 5.6: A sample matrix, its associated row-net hypergraph and NIG representation of the associated hypergraph

and n_j are decremented by 1, for each $n_j \in nets[u]$ to reflect the pin deletion in the edge weights of NIG.

5.4 Generalization

Until here, we have assumed that for ordering Z we have also given its decomposition $Z = AA^T$. However, in most of the applications this is not the case, that is, A is usually unknown. Here, we propose a simple yet effective decomposition of symmetric matrices for hypergraph partitioning-based nested dissection. Let $\mathcal{G}$ be the standard graph model representation of matrix Z. Our aim is to find a matrix A such that $AA^T = Z$. In graph theoretical view, we are trying to find a hypergraph $\mathcal{H}$ such that its NIG is $\mathcal{G}$. Obviously net set of the target hypergraph $\mathcal{H}$ is already identified by the definition of NIG. That is, there must be a net n_i in hypergraph $\mathcal{H}$ corresponding to each vertex v_i in $\mathcal{G}$. The node set of $\mathcal{H}$ is defined as follows. There is a node u_{ij} in $\mathcal{H}$ corresponding to edge $e_{ij} \in \mathcal{E}$ with the net list $nets[u_{ij}] = \{n_i, n_j\}$. As mentioned earlier, during the construction of NIG $\mathcal{G}$ from a hypergraph $\mathcal{H}$, each node of $\mathcal{H}$ induces a clique among the vertices of $\mathcal{G}$ that correspond to nets incident to that node in $\mathcal{H}$. It is clear that, with the proposed decomposition, each node of $\mathcal{H}$ induces distinct 2-cliques, therefore the proposed decomposition is referred to here as *2-clique* decomposition.

In matrix theoretical view, matrix A is the edge-incidence matrix of NIG $\mathcal{G}$. That is, each row of matrix A corresponds to a vertex in $\mathcal{G}$. Each column of matrix A corresponds to an edge in $\mathcal{G}$, such that there are exactly two nonzeros

```
initialize W[j] ← 0 for i = 1, ..., |𝒩|
for each net n_i ∈ 𝒩 do
    for each node u ∈ pins[n_i] do
        for each n_j ∈ nets[u] do
            W[j] ← W[j] + 1
    for each node u ∈ pins[n_i] do
        flag ← TRUE
        for each n_j ∈ nets[u] do
            if n_j ≠ n_i and W[j] = 1 then
                flag ← FALSE
                break
        if flag = TRUE then
            nets[u] ← nets[u] − {n_i}
            pins[n_j] ← pins[n_j] − {u}
            for each n_j ∈ nets[u] do
                W[j] ← W[j] − 1
    for each node u ∈ pins[n_i] do
        for each n_j ∈ nets[u] do
            W[j] ← 0
```

Figure 5.7: Hypergraph Sparsening Algorithm for $\mathcal{H} = (\mathcal{U}, \mathcal{N})$

in each column representing the two end points of the edge. Note that, the hypergraph mentioned in the previous paragraph is the row-net representation of the edge-incident matrix A.

5.5 Extending Supernode Concept

Supernode concept has been widely used in MD ordering [74, 57]. In matrix theoretical view, supernodes correspond to the columns with identical sparsity pattern. In graph theoretical view, a supernode corresponds to a clique of vertices with identical adjacency structure. The nice property of a supernode is that all nodes in the supernode can be eliminated in one step. In MD-based ordering algorithms, supernodes are identified and the ordering algorithm works on the compressed graph obtained by merging vertices constituting the supernodes. The supernode concept has also been exploited in a dynamic manner by identifying supernodes formed during the elimination.

The supernode concept has also been exploited in nested dissection based ordering algorithms as follows. If any constituent vertex of a supernode belongs to vertex separator $\mathcal{V}_S$ in $\Pi_{GPVS} = \{\mathcal{V}_1, \mathcal{V}_2; \mathcal{V}_S\}$, then all other constituent vertices of the supernode belong to separator $\mathcal{V}_S$. In a similar manner, if any constituent vertex of a supernode belongs to $\mathcal{V}_1$ ($\mathcal{V}_2$), then all other constituent vertices of the supernode belong to $\mathcal{V}_1$ ($\mathcal{V}_2$). So nested bisection based algorithms can also work on compressed graphs. In this work, we extend the supernode concept for nested dissection based ordering. We claim that for nested dissection based ordering, the constituent vertices of a supernode need not to be connected. That is, a set of disconnected (non-adjacent) vertices with identical adjacency structure can also be merged to form supernodes. The former and latter types of supernodes will be referred to here as *connected* (conventional) and *disconnected* supernodes.

The algorithm [4] used for identifying connected supernodes computes a hash value for each vertex v_i as

$$hash(v_i) = i + \sum_{v_j \in Adj(v_i)} j. \tag{5.1}$$

These hash values are exploited to quickly identify connected supernodes. It is obvious that, if two vertices have different hash values, then they have different adjacency structure. The algorithm first sorts the vertices in $\mathcal{G}$ by hash value. The sorted list is divided into subsets so that each subset contains the vertices with identical hash value. Then only adjacency structures of the vertices in these subsets are compared. We made some enhancements in the implementation of this algorithm as follows. The degree of the vertices are used as a secondary key in the sorting to reduce the number of vertices with identical key. The sorted list is again divided into subsets so that each subset contains the vertices with identical hash value and degree. For each vertex v_i in this subset (if it is not selected as a constituent of a supernode yet), adjacent vertices are marked with i in a mark array (i.e., $mark[v_j] = i$ for $v_j \in Adj(v_i)$). Then only the adjacency structure of the vertices, in the same subset, adjacent to v_i are compared with the adjacency structure of v_i. Note that we can skip the adjacency structure comparison for a vertex v_j if it is not adjacent to v_i (i.e., $mark[v_j] \neq i$). During the adjacency structure comparison for a vertex v_j, we check if all its adjacent vertices are also marked with i.

The algorithm for identifying disconnected supernodes works as follows. The hash values are computed as

$$hash(v_i) = \sum_{v_j \in Adj(v_i)} j. \tag{5.2}$$

Vertices of graph $\mathcal{G}$ are sorted by hash value and degree. The sorted list is again divided into subset containing identical key values. For each vertex v_i in each subset (if it is not selected as a constituent of a supernode yet), adjacent vertices are marked with i in a mark array (i.e., $mark[v_j] = i$ for $v_j \in Adj(v_i)$). Then only the adjacency structure of the vertices, in the same subset, non-adjacent to v_i are compared with the adjacency structure of v_i. That is, in this algorithm, we can skip the adjacency comparison of a vertex v_j if it is adjacent to v_i (i.e.,$mark[v_j] = i$).

5.6 Experimental Results

We have tested the proposed hypergraph partitioning-based nested dissection method on the ordering of various realistic sparse test matrices arising in different application domains [24, 16, 18, 23]. Table 5.1 illustrates the properties of the test matrices. In this table, M denotes the number of rows/columns of matrix Z, and NZ denotes the total number of nonzeros. For the matrices arising from LP problems, number of columns N and total number of nonzeros NZ are also listed for matrix A, where $Z = AA^T$. The number of rows of A is equal to the number of rows/columns of Z. This table also displays the Multiple Minimum Degree [57] (MMD) ordering results in terms of operation count (shown as "OPC") and total number of nonzeros after factorization (shown as "NZF"). We have used MMD implementation of SMOOTH [7] with the parameters: compressFlag=6 for compression before elimination and after each elimination step, prioType=1 for exact external degree for each vertex, stepType=1 for independent set elimination.

Table 5.2 displays the the number of connected and disconnected supernodes identified by the algorithms described in Section 5.5, as percent of M. For the matrices arising from LP problems, the clique discarding and sparsening algorithms presented in Section 5.3.1 and Section 5.3.2 are also applied. The number of discarded/deleted columns and nonzeros of A are also display in this table as percents of N and NZ, respectively. As seen in Table 5.2, general matrices have considerable amount of connected supernodes (approximately 26% on the average), however disconnected supernodes are very rare (less than 1% on the average). In LP problems, percent of disconnected supernodes is 3.64 and percent of connected supernodes is 5.48, on the average. Approximately 2% of the columns and nonzeros of A is identified as redundant by clique discarding algorithm, on the average. As seen Table 5.2, considerable amount (20% on the average) of nonzeros (pins) of A ($\mathcal{H}$) are deleted by the sparsening algorithm.

The nested dissection based algorithms usually work in an incomplete manner. That is, nested dissection is applied until the parts are fairly small, since the MD algorithm is quite effective for modest-size graphs. The subgraphs induced by the parts correspond to the standard graph representation of the decoupled block-diagonal submatrices. There are various possible ordering schemes for the

parts and separators after a $K = 2^{\ell}$-way nested dissection. Let $Z_1, Z_2, \ldots, Z_K$ be decoupled parts of symmetric matrix Z by separators $S_1, S_2, \ldots, S_\ell$ at each level of recursion. Figure 5.8 illustrates a sample for this decoupling process for $K = 4$. The difference of the ordering schemes lies in which ordering method is used to order vertices in the decoupled parts, and how the vertices in separators are ordered. Four possible ordering schemes as follows:

ND-MD all decoupled block-diagonal submatrices are ordered first by MD, then all separators are ordered in depth-first order, i.e., S_ℓ is ordered just after the orderings of $Z_1, Z_2, \ldots, Z_K$, then $S_{\ell-1}$ is ordered and so on, such that S_1 is ordered last.

ND-CMD all decoupled block-diagonal submatrices are ordered first by constraint minimum degree [59] (CMD), then all separators are ordered in depth-first order.

multisection-MD all decoupled block-diagonal submatrices are ordered first by MD, then all separators are ordered together.

multisection-CMD all decoupled block-diagonal submatrices are ordered first by CMD, then all separators are ordered together.

With this classification, ordering code of MeTiS [44] falls into the class ND-MD, and BEND [38] falls into the class ND-CMD. In their recent work [9], Ashcraft and Liu states that CMD [59] algorithm produces better orderings in nested dissection and multisection ordering. The results presented in their work also show that multisection generates better orderings than nested dissection. Hence, their ordering code, we call it SMOOTH as the name of whole package is SMOOTH [7], falls into class multisection-CMD.

MSMD object in the SMOOTH software package [7], is a piece-of-art ordering object. It contains both CMD and MMD features combined in a brilliant way. The idea is as follows, MSMD orders the vertices by *stages*, i.e., vertices in stage k will be ordered before vertices in stage $k + 1$. Inside the stages, it basically does MMD ordering, however, selection criteria can also be changed, i.e., instead of using actual degree, approximate degree can be used. With this code, development of "ND-CMD" and "multisection-CMD" ordering codes are simple tasks.

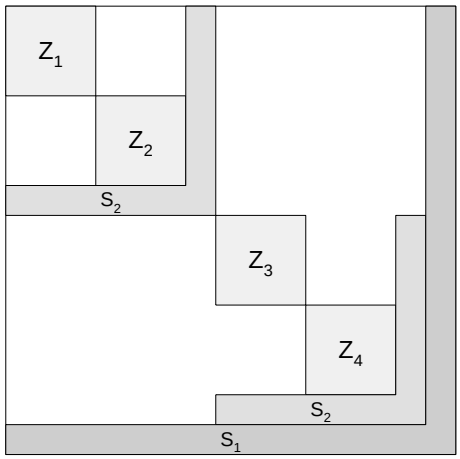

Figure 5.8: 4-way decoupled matrix Z using recursive dissection.

Therefore, we have used MSMD object of SMOOTH [7] in the development of our hypergraph partitioning-based nested dissection ordering tool oPaToH. We have incorporated, both ND-CMD and multisection-CMD schemes. In the current implementation oPaToH-ND stands for ordering code of PaToH which uses ND-CMD, and oPaToH-MS stands for multisection version.

The average ordering performance of the various tools are displayed in Tables 5.3-5.5 relative to MMD. The results of GP-based nested dissection ordering tools onmetis and oemetis, graph partitioning-based multisection ordering tool SMOOTH are displayed in these tables. The proposed HP-based multisection and nested dissection ordering results using PaToH are also displayed in these tables. In Tables 5.3-5.5, "2-Clique oPaToH" denotes the hypergraph partitioning-based ordering of matrix Z using the 2-Clique decomposition described in Section 5.4, whereas "oPaToH using A" denotes the hypergraph partitioning-based ordering of matrix Z using the given constraint matrix A for LP problems. Hence, no result is displayed in those columns for general matrices. For each problem, ordering tools were run 50 times starting from different random seeds and the average results are displayed in the tables.

Table 5.3 displays the average ordering performance of the tools in terms of operation count. For general matrices, best ordering results are obtain by SMOOTH, on the average, SMOOTH produces 27% better orderings than MMD. The proposed HP-based multisection ordering (oPaToH-MS) produces the second

best solutions, on the average. oPaToH-MS produces consistently better orderings than MMD for each problem, and produces 25% better orderings than MMD on the average. In the ordering of matrices arising from LP problems, ND version of PaToH produces better orderings than MS based version, achieving 45% better orderings than MMD heuristic. For these matrices, oPaToH-ND using A produces 17% and 43% better orderings than onmetis and SMOOTH. It is interesting to note that indirect GPVS based nested dissection tool oemetis produces very inconsistent results.

Average ordering performance of the tools in terms of nonzero counts in the factor matrices are presented in Table 5.4. For general matrices, there is no clear winner. All the nested dissection and multisection based tools perform equally well by producing approximately 10% less nonzero than MMD. In LP problems, again oPaToH-ND produces best results in terms of nonzero counts. oPaToH-ND produces approximately 22% less nonzeros than MMD, and 9% less nonzeros than onmetis. For these problems, SMOOTH produces nearly the same amount of nonzeros with MMD.

Table 5.5 displays the average execution times of the tools relative to MMD ordering. In this table, a ratio smaller than 1.0 indicates that the respective tool is faster than the MMD ordering. The fastest tool is the direct GPVS based ordering code onmetis of MeTiS. Although it is only 5% faster than MMD in the ordering of general matrices, it runs approximately 3.6 times faster than MMD in the ordering of matrices arising from LP problems. SMOOTH runs 4.7 and 1.8 times slower than MMD in the ordering of general matrices and matrices arising from LP problems. 2-clique decomposition yields the slowest ordering. This is an expected result, since the running time of the hypergraph partitioning is proportional to the number of pins and nodes, and the 2-clique model generates a hypergraph with NZ nodes and $2NZ$ pins. However, the proposed HP-based ordering methods is only 21% slower than MMD ordering while producing superior results than MMD.

Table 5.1: Properties of test matrices and results of MMD orderings

	matrix Z		matrix A where $Z = AA^T$		MMD	
name	M	NZ	N	NZ	NZF	OPC
General Matrices						
BCSSTK23	3,134	45,178	-	-	461,697	1.46E+08
BCSSTK21	3,600	26,600	-	-	116,083	6.75E+06
BCSSTK15	3,948	117,816	-	-	653,831	1.68E+08
3elt	4,720	32,164	-	-	92,188	3.11E+06
BCSSTK16	4,884	290,378	-	-	741,200	1.45E+08
BCSSTK17	10,974	428,650	-	-	1,136,428	1.99E+08
BCSSTK18	11,948	149,090	-	-	642,176	1.30E+08
BCSSTK25	15,439	252,241	-	-	1,515,540	3.31E+08
BCSSTK32	44,609	2,014,701	-	-	5,146,621	1.05E+09
brack2	62,631	795,749	-	-	7,482,073	3.22E+09
598a	110,971	1,594,839	-	-	45,116,662	5.87E+10
crystk01	4,875	315,891	-	-	1,094,672	3.46E+08
lshp3025	3,025	20,833	-	-	75,332	3.46E+06
lshp3466	3,466	23,896	-	-	89,551	4.39E+06
mplate	5,962	142,190	-	-	2,172,166	1.53E+09
nasa4704	4,704	104,756	-	-	269,427	3.24E+07
pwt	36,519	326,107	-	-	1,810,221	2.38E+08
s1rmq4m1	5,489	281,111	-	-	658,508	1.15E+08
s2rmq4m1	5,489	281,111	-	-	658,508	1.15E+08
s3rmq4m1	5,489	281,111	-	-	658,508	1.15E+08
shuttle-eddy	10,429	103,599	-	-	389,810	2.61E+07
skirt	12,598	196,520	-	-	494,045	3.63E+07
vibrobox	12,328	342,828	-	-	2,119,728	9.16E+08
LP Problems						
NL	7,039	105,089	9,718	41,428	282,929	3.78E+07
CQ9	9,278	221,590	13,778	88,897	451,108	5.74E+07
GE	10,099	112,129	11,098	39,554	294,188	3.47E+07
CO9	10,789	249,205	14,851	101,578	499,511	6.40E+07
fome12	24,284	329,068	48,920	142,528	6,314,673	5.19E+09
pltexpA4-6	26,894	269,736	70,364	143,059	2,329,048	1.10E+09
world	34,506	582,064	32,734	164,470	1,789,127	2.77E+08
mod2	34,774	604,910	31,728	165,129	1,823,079	2.72E+08
lpl1	39,951	541,217	125,000	381,259	3,146,595	1.21E+09
fxm3-16	41,340	765,526	64,162	370,839	637,294	1.97E+07
cre-b	9,648	398,806	72,447	256,095	954,754	3.82E+08
cre-d	8,926	372,266	69,980	242,646	870,409	3.01E+08
delf036	3,170	33,508	5,459	14,202	50,025	1.78E+06
dfl001	6,071	82,267	12,230	35,632	1,599,555	1.34E+09
ex3sta1	17,443	679,857	8,156	59,419	25,649,479	7.28E+10
ken-07	2,426	14,382	3,602	8,404	15,553	2.17E+05
ken-11	14,694	82,454	21,349	49,058	134,394	4.18E+06
ken-13	28,632	161,804	42,659	97,246	355,934	1.71E+07
large036	4,282	50,696	6,822	18,840	75,363	3.17E+06
model10	4,400	293,260	15,447	149,000	516,068	1.14E+08
pds-02	2,953	23,281	7,535	16,390	40,920	1.73E+06
pds-06	9,881	88,003	28,655	62,524	573,506	2.05E+08
pds-10	16,558	149,658	48,763	106,436	1,618,218	1.05E+09
pds-20	33,874	320,196	105,728	230,200	6,889,030	9.22E+09
rlfprim	58,866	9,119,596	8,052	265,927	301,830,670	2.56E+12

Table 5.2: Compression and sparsening results

	Supernodes					
	connected	disconnected	Clique Discarding		Sparsening	
name	%M	%M	%N	%NZ	%N	%NZ
General Matrices						
BCSSTK23	6.51	0.00	-	-	-	-
BCSSTK21	0.00	0.00	-	-	-	-
BCSSTK15	0.00	0.13	-	-	-	-
3elt	0.00	0.00	-	-	-	-
BCSSTK16	63.60	1.86	-	-	-	-
BCSSTK17	52.44	4.79	-	-	-	-
BCSSTK18	8.55	6.65	-	-	-	-
BCSSTK25	14.61	0.05	-	-	-	-
BCSSTK32	66.78	0.07	-	-	-	-
brack2	0.00	0.00	-	-	-	-
598a	0.00	0.00	-	-	-	-
crystk01	64.78	0.10	-	-	-	-
lshp3025	0.00	0.00	-	-	-	-
lshp3466	0.00	0.00	-	-	-	-
mplate	5.67	0.00	-	-	-	-
nasa4704	50.51	0.62	-	-	-	-
pwt	0.01	0.16	-	-	-	-
s1rmq4m1	82.49	0.00	-	-	-	-
s2rmq4m1	82.49	0.00	-	-	-	-
s3rmq4m1	82.49	0.00	-	-	-	-
shuttle-eddy	0.63	0.00	-	-	-	-
skirt	15.46	0.02	-	-	-	-
vibrobox	0.10	0.00	-	-	-	-
average	25.96	0.63	-	-	-	-
LP Problems						
NL	0.38	0.75	1.76	0.62	17.03	13.85
CQ9	4.67	1.46	5.12	0.82	12.91	31.40
GE	12.18	2.11	1.39	2.47	18.16	31.25
CO9	6.83	1.21	7.75	1.44	11.90	34.47
fome12	0.00	0.96	0.01	0.01	14.93	7.88
pltexpA4-6	0.00	5.78	0.00	0.00	56.24	43.96
world	8.67	1.28	0.62	0.26	7.76	11.43
mod2	9.61	1.29	0.60	0.24	6.45	12.27
lpl1	0.00	6.34	0.04	0.03	53.84	52.99
fxm3-16	14.51	9.95	9.59	24.49	37.41	65.01
cre-b	0.07	25.01	0.00	0.00	2.26	12.47
cre-d	0.10	27.48	0.00	0.00	2.47	12.81
delf036	11.29	0.00	7.07	5.08	26.93	37.90
dfl001	0.00	0.96	0.01	0.01	14.93	7.88
ex3sta1	26.83	0.00	0.01	0.00	12.75	21.21
ken-07	0.00	0.00	0.00	0.00	2.05	1.46
ken-11	0.00	3.29	0.00	0.00	3.65	3.07
ken-13	0.59	1.77	0.00	0.00	2.15	1.81
large036	18.52	0.00	7.07	6.38	26.93	40.90
model10	21.34	1.09	1.30	0.16	7.30	55.18
pds-02	0.00	0.07	0.00	0.00	0.15	0.20
pds-06	0.00	0.04	0.00	0.00	0.08	0.11
pds-10	0.00	0.04	0.00	0.00	0.07	0.09
pds-20	0.00	0.25	0.00	0.00	0.05	0.07
rlfprim	1.34	0.00	0.00	0.00	0.00	0.30

Table 5.3: Operation counts of various methods relative to MMD

name	onmetis	oemetis	SMOOTH	2-Clique oPaToH		oPaToH using A	
				MS	ND	MS	ND
General Matrices							
BCSSTK23	0.65	0.69	0.65	0.73	0.71	-	-
BCSSTK21	0.94	1.09	0.73	0.93	0.94	-	-
BCSSTK15	0.53	0.53	0.61	0.71	0.70	-	-
3elt	0.93	0.97	0.84	0.82	0.85	-	-
BCSSTK16	1.02	1.02	0.84	0.92	1.11	-	-
BCSSTK17	0.97	0.94	0.68	0.76	0.83	-	-
BCSSTK18	0.68	0.70	0.71	0.72	0.77	-	-
BCSSTK25	1.14	1.16	0.77	0.91	1.21	-	-
BCSSTK32	1.27	1.51	0.83	0.84	0.90	-	-
brack2	0.58	0.60	0.65	0.67	0.70	-	-
598a	0.34	0.34	0.48	0.53	0.48	-	-
crystk01	0.79	0.63	0.65	0.70	0.78	-	-
lshp3025	0.92	0.92	0.77	0.78	0.81	-	-
lshp3466	0.89	0.91	0.76	0.77	0.80	-	-
mplate	0.39	0.35	0.47	0.49	0.45	-	-
nasa4704	1.08	1.07	0.98	0.85	0.79	-	-
pwt	0.46	0.46	0.52	0.54	0.53	-	-
s1rmq4m1	0.94	0.76	0.82	0.80	0.82	-	-
s2rmq4m1	0.95	0.76	0.82	0.82	0.89	-	-
s3rmq4m1	0.95	0.76	0.82	0.82	0.84	-	-
shuttle-eddy	0.84	0.86	0.63	0.68	0.83	-	-
skirt	0.86	0.87	0.67	0.79	0.80	-	-
vibrobox	1.06	1.06	1.88	0.95	0.84	-	-
geomean	0.79	0.78	0.73	0.75	0.78	-	-
LP Problems							
NL	1.16	20.16	1.02	0.94	0.97	0.95	0.97
CQ9	0.79	28.35	0.74	0.71	0.71	0.65	0.67
GE	0.68	0.90	0.61	0.82	0.81	0.66	0.64
CO9	0.88	33.48	0.76	0.75	0.77	0.74	0.77
fome12	0.58	0.93	2.01	0.46	0.46	0.46	0.46
pltexpA4-6	0.16	0.24	0.38	0.13	0.13	0.09	0.10
world	1.48	2.78	1.66	0.92	0.92	0.82	0.81
mod2	1.56	2.93	1.57	0.91	0.90	0.80	0.81
lpl1	1.57	12.33	1.25	0.96	0.94	0.94	0.96
fxm3-16	1.41	1.59	1.29	0.99	0.99	0.97	0.97
cre-b	0.58	0.67	0.78	0.49	0.56	0.51	0.57
cre-d	0.56	0.63	1.17	0.56	0.56	0.57	0.60
delf036	1.02	1.26	0.92	0.79	0.95	0.79	0.92
dfl001	0.59	0.91	1.89	0.45	0.40	0.44	0.39
ex3sta1	0.11	0.11	0.18	0.16	0.14	0.14	0.11
ken-07	1.06	18.16	1.00	0.95	0.95	0.95	0.95
ken-11	1.00	113.04	0.98	0.97	0.97	0.97	0.97
ken-13	1.07	307.63	1.09	0.99	0.98	0.99	0.99
large036	1.05	1.08	1.10	0.77	0.92	0.76	0.92
model10	0.55	0.50	0.73	0.54	0.49	0.70	0.51
pds-02	1.21	1.25	1.51	0.95	0.99	0.91	0.89
pds-06	0.33	0.48	0.94	0.54	0.44	0.49	0.37
pds-10	0.35	0.67	1.24	0.67	0.49	0.65	0.39
pds-20	0.41	0.71	0.89	0.79	0.70	0.72	0.48
rlfprim	0.14	0.14	0.65	-	-	0.17	0.13
geomean	0.66	2.33	0.95	0.65	0.64	0.59	0.55

Table 5.4: Nonzero counts of various methods relative to MMD

name	onmetis	oemetis	SMOOTH	2-Clique oPaToH		oPaToH using A	
				MS	ND	MS	ND
General Matrices							
BCSSTK23	0.85	0.87	0.83	0.89	0.89	-	-
BCSSTK21	0.96	1.04	0.88	0.99	0.99	-	-
BCSSTK15	0.76	0.76	0.80	0.85	0.85	-	-
3elt	1.00	1.02	0.95	0.94	0.95	-	-
BCSSTK16	1.00	1.01	0.92	0.96	1.03	-	-
BCSSTK17	1.00	0.99	0.87	0.91	0.93	-	-
BCSSTK18	0.92	0.94	0.90	0.89	0.91	-	-
BCSSTK25	1.06	1.08	0.89	0.95	1.04	-	-
BCSSTK32	1.12	1.17	0.94	0.95	0.96	-	-
brack2	0.81	0.82	0.84	0.84	0.85	-	-
598a	0.60	0.60	0.69	0.72	0.69	-	-
crystk01	0.92	0.82	0.83	0.86	0.90	-	-
lshp3025	0.99	0.99	0.92	0.92	0.93	-	-
lshp3466	0.98	0.98	0.91	0.92	0.92	-	-
mplate	0.65	0.62	0.70	0.71	0.69	-	-
nasa4704	1.09	1.08	1.01	0.94	0.92	-	-
pwt	0.76	0.76	0.77	0.78	0.78	-	-
s1rmq4m1	0.99	0.92	0.92	0.92	0.93	-	-
s2rmq4m1	0.99	0.92	0.92	0.93	0.95	-	-
s3rmq4m1	0.99	0.92	0.92	0.93	0.94	-	-
shuttle-eddy	0.93	0.93	0.83	0.85	0.91	-	-
skirt	0.98	0.98	0.86	0.92	0.93	-	-
vibrobox	1.01	1.03	1.29	0.91	0.87	-	-
geomean	0.92	0.91	0.88	0.89	0.90	-	-
LP Problems							
NL	1.09	3.83	1.03	0.98	0.99	0.98	0.99
CQ9	0.94	4.05	0.92	0.89	0.89	0.87	0.88
GE	0.94	1.02	0.88	0.94	0.94	0.89	0.88
CO9	0.99	4.51	0.92	0.91	0.92	0.90	0.91
fome12	0.81	1.04	1.50	0.71	0.71	0.70	0.71
pltexpA4-6	0.55	0.67	0.96	0.46	0.46	0.42	0.43
world	1.20	1.63	1.31	0.97	0.97	0.93	0.92
mod2	1.22	1.67	1.28	0.97	0.96	0.93	0.92
lpl1	1.24	3.73	1.11	0.98	0.97	0.97	0.97
fxm3-16	1.12	1.17	1.06	1.00	1.00	1.00	1.00
cre-b	0.83	0.87	0.93	0.76	0.79	0.77	0.80
cre-d	0.82	0.84	1.09	0.79	0.79	0.79	0.81
delf036	1.04	1.13	0.99	0.94	0.98	0.94	0.97
dfl001	0.82	1.03	1.46	0.70	0.67	0.69	0.66
ex3sta1	0.31	0.32	0.38	0.36	0.34	0.33	0.31
ken-07	1.04	2.31	1.00	0.99	0.99	0.99	0.99
ken-11	1.02	5.38	1.02	0.99	0.99	0.99	0.99
ken-13	1.06	8.11	1.07	1.01	1.00	1.01	1.00
large036	1.06	1.08	1.05	0.94	0.98	0.93	0.97
model10	0.80	0.77	0.90	0.77	0.75	0.85	0.76
pds-02	1.09	1.10	1.14	0.98	1.00	0.97	0.97
pds-06	0.70	0.87	1.09	0.80	0.74	0.77	0.71
pds-10	0.72	0.97	1.24	0.83	0.74	0.82	0.70
pds-20	0.70	0.93	1.07	0.85	0.81	0.82	0.71
rlfprim	0.38	0.38	0.83	-	-	0.40	0.36
geomean	0.86	1.40	1.02	0.83	0.83	0.80	0.78

Table 5.5: Ordering runtimes of various methods relative to MMD

name	onmetis	oemetis	SMOOTH	2-Clique oPaToH		oPaToH using A	
				MS	ND	MS	ND
General Matrices							
BCSSTK23	0.63	0.50	1.80	4.14	4.10	-	-
BCSSTK21	0.77	0.65	1.73	4.90	4.84	-	-
BCSSTK15	1.07	0.86	2.76	14.76	14.84	-	-
3elt	1.01	0.98	3.03	6.60	6.64	-	-
BCSSTK16	1.12	4.56	14.74	8.75	8.70	-	-
BCSSTK17	1.67	3.70	9.18	11.13	11.13	-	-
BCSSTK18	1.00	0.82	3.28	8.01	8.03	-	-
BCSSTK25	1.16	0.96	2.82	7.53	7.53	-	-
BCSSTK32	1.66	5.55	11.67	10.42	10.46	-	-
brack2	0.91	0.84	2.49	8.72	8.67	-	-
598a	0.74	0.69	2.39	8.43	8.45	-	-
crystk01	1.04	4.63	16.16	7.91	7.92	-	-
lshp3025	0.97	0.93	2.92	6.82	6.80	-	-
lshp3466	1.00	0.89	2.82	6.61	6.66	-	-
mplate	0.50	0.40	1.75	5.41	5.43	-	-
nasa4704	0.98	2.54	5.89	5.64	5.87	-	-
pwt	1.19	1.19	3.32	10.15	10.16	-	-
s1rmq4m1	0.76	8.01	18.90	2.52	2.55	-	-
s2rmq4m1	0.73	8.30	19.86	2.43	2.43	-	-
s3rmq4m1	0.70	8.07	18.96	2.49	2.52	-	-
shuttle-eddy	1.20	1.18	3.30	10.58	10.61	-	-
skirt	1.31	1.42	2.97	14.30	14.34	-	-
vibrobox	0.71	0.65	4.28	11.93	11.91	-	-
geomean	0.95	1.57	4.71	7.00	7.02	-	-
LP Problems							
NL	0.17	0.16	2.13	2.90	2.84	0.99	0.82
CQ9	0.14	0.14	1.78	5.24	4.70	1.18	0.87
GE	1.06	0.86	2.82	5.12	5.08	2.04	1.96
CO9	0.12	0.12	1.37	3.97	3.86	0.98	0.78
fome12	0.10	0.08	0.92	2.42	2.58	1.74	1.61
pltexpA4-6	1.05	0.83	1.86	4.72	4.72	2.09	2.00
world	0.24	0.21	1.35	3.78	3.72	0.92	0.73
mod2	0.30	0.25	1.57	4.21	4.19	0.92	0.81
lpl1	0.23	0.21	1.56	2.21	2.21	0.99	0.93
fxm3-16	1.83	1.90	2.96	14.77	15.05	2.61	2.51
cre-b	0.21	0.18	3.66	14.02	14.51	3.17	3.06
cre-d	0.15	0.14	2.62	9.26	9.21	2.62	2.56
delf036	1.26	1.11	3.23	7.28	7.27	2.62	2.67
dfl001	0.08	0.06	0.96	2.14	0.81	1.61	0.48
ex3sta1	0.36	0.45	11.09	4.43	4.47	0.75	0.71
ken-07	0.78	0.94	1.14	4.57	4.74	3.29	3.22
ken-11	0.72	0.84	1.89	6.49	6.12	3.20	3.20
ken-13	0.31	0.35	0.85	6.30	4.16	4.97	2.25
large036	1.13	1.26	3.72	7.67	7.70	2.52	2.55
model10	0.65	0.97	7.63	16.56	16.11	3.28	3.33
pds-02	0.80	0.62	2.73	3.56	3.59	3.29	3.37
pds-06	0.14	0.12	0.84	0.91	0.79	0.82	0.76
pds-10	0.05	0.04	0.38	0.47	0.32	0.42	0.30
pds-20	0.01	0.01	0.11	0.37	0.17	0.24	0.14
rlfprim	0.35	0.35	8.47	-	-	0.12	0.12
geomean	0.30	0.28	1.83	3.96	3.53	1.43	1.10

Chapter 6

PaToH: A Multilevel Hypergraph Partitioning Tool

We exploit the successful multilevel methodology [13, 35, 46] proposed and implemented for graph partitioning [36, 44] to develop a new multilevel hypergraph partitioning tool, called PaToH (PaToH: **Pa**rtitioning **To**ols for **H**ypergraphs).

The data structures used to store hypergraphs in PaToH mainly consist of the following arrays. The *NETLST* array stores the net lists of the vertices. The *PINLST* array stores the pin lists of the nets. The size of both arrays is equal to the total number of pins in the hypergraph. Two auxiliary index arrays *VTXS* and *NETS* of sizes $|\mathcal{V}|+1$ and $|\mathcal{N}|+1$ hold the starting indices of the net lists and pin lists of the vertices and nets in the *NETLST* and *PINLST* arrays, respectively. In sparse matrix storage terminology, this scheme corresponds to storing the given matrix both in *Compressed Sparse Row (CSR)* and *Compressed Sparse Column (CSC)* formats [50] without storing the numerical data. In the column-net model proposed for rowwise decomposition, the *VTXS* and *NETLST* arrays correspond to the CSR storage scheme, and the *NETS* and *PINLST* arrays correspond to the CSC storage scheme. This correspondence is dual in the row-net model proposed for columnwise decomposition.

The storage requirement of the proposed hypergraph models is as follows. For an $M \times M$ square matrix with Z off-diagonal nonzero entries, the hypergraph

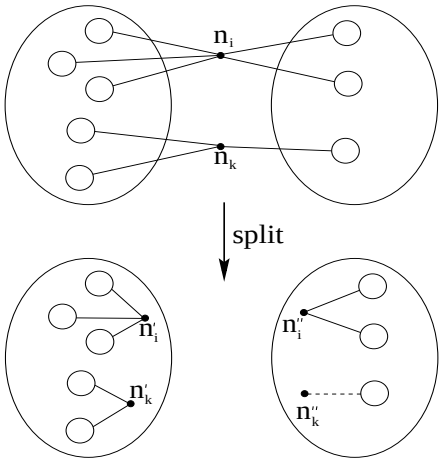

Figure 6.1: Cut-net splitting during recursive bisection.

models contain $|\mathcal{V}| = M$ vertices, $|\mathcal{N}| = M$ nets and $p = M + Z$ pins for both symmetric and unsymmetric matrices. Note that M pins comes from maintaining the diagonal entries of the matrix. Hence, the storage requirement of both hypergraph models is $S_{\mathcal{H}} = 5M + 2Z$ words, where $2M$ words come from index arrays *VTXS* and *NETS*, M words are required to store vertex weights, and $2(M + Z)$ words come from *NETLST* and *PINLST* arrays.

The K-way graph/hypergraph partitioning problem is usually solved by recursive bisection. In this scheme, first a 2-way partition of $\mathcal{G}/\mathcal{H}$ is obtained, and then this bipartition is further partitioned in a recursive manner. After $\lg_2 K$ phases, graph $\mathcal{G}/\mathcal{H}$ is partitioned into K parts. PaToH achieves K-way hypergraph partitioning by recursive bisection for any K value (i.e., K is not restricted to be a power of 2).

The connectivity cutsize metric given in (2.4.b) needs special attention in K-way hypergraph partitioning by recursive bisection. Note that the cutsize metrics given in (2.4.a) and (2.4.b) become equivalent in hypergraph bisection. Consider a bipartition $\mathcal{V}_{\mathcal{A}}$ and $\mathcal{V}_{\mathcal{B}}$ of $\mathcal{V}$ obtained after a bisection step. It is clear that $\mathcal{V}_{\mathcal{A}}$ and $\mathcal{V}_{\mathcal{B}}$ and the internal nets of parts $\mathcal{A}$ and $\mathcal{B}$ will become the vertex and net sets of $\mathcal{H}_{\mathcal{A}}$ and $\mathcal{H}_{\mathcal{B}}$, respectively, for the following recursive bisection steps. Note that each cut net of this bipartition already contributes 1 to the total cutsize of the final K-way partition to be obtained by further recursive bisections. However, the further recursive bisections of $\mathcal{V}_{\mathcal{A}}$ and $\mathcal{V}_{\mathcal{B}}$ may increase the connectivity of these cut nets. In parallel SpMxV view, while each cut net already incurs

the communication of a single word, these nets may induce additional communication because of the following recursive bisection steps. Hence, after every hypergraph bisection step, each cut net n_i is split into two pin-wise disjoint nets $n'_i = pins[n_i] \bigcap \mathcal{V}_A$ and $n''_i = pins[n_i] \bigcap \mathcal{V}_B$, and then these two nets are added to the net lists of $\mathcal{H}_A$ and $\mathcal{H}_B$ if $|n'_i| > 1$ and $|n''_i| > 1$, respectively. Note that the single-pin nets are discarded during the split operation since such nets cannot contribute to the cutsize in the following recursive bisection steps. Thus, the total cutsize according to (2.4.b) will become equal to the sum of the number of cut nets at every bisection step by using the above cut-net split method. Figure 6.1 illustrates two cut nets n_i and n_k in a bipartition, and their splits into nets n'_i, n''_i and n'_k, n''_k, respectively. Note that net n''_k becomes a single-pin net and it is discarded.

Similar to multilevel graph and hypergraph partitioning tools Chaco [36], MeTiS [44] and hMeTiS [47], the multilevel hypergraph bisection algorithm used in PaToH consists of 3 phases: coarsening, initial partitioning and uncoarsening. The following sections briefly summarize our multilevel bisection algorithm. Although PaToH works on weighted nets, we will assume unit cost nets both for the sake of simplicity of presentation and for the fact that all nets are assigned unit cost in the hypergraph representation of sparse matrices.

6.1 Coarsening Phase

In this phase, the given hypergraph $\mathcal{H} = \mathcal{H}_0 = (\mathcal{V}_0, \mathcal{N}_0)$ is coarsened into a sequence of smaller hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1)$, $\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2)$, ..., $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{N}_m)$ satisfying $|\mathcal{V}_0| > |\mathcal{V}_1| > |\mathcal{V}_2| > \ldots > |\mathcal{V}_m|$. This coarsening is achieved by coalescing disjoint subsets of vertices of hypergraph $\mathcal{H}_i$ into *multinodes* such that each multinode in $\mathcal{H}_i$ forms a single vertex of $\mathcal{H}_{i+1}$. The weight of each vertex of $\mathcal{H}_{i+1}$ becomes equal to the sum of its constituent vertices of the respective multinode in $\mathcal{H}_i$. The net set of each vertex of $\mathcal{H}_{i+1}$ becomes equal to the union of the net sets of the constituent vertices of the respective multinode in $\mathcal{H}_i$. Here, multiple pins of a net $n \in \mathcal{N}_i$ in a multinode cluster of $\mathcal{H}_i$ are contracted to a single pin of the respective net $n' \in \mathcal{N}_{i+1}$ of $\mathcal{H}_{i+1}$. Furthermore, the single-pin nets obtained during this contraction are discarded. Note that such single-pin nets correspond

to the internal nets of the clustering performed on $\mathcal{H}_i$. The coarsening phase terminates when the number of vertices in the coarsened hypergraph reduces below 100 (i.e. $|\mathcal{V}_m| \leq 100$).

Clustering approaches can be classified as *agglomerative* and *hierarchical*. In the agglomerative clustering, new clusters are formed one at a time, whereas in the hierarchical clustering several new clusters may be formed simultaneously. In PaToH, we have implemented both randomized matching–based hierarchical clustering and randomized hierarchic–agglomerative clustering. The former and latter approaches will be abbreviated as matching–based clustering and agglomerative clustering, respectively.

The matching-based clustering works as follows. Vertices of $\mathcal{H}_i$ are visited in a random order. If a vertex $u \in \mathcal{V}_i$ has not been matched yet, one of its unmatched *adjacent* vertices is selected according to a criterion. If such a vertex v exists, we merge the matched pair u and v into a cluster. If there is no unmatched adjacent vertex of u, then vertex u remains unmatched, i.e., u remains as a singleton cluster. Here, two vertices u and v are said to be adjacent if they share at least one net, i.e., $nets[u] \cap nets[v] \neq \emptyset$. The selection criterion used in PaToH for matching chooses a vertex v with the highest connectivity value N_{uv}. Here, connectivity $N_{uv} = |nets[u] \cap nets[v]|$ refers to the number of shared nets between u and v. This matching-based scheme is referred to here as *Heavy Connectivity Matching* (*HCM*).

The matching-based clustering allows the clustering of only pairs of vertices in a level. In order to enable the clustering of more than two vertices at each level, we have implemented a randomized agglomerative clustering approach. In this scheme, each vertex u is assumed to constitute a singleton cluster $C_u = \{u\}$ at the beginning of each coarsening level. Then, vertices are visited in a random order. If a vertex u has already been clustered (i.e. $|C_u| > 1$) it is not considered for being the source of a new clustering. However, an unclustered vertex u can choose to join a multinode cluster as well as a singleton cluster. That is, all adjacent vertices of an unclustered vertex u are considered for selection according to a criterion. The selection of a vertex v adjacent to u corresponds to including vertex u to cluster C_v to grow a new multinode cluster $C_u = C_v = C_v \cup \{u\}$. Note that no singleton cluster remains at the end of this process as far as there exists no isolated

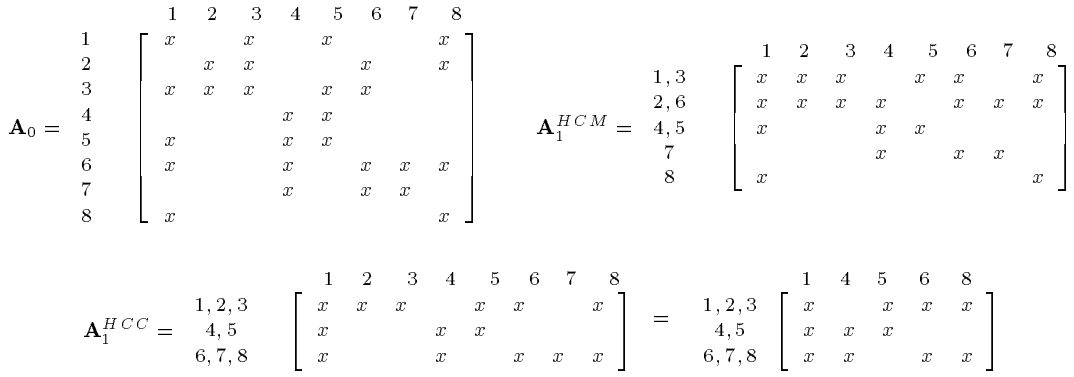

Figure 6.2: Matching-based clustering $\mathbf{A}_1^{HCM}$ and agglomerative clustering $\mathbf{A}_1^{HCC}$ of the rows of matrix $\mathbf{A}_0$.

vertex. The selection criterion used in PaToH for agglomerative clustering chooses a singleton or multinode cluster C_v with the highest $N_{u,C_v}/W_{u,C_v}$ value, where $N_{u,C_v} = |nets[u] \cap \bigcup_{x \in C_v} nets[x]|$ and W_{u,C_v} is the weight of the multinode cluster candidate $\{u\} \cup C_v$. The division of N_{u,C_v} by W_{u,C_v} is an effort for avoiding the polarization towards very large clusters. This agglomerative clustering scheme is referred to here as *Heavy Connectivity Clustering* (*HCC*).

The objective in both HCM and HCC is to find highly connected vertex clusters. Connectivity values N_{uv} and N_{u,C_v} used for selection serve this objective. Note that N_{uv} (N_{u,C_v}) also denotes the lower bound in the amount of decrease in the number of pins because of the pin contractions to be performed when u joins v (C_v). Recall that there might be additional decrease in the number of pins because of single-pin nets that may occur after clustering. Hence, the connectivity metric is also an effort towards minimizing the complexity of the following coarsening levels, partitioning phase and refinement phase since the size of a hypergraph is equal to the number of its pins.

In rowwise matrix decomposition context (i.e. column-net model), the connectivity metric corresponds to the number of common column indices between two rows or row groups. Hence, both HCM and HCC try to combine rows or row groups with similar sparsity patterns. This in turn corresponds to combining rows or row groups which need similar sets of **x**-vector components in the pre-communication scheme. A dual discussion holds for the row-net model. Figure 6

illustrates a single level coarsening of an 8×8 sample matrix $\mathbf{A}_0$ in the column-net model using HCM and HCC. The original decimal ordering of the rows is assumed to be the random vertex visit order. As seen in Fig. 6, HCM matches row pairs $\{1,3\}$, $\{2,6\}$ and $\{4,5\}$ with the connectivity values of 3, 2 and 2, respectively. Note that the total number of nonzeros of $\mathbf{A}_0$ reduces from 28 to 21 in $\mathbf{A}_1^{HCM}$ after clustering. This difference is equal to the sum $3+2+2=7$ of the connectivity values of the matched row-vertex pairs since pin contractions do not lead to any single-pin nets. As seen in Fig. 6, HCC constructs three clusters $\{1,2,3\}$, $\{4,5\}$ and $\{6,7,8\}$ through the clustering sequence of $\{1,3\}$, $\{1,2,3\}$, $\{4,5\}$, $\{6,7\}$ and $\{6,7,8\}$ with the connectivity values of 3, 4, 2, 3 and 2, respectively. Note that pin contractions lead to three single-pin nets n_2, n_3 and n_7, thus columns 2, 3 and 7 are removed. As also seen in Fig. 6, although rows 7 and 8 remain unmatched in HCM, every row is involved in at least one clustering in HCC.

Both HCM and HCC necessitate scanning the pin lists of all nets in the net list of the source vertex to find its adjacent vertices for matching and clustering. In the column-net (row-net) model, the total cost of these scan operations can be as expensive as the total number of multiply and add operations which lead to nonzero entries in the computation of $\mathbf{A}\mathbf{A}^T$ ($\mathbf{A}^T\mathbf{A}$). In HCM, the key point to efficient implementation is to move the matched vertices encountered during the scan of the pin list of a net to the end of its pin list through a simple swap operation. This scheme avoids the re-visits of the matched vertices during the following matching operations at that level. Although this scheme requires an additional index array to maintain the temporary tail indices of the pin lists, it achieves substantial decrease in the run-time of the coarsening phase. Unfortunately, this simple yet effective scheme cannot be fully used in HCC. Since a singleton vertex can select a multinode cluster, the re-visits of the clustered vertices are partially avoided by maintaining only a single vertex to represent the multinode cluster in the pin-list of each net connected to the cluster, through simple swap operations. Through the use of these efficient implementation schemes the total cost of the scan operations in the column-net (row-net) model can be as low as the total number of nonzeros in $\mathbf{A}\mathbf{A}^T$ ($\mathbf{A}^T\mathbf{A}$). In order to maintain this cost within reasonable limits, all nets of size greater than $4s_{avg}$ are not considered in a bi-partitioning step, where s_{avg} denotes the average net size of the hypergraph to

be partitioned in that step. Note that such nets can be reconsidered during the further levels of recursion because of net splitting.

The cluster growing operation in HCC requires disjoint-set operations for maintaining the representatives of the clusters, where the union operations are restricted to the union of a singleton source cluster with a singleton or a multinode target cluster. This restriction is exploited by always choosing the representative of the target cluster as the representative of the new cluster. Hence, it is sufficient to update the representative pointer of only the singleton source cluster joining to a multinode target cluster. Therefore, each disjoint-set operation required in this scheme is performed in $O(1)$ time.

6.2 Initial Partitioning Phase

The goal in this phase is to find a bipartition on the coarsest hypergraph $\mathcal{H}_m$. In PaToH, we use *Greedy Hypergraph Growing (GHG)* algorithm for bisecting $\mathcal{H}_m$. This algorithm can be considered as an extension of the GGGP algorithm used in MeTiS to hypergraphs. In GHG, we grow a cluster around a randomly selected vertex. During the coarse of the algorithm, the selected and unselected vertices induce a bipartition on $\mathcal{H}_m$. The unselected vertices connected to the growing cluster are inserted into a priority queue according to their FM gains. Here, the gain of an unselected vertex corresponds to the decrease in the cutsize of the current bipartition if the vertex moves to the growing cluster. Then, a vertex with the highest gain is selected from the priority queue. After a vertex moves to the growing cluster, the gains of its unselected adjacent vertices which are currently in the priority queue are updated and those not in the priority queue are inserted. This cluster growing operation continues until a predetermined bipartition balance criterion is reached. As also mentioned in MeTiS, the quality of this algorithm is sensitive to the choice of the initial random vertex. Since the coarsest hypergraph $\mathcal{H}_m$ is small, we run GHG 4 times starting from different random vertices and select the best bipartition for refinement during the uncoarsening phase.

6.3 Uncoarsening Phase

At each level i (for $i = m, m-1, \ldots, 1$), bipartition Π_i found on $\mathcal{H}_i$ is projected back to a bipartition Π_{i-1} on $\mathcal{H}_{i-1}$. The constituent vertices of each multinode in $\mathcal{H}_{i-1}$ is assigned to the part of the respective vertex in $\mathcal{H}_i$. Obviously, Π_{i-1} of $\mathcal{H}_{i-1}$ has the same cutsize with Π_i of $\mathcal{H}_i$. Then, we refine this bipartition by running a *Boundary FM (BFM)* hypergraph bipartitioning algorithm on $\mathcal{H}_{i-1}$ starting from initial bipartition Π_{i-1}. BFM moves only the boundary vertices from the overloaded part to the under-loaded part, where a vertex is said to be a boundary vertex if it is connected to an at least one cut net.

BFM requires maintaining the *pin-connectivity* of each net for both initial gain computations and gain updates. The pin-connectivity $\sigma_k[n] = |n \cap \mathcal{P}_k|$ of a net n to a part $\mathcal{P}_k$ denotes the number of pins of net n that lie in part $\mathcal{P}_k$, for $k = 1, 2$. In order to avoid the scan of the pin lists of all nets, we adopt an efficient scheme to initialize the σ values for the first BFM pass in a level. It is clear that initial bipartition Π_{i-1} of $\mathcal{H}_{i-1}$ has the same cut-net set with Π_i of $\mathcal{H}_i$. Hence, we scan only the pin lists of the cut nets of Π_{i-1} to initialize their σ values. For each other net n, $\sigma_1[n]$ and $\sigma_2[n]$ values are easily initialized as $\sigma_1[n] = s_n$ and $\sigma_2[n] = 0$ if net n is internal to part $\mathcal{P}_1$, and $\sigma_1[n] = 0$ and $\sigma_2[n] = s_n$ otherwise. After initializing the gain value of each vertex v as $g[v] = -d_v$, we exploit σ values as follows. We re-scan the pin list of each external net n and update the gain value of each vertex $v \in pins[n]$ as $g[v] = g[v] + 2$ or $g[v] = g[v] + 1$ depending on whether net n is *critical* to the part containing v or not, respectively. An external net n is said to be critical to a part k if $\sigma_k[n] = 1$ so that moving the single vertex of net n that lies in that part to the other part removes net n from the cut. Note that two-pin cut nets are critical to both parts. The vertices visited while scanning the pin-lists of the external nets are identified as boundary vertices and only these vertices are inserted into the priority queue according to their computed gains.

In each pass of the BFM algorithm, a sequence of unmoved vertices with the highest gains are selected to move to the other part. As in the original FM algorithm, a vertex move necessitates gain updates of its adjacent vertices. However, in the BFM algorithm, some of the adjacent vertices of the moved

vertex may not be in the priority queue, because they may not be boundary vertices before the move. Hence, such vertices which become boundary vertices after the move are inserted into the priority queue according to their updated gain values. The refinement process within a pass terminates either no *feasible* move remains or the sequence of last $max\{50,\ 0.001|\mathcal{V}_i|\}$ moves does not yield a decrease in the total cutsize. A move is said to be feasible if it does not disturb the load balance criterion (2.1) with $K=2$. At the end of a BFM pass, we have a sequence of tentative vertex moves and their respective gains. We then construct from this sequence the maximum prefix subsequence of moves with the maximum prefix sum which incurs the maximum decrease in the cutsize. The permanent realization of the moves in this maximum prefix subsequence is efficiently achieved by rolling back the remaining moves at the end of the overall sequence. The initial gain computations for the following pass in a level is achieved through this rollback. The overall refinement process in a level terminates if the maximum prefix sum of a pass is not positive. In the current implementation of PaToH, at most 2 BFM passes are allowed at each level of the uncoarsening phase.

Chapter 7

Conclusion

Two computational hypergraph models were proposed to decompose sparse matrices in 1D for minimizing communication volume while maintaining load balance during repeated parallel matrix-vector product computations. The proposed models enable the representation and hence the decomposition of structurally nonsymmetric matrices as well as structurally symmetric matrices. Furthermore, they introduce a much more accurate representation for the communication requirement than the standard computational graph model widely used in the literature for the parallelization of various scientific applications. The proposed models reduce the 1D decomposition problem to the well-known hypergraph partitioning problem thus enabling the use of circuit partitioning heuristics widely used in VLSI design. Experimental results carried out on a wide range of sparse test matrices arising in different application domains confirmed the validity of the proposed hypergraph models. In the 1D decomposition of the test matrices, the use of the proposed hypergraph models instead of the graph models achieved 30%-38% decrease in the communication volume requirement of a single parallel matrix-vector multiplication at the expense of only 34%–130% increase in the decomposition time by using PaToH, on the average.

In the literature, there was a lack of existence of 2D decomposition heuristics for parallel SpMxV computations. This thesis provides three different hypergraph models for 2D decomposition of sparse matrices, a fine-grain hypergraph model and hypergraph models for jagged-like and checkerboard decompositions.

The proposed fine-grain hypergraph model produced the best decompositions in terms of communication volume. For the architecture with high start-up costs, number of messages is also important. For those kind of architectures, checkerboard decomposition model is a good choice, since it restricts the communication to be done only on the rows or columns of the 2D processor mesh, hence the upper bound on the number of messages is very low. In the 2D decomposition of the test matrices, all of the proposed hypergraph models produces 26%-59% better decompositions, on the overall average, than the standard graph model that enables 1D decompositions.

Graph and graph partitioning are also widely used in nested dissection based low fill ordering tools. Graph partitioning encountered in this domain is formulated as graph partitioning by vertex separator (GPVS). In this thesis, we showed that GPVS problem can be formulated as hypergraph partitioning problem. We have exploited this finding to develop a novel hypergraph partitioning based fill reducing ordering method, to order the AA^T kind matrices encountered in the solution of LP problems. For general symmetric matrices, the proposed method extended by the notion of 2-clique decomposition of the matrix. In the ordering of matrices arising from LP problems, the proposed method produced 45% better orderings than MMD ordering heuristic in terms of operation count, by the expense of 20% larger execution time. In the ordering of general symmetric test matrices, the proposed method produces 25% better orderings than MMD, however it is approximately 7 times slower than MMD implementation we have used.

In this work, a successful multilevel hypergraph partitioning tool PaToH was also implemented. PaToH is found to be approximately 4 times faster than its only competitor hMeTiS while producing the same quality results. 2D checkerboard decomposition requires multi-constraint hypergraph partitioning. Hence, PaToH was extended to handle the multi-constraints. Hypergraph partitioning based nested dissection also requires additional extensions, such as balance on nets, embedded constrained minimum degree, etc. PaToH was also modified to handle balance on nets, and multi-stage ordering code of SMOOTH is embedded to produce nested dissection and multisection ordering results based on hypergraph partitioning.

This work was an effort towards showing that the computational hypergraph model is more powerful than the standard computational graph model as it provides a more versatile representation for the interactions among the atomic tasks of the computational domains. In the computational graph model for general applications, each edge usually represents a two-way interaction between a pair of atomic tasks implicitly. The net (hyperedge) concept in the computational hypergraph model has the additional power of representing a multiway interaction explicitly among a set of atomic tasks through a shared data item in data parallel applications. Hence, the graph model suffices when an edge represents a unique data item of which intermediate result(s) should be communicated between exactly two processors if the atomic tasks represented by the two end vertices of this edge are assigned to different processors. Unfortunately, this is not the case in all scientific applications. There is usually a multiway interaction among the atomic tasks and thus the hypergraph is a more promising model for the decomposition of the computational domains.

Bibliography

[1] C. J. Alpert, L. W. Hagen, and A. B. Kahng. A hybrid multilevel/genetic approach for circuit partitioning. Technical report, UCLA Computer Science Department, 1996.

[2] C. J. Alpert and A. B. Kahng. Recent directions in netlist partitioning: A survey. *VLSI Journal*, 19(1–2):1–81, 1995.

[3] P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. Technical Report TR-94-039, University of Florida, Dec 1994.

[4] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM J. Sci. Statist. Comput.*, 16:1404–1411, 1995.

[5] C. Ashcraft and J. W. H. Liu. A partition improvement algorithm for generalized nested dissection. Technical Report BCSTECH-94-020, Boeing Computer Services, Seattle, WA, 1994.

[6] C. Ashcraft and J. W. H. Liu. Using domain decomposition to find graph bisectors. Technical Report ISSTECH-95-024, Boeing Information and Support Service, 1995.

[7] C. Ashcraft and J. W. H. Liu. *SMOOTH: A software package for ordering sparse matrices*, 1996.

[8] C. Ashcraft and J. W. H. Liu. Applications of the dulmage-mendelsohn decomposition and network flow to graph bisection improvement. *SIAM Journal on Matrix Analysis and Applications*, 19(2):325–354, 1998.

[9] C. Ashcraft and J. W. H. Liu. Robust ordering of sparse matrices using multisection. *SIAM Journal on Matrix Analysis and Applications*, 19(3):816–832, 1998.

[10] C. Aykanat, F. Ozguner, F. Ercal, and P. Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transactions on Computers*, 37:1554–1567, Dec 1988.

[11] C. Aykanat, A. Pınar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into singly-bordered block-diagonal form for parallel solution of lp problems. *submitted for publication.*

[12] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Infomation Processing Letters*, 42(3):153–159, May 1992.

[13] T. N. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452, 1993.

[14] T. Bultan and C. Aykanat. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, 16:292–305, 1992.

[15] W. Camp, S. J. Plimpton, B.A Hendrickson, and R. W. Leland. Massively parallel methods for engineering and science problems. *Communication of ACM*, 37(4):31–41, April 1994.

[16] W. J. Carolan, J. E. Hill, J. L. Kennington, S. Niemi, and S. J. Wichmann. An empirical evaluation of the korbx algorithms for military airlift applications. *Operations Research*, 38(2):240–248, 1990.

[17] Ü. V. Çatalyürek and C. Aykanat. Decomposing irregularly sparse matrices for parallel matrix-vector multiplications. *Lecture Notes in Computer Science*, 1117:75–86, 1996.

[18] IOWA Optimization Center. Linear programming problems. ftp://col.biz.uiowa.edu:pub/testprob/lp/gondzio.

[19] C.-K. Cheng and Y.-C. Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Transactions on Computer-Aided Design*, 10(12):1502–1511, December 1991.

[20] J. Cong, L. Hagen, and A. B. Kahng. Net partitions yield better module partitions. In *Proceedings of 29th ACM/IEEE Design Automation Conference*, pages 47–52, 1992.

[21] J. Cong, W. Labio, and N. Shivakumar. Multi-way vlsi circuit partitioning based on dual net representation. In *Proceedings of IEEE International Conference on Computer-Aided Design*, pages 56–62, 1994.

[22] J. Cong and M'L. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 755–760, 1993.

[23] T. Davis. University of florida sparse matrix collection: http://www.cise.ufl.edu/ davis/sparse/. *NA Digest*, 92/96/97(42/28/23), 1994/1996/1997.

[24] I. S. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, march 1989.

[25] 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*, pages 175–181, 1982.

[26] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[27] J. A. George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10:345–363, 1973.

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

[29] M. K. Goldberg and M. Burstein. Heuristic improvement techniques for bisection of vlsi networks. In *Proc. IEEE Intl. Conf. Computer Design*, pages 122–125, 1983.

[30] A. Gupta. Fast and effective algorithms for graph partitioning and sparse matrix ordering. Technical Report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.

[31] A. Gupta. Watson graph partitioning package. Technical Report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.

[32] G. Hachtel, A.R. Newton, and A. Sangiovanni-Vincentelli. An algorithm for optimal pla folding. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1(2):63–77, 1982.

[33] B. Hendrickson. Graph partitioning and parallel solvers: has the emperor no clothes? *Lecture Notes in Computer Science*, 1457:218–225, 1998.

[34] B. Hendrickson and T. G. Kolda. Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel processing. *submitted to SIAM Journal on Scientific Computing.*

[35] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories, 1993.

[36] B. Hendrickson and R. Leland. *The Chaco user's guide, version 2.0.* Sandia National Laboratories, Alburquerque, NM, 87185, 1995.

[37] B. Hendrickson, R. Leland, and S. Plimpton. An efficient parallel algorithm for matrix-vector multiplication. *Int. J. High Speed Computing*, 7(1):73–88, 1995.

[38] B. Hendrickson and E. Rothberg. Effective sparse matrix ordering: just around the bend. In *Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing.*

[39] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. Technical Report SAND96-0868J, Sandia National Laboratories, Mar 1996.

[40] E. Ihler, D. Wagner, and F. Wagner. Modeling hypergraphs by graphs with the same mincut properties. *Information Processing Letters*, 45(4):171–175, March 1993.

[41] M. Kaddoura, C. W. Qu, and S. Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Parallel and Distributed Technology*, 3(3):63–69, 1995.

[42] A. B. Kahng. Fast hypergraph partition. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 762–766, 1989.

[43] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[44] G. Karypis and V. Kumar. *MeTiS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 3.0.* University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[45] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report 98-019, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, May 1998.

[46] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, to appear.

[47] 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.

[48] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, February 1970.

[49] T. G. Kolda. Partitioning sparse rectangular matrices for parallel processing. *Lecture Notes in Computer Science*, 1457:68–79, 1998.

[50] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[51] V. Lakamsani, L. N. Bhuyan, and D. S. Linthicum. Mapping molecular dynamics computations on to hypercubes. *Parallel Computing*, 21:993–1013, 1995.

[52] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart, and Winston, 1976.

[53] C. E. Leiserson and J. G. Lewis. Orderings for parallel sparse symmetric matrix factorization. In *Third SIAM Conference on Parallel Processing for Scientific Computing*, pages 27–31, 1987.

[54] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Willey–Teubner, Chichester, U.K., 199.

[55] J. G. Lewis, D. G. Payne, and R. A. van de Geijn. Matrix-vector multiplication and conjugate gradient algorithms on distributed memory computers. In *Proceedings of the Scalable High Performance Computing Conference*, 1994.

[56] J. G. Lewis and R. A. van de Geijn. Distributed memory matrix-vector multiplication and conjugate gradient algorithms. In *Proceedings of Supercomputing'93*, pages 15–19, Portland, OR, November 1993.

[57] J. W. H. Liu. Modification of the minimum degree algorith by multiple elimination. *ACM Transactions on Mathematical Software*, 11:141–153, 1985.

[58] J. W. H. Liu. A graph partitioning algorithm by node seperator. *ACM Transactions on Mathematical Software*, 15(3):198–219, Sep 1989.

[59] J. W. H. Liu. On the minimum degree ordering with constraints. *SIAM J. Sci. Statist. Comput.*, 10:1136–1145, 1989.

[60] O. C. Martin and S. W. Otto. Partitioning of unstructured meshes for load balancing. *Concurrency: Practice and Experience*, 7(4):303–314, 1995.

[61] S. G. Nastea, O. Frieder, and T. El-Ghazawi. Load-balanced sparse matrix-vector multiplication on parallel computers. *Journal of Parallel and Distributed Computing*, 46:439–458, 1997.

[62] A. T. Ogielski and W. Aielo. Sparse matrix computations on parallel processor arrays. *SIAM Journal on Numerical Analysis*, 1993.

[63] A. Pınar, Ü. V. Çatalyürek, C. Aykanat, and M. Pınar. Decomposing linear programs for parallel solution. *Lecture Notes in Computer Science*, 1041:473–482, 1996.

[64] C. Pommerell, M. Annaratone, and W. Fichtner. A set of new mapping and colloring heuristics for distributed-memory parallel processors. *SIAM Journal of Scientific and Statistical Computing*, 13(1):194–226, January 1992.

[65] A. Pothen and C. J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, 1990.

[66] A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis and Applications*, 11(3):430–452, 1990.

[67] A. Pothen, H. D. Simon, L. Wang, and S. T. Bernhard. Towards a fast implementation of spectral nested dissection. In *Proceedings of Supercomputing'92*, pages 42–51, 1992.

[68] C.-W. Qu and S. Ranka. Parallel incremental graph partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):884–896, 1997.

[69] E. Rothberg. Ordering sparse matrices using approximate minimum local fill. *submitted for publication*, 1996.

[70] Y. Saad, K. Wu, and S. Petiton. Sparse matrix computations on the cm-5. In *Proc. 6th SIAM Conf. on Parallel Processing for Scientifical Computing*, 1993.

[71] K. Schloegel, G. Karypis, and V. Kumar. A new algorithm for multi-objective graph partitioning. Technical Report 99-003, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN 55455, Sep 1999.

[72] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proceedings of the 9th ACM/IEEE Design Automation Conference*, pages 57–62, 1972.

[73] H. Shin and C. Kim. A simple yet effective technique for partitioning. *IEEE Transactions on VLSI Systems*, 1(3):380–386, Sep 1993.

[74] W. F. Tinney and J. W. Walker. Direct solution of sparse network equations by optimally ordered triangular factorization. In *Proc. IEEE*, volume 55, pages 1801–1809, 1967.

[75] Y.-C. Wei and C.-K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Transactions on Computer-Aided Design*, 10(7):911–921, July 1991.