RECURSIVE BIPARTITIONING MODELS FOR PERFORMANCE IMPROVEMENT IN SPARSE MATRIX COMPUTATIONS

A DISSERTATION SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER ENGINEERING

By

Seher Acer

August 2017

RECURSIVE BIPARTITIONING MODELS FOR PERFORMANCE IMPROVEMENT IN SPARSE MATRIX COMPUTATIONS

By Seher Acer

August 2017

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

Cevdet Aykanat (Advisor)

Muhammet Mustafa Özdal

Murat Manguoğlu

Tayfun Küçükyılmaz

Engin Demir

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

RECURSIVE BIPARTITIONING MODELS FOR PERFORMANCE IMPROVEMENT IN SPARSE MATRIX COMPUTATIONS

Seher Acer
Ph.D. in Computer Engineering
Advisor: Cevdet Aykanat
August 2017

Sparse matrix computations are among the most important building blocks of linear algebra and arise in many scientific and engineering problems. Depending on the problem type, these computations may be in the form of sparse matrix dense matrix multiplication (SpMM), sparse matrix vector multiplication (SpMV), or factorization of a sparse symmetric matrix. For both SpMM and SpMV performed on distributed-memory architectures, the associated data and task partitions among processors affect the parallel performance in a great extent, especially for the sparse matrices with an irregular sparsity pattern. Parallel SpMM is characterized by high volumes of data communicated among processors, whereas both the volume and number of messages are important for parallel SpMV. For the factorization performed in envelope methods, the envelope size (i.e., profile) is an important factor which determines the performance. For improving the performance in each of these sparse matrix computations, we propose graph/hypergraph partitioning models that exploit the advantages provided by the recursive bipartitioning (RB) paradigm in order to meet the specific needs of the respective computation. In the models proposed for SpMM and SpMV, we utilize the RB process to enable targeting multiple volume-based communication cost metrics and the combination of volume- and number-based communication cost metrics in their partitioning objectives, respectively. In the model proposed for the factorization in envelope methods, the input matrix is reordered by utilizing the RB process in which two new quality metrics relating to profile minimization are defined and maintained. The experimantal results show that the proposed RB-based approach outperforms the state-of-the-art for each mentioned computation.

Keywords: Sparse matrices, recursive bipartitioning, graph partitioning, hypergraph partitioning, distributed-memory architectures, communication cost, envelope methods, factorization, profile reduction.

ÖZET

SEYREK MATRİS HESAPLAMALARINDA PERFORMANS İYİLEŞMESİ İÇİN ÖZYİNELEMELİ İKİYE BÖLÜMLEME MODELLERİ

Seher Acer
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: Cevdet Aykanat
Ağustos 2017

Seyrek matris hesaplamaları lineer cebirin en önemli yapıtaşlarından olup bilim ve mühendislik alanlarında birçok problemde ortaya çıkmaktadırlar. Bu hesaplamalar, problem türüne bağlı olarak seyrek matris yoğun matris çarpımı (SyMM), seyrek matris vektör çarpımı (SyMV), veya seyrek simetrik bir matrisin faktörizasyonu şeklinde olabilirler. Dağıtık bellekli mimarilerde gerçekleştirilen SyMM ve SyMV işlemlerinde, özellikle düzensiz seyreklik örüntüsü olan matrisler için, işlemciler arasındaki veri ve görev bölümlemesi paralel performansı fazlaca etkilemektedir. Paralel SyMM işlemciler arasındaki yüksek hacimli iletişimler ile nitelendirilirken, paralel SyMV için hem iletişim hacmi hem de mesaj sayısı önemli olmaktadır. Zarf yöntemlerinde gerçekleştirilen faktörizasyonda, matris zarfının büyüklüğü yani matris profili faktörizasyon performansını belirleyen önemli bir etmendir. Bahsi geçen hesaplamaların performanslarını iyileştirmek amacıyla bu hesaplamaların herbiri için özyinelemeli ikiye bölümleme (ÖİB) paradigmasını kullanan çizge/hiperçizge bölümleme modelleri önermekteyiz. Önerilmekte olan modeller, ÖİB tarafından sunulan avantajları ilgili hesaplamanın performans iyileşmesi yönünde spesifik ihtiyaçlarını karşılama amacıyla kullanmaktadırlar. ÖİB işlemi bölümleme objektifinin, SyMM için önerilen modellerde birden fazla hacim tabanlı iletişim maliyeti ölçütlerini, SyMV için ise hem hacim hem mesaj sayısı ölçütlerini hedefleyecek şekilde kullanılmasını sağlamaktadır. Zarf yöntemlerindeki faktörizasyon için önerilen modelde ise, ÖİB işlemi, matris profilini küçültme ile yakından ilişkili olan iki yeni kalite ölçütünün tanımlanıp hedeflenmesine izin vererek girdi matrisin satır ve sütunlarını bu hedefle yeniden sıralanmasını sağlamakadır. Deneysel sonuçlar ÖİB yaklaşımının bahsi geçen herbir hesaplama için alanında var olan en iyi yöntemlerden daha başarılı olduğunu göstermektedir.

Anahtar sözcükler: Seyrek matrisler, özyinelemeli ikiye bölümleme, çizge

bölümleme, hiperçizge bölümleme, dağıtık bellekli mimariler, iletişim maliyeti, zarf yöntemleri, faktörizasyon, profil azaltma.

Acknowledgement

I am grateful to Prof. Dr. Cevdet Aykanat for his invaluable guidance and support during my Ph.D. studies. It has always been a privilige to do research with him. I owe thanks to Murat Manguoğlu, Özcan Öztürk, Mustafa Özdal, Tayfun Küçükyılmaz and Engin Demir, for their feedbacks on this thesis. I am also grateful to my colleagues Enver and Oguz, who have valuable contributions in this thesis. My friends made me feel home at Bilkent; I am grateful to Özlem, Elif İmre, Merve, Ayşe, Semih, İstemi, İlker, Sinan, Burak, Başak, Bengü, Elif Eser, and Jonathon for everything we shared. I would like to express my deepest gratitudes to my mother, my father and my sister for their unconditional love and support. As I grow older and witness more, I feel more and more lucky to have such a decent family. I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) for supporting me throughout my Ph.D. studies under the national scholarship program BIDEB 2211. I also thank TÜBİTAK 1001 program for supporting me in project EEEAG-115E212.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sparse matrix computations are among the most important building blocks of linear algebra and arise in many scientific and engineering problems in different forms. This thesis considers three forms of sparse matrix computations: sparse matrix dense matrix multiplication (SpMM), sparse matrix vector multiplication (SpMV) and the factorization of a sparse symmetric matrix performed in envelope methods.

SpMM is a common operation in block iterative methods such as block conjugate gradient variants [2, 3, 4, 5], block Lanczos [6] and block Arnoldi [7]. It is also used in big data analytics and corresponds to the computations regarding a level of the level-synchronized multi-source breadth-first search [8, 9, 10, 11, 12, 13], which is used in centrality measures [14]. The performance of SpMM on a distributed-memory architecture is mainly characterized by the amount of data communicated among processors, i.e., communication volume, due to the possibility of communicating a whole row of the dense matrix for a single nonzero entry of the sparse matrix. For improving the performance of distributed-memory-parallel SpMM, multiple performance metrics regarding the communication volume should be minimized in the partitioning objective.

SpMV is also a common operation in iterative solvers. The performance of

SpMV on a distributed-memory architecture is not dominated by a single type of overhead such as volume, but better expressed as a combination of the metrics regarding the bandwidth and latency costs. Bandwidth cost is defined using the amount of the data communicated, whereas the latency cost is defined using the number of messages communicated. For improving the performance of distributed-memory-parallel SpMV, multiple performance metrics regarding the bandwidth and latency costs should be minimized in the partitioning objective.

Envelope methods are widely utilized for solving sparse symmetric systems of linear equations. These methods only store the numerical values in the envelope of the input sparse matrix and perform computations on these values. Hence, the size of the envelope, which is known as profile, determines the storage and runtime complexities of these methods [15, 16, 17, 18]. For improving the performance of these methods, the profile of the input sparse matrix should be minimized by a symmetric permutation of rows and columns.

In this thesis, we propose recursive-bipartitioning-based graph/hypergraph partitioning models for achieving performance improvement in the above-mentioned sparse matrix computations. The recursive bipartitioning (RB) paradigm is widely utilized for achieving multi-way graph/hypergraph partitioning. In the RB paradigm, an input graph/hypergraph is bipartitioned (i.e., partitioned into two parts), then two new graphs/hypergraphs are formed using these two parts, and then these graphs/hypergraphs are recursively bipartitioned repeating the same procedure at each bipartitioning until the desired number of parts is obtained. The objective of multi-way partitioning is handled in the RB process by various techniques such as cut-edge/net removal and cut-net splitting [19].

The RB paradigm provides many flexibilities which direct multi-way partitioning can not provide. An example to these flexibilities might be measuring the overall partition quality at each RB step and adjusting the partitioning parameters accordingly. Another example might be adding additional objectives which can only be formulated by the overall partition information to the individual RB steps. The models proposed in this thesis all exploit the flexibilities provided by

the RB paradigm, each in a different way. For SpMM, we utilize the RB paradigm to propose graph/hypergraph partitioning models which can minimize multiple volume-based metrics simultaneously in a single partitioning phase. For encoding the volume-based metrics other than total volume, we use multi-constraint partitioning and assign communication loads to vertices as additional vertex weights in each RB step. For SpMV, we utilize the RB paradigm to propose two hypergraph partitioning models which can minimize bandwith and latency costs simultaneously in a single partitioning phase. For encoding the latency cost, we use message nets to represent the messages between processor groups and add them to the respective hypergraph in each RB step. For the envelope methods, we utilize the RB paradigm to propose a hypergraph partitioning model which reorders the matrix by maintaining two quality metrics which relate to profile minimization. For this purpose, we use two different nets for each row of the matrix and manipulate these nets in each RB step accordingly.

The rest of the thesis is organized as follows. Chapter 2 gives background on graph and hypergraph partitioning problems, their variants and the RB paradigm. Chapter 3 presents the proposed graph and hypergraph partitioning models for improving the performance of the distibuted-memory parallel SpMM and the corresponding experimental evaluation. Chapter 4 presents the proposed hypergraph partitioning models for improving the performance of the distributed-memory-parallel SpMV and the corresponding experimental evaluation. Chapter 5 presents the proposed hypergraph partitioning model for improving the perfomance of envelope methods by profile reduction and the corresponding experimental evaluation. Chapter 6 concludes the thesis.

Chapter 2

Background

2.1 Graph partitioning problem

A graph $G = (\mathcal{V}, \mathcal{E})$ is defined as a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges, where each edge connects a pair of distinct vertices. The edge that connects vertices v_i and v_j is denoted by $e_{i,j}$. A vertex v_j is said to be adjacent to vertex v_i if $e_{i,j} \in \mathcal{E}$. The set of vertices adjacent to v_i is denoted by $Adj(v_i)$ and formulated as

$$Adj(v_i) = \{v_j : e_{i,j} \in \mathcal{E}\}.$$

$\Pi_K(G) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ denotes a K-way partition of G if vertex parts are mutually disjoint and exhaustive. For a given $\Pi_K(G)$, an edge $e_{i,j}$ is said to be cut if the vertices connected by $e_{i,j}$ are assigned to different parts. A vertex v_i is said to be a boundary vertex in $\Pi_K(G)$ if it is connected by at least one cut edge.

In G, each edge $e_{i,j}$ is assigned a cost, which is denoted by $c(e_{i,j})$. For a given partition $\Pi_K(G)$, the cutsize is defined as the sum of the costs of the cut edges,

which is formulated as

$$cutsize = \sum_{e_{i,j}:v_i \in \mathcal{V}_k, v_j \in \mathcal{V}_{\ell \neq k}} c(e_{i,j}).$$

In G, each vertex v_i is assigned a weight, which is denoted by $w(v_i)$. For a given partition $\Pi_K(G)$, the weight $W(\mathcal{V}_k)$ of part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, that is, $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$. For a pre-determined ϵ value, a given partition $\Pi_K(G)$ is said to be balanced if the following condition holds for each part $\mathcal{V}_k \in \Pi_K(\mathcal{H})$:

$$W(\mathcal{V}_k) \leq W_{avg}(1 + \epsilon).$$

Here, W_{avg} denotes the average part weight, which is formulated as $W_{avg} = \sum_{v_i \in \mathcal{V}} w(v_i)/K$.

For given K and ϵ values, the graph partitioning problem is defined as finding a K-way partition $\Pi_K(G)$ with the objective of minimizing the cutsize under the constraint of maintaining balance on the weights of the parts. Graph partitioning problem is known to be NP-hard [20].

2.2 Hypergraph partitioning problem

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set $\mathcal{V}$ of vertices and a set $\mathcal{N}$ of nets (hyperedges), where each net connects a subset of vertices. The subset of vertices connected by a net n_i is denoted by $Pins(n_i)$. $\Pi_K(\mathcal{H}) = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ denotes a K-way partition of $\mathcal{H}$ if vertex parts are mutually disjoint and exhaustive. For a given $\Pi_K(\mathcal{H})$, a net n_i is said to connect a part $\mathcal{V}_k$ if it connects at least one vertex in $\mathcal{V}_k$, i.e., $Pins(n_i) \cap \mathcal{V}_k \neq \emptyset$. The connectivity $\Lambda(n_i)$ of n_i is defined as the set of the parts that are connected by n_i. That is, $\Lambda(n_i) = \{\mathcal{V}_k : Pins(n_i) \cap \mathcal{V}_k \neq \emptyset\}$. The number of parts that are connected by n_i is denoted $\lambda(n_i)$, that is, $\lambda(n_i) = |\Lambda(n_j)|$. A net n_i is said to be cut if it connects multiple parts, i.e., $\lambda(n_j) > 1$, and

uncut (internal), otherwise, i.e., $\lambda(n_i) = 1$. A vertex v_i is said to be a boundary vertex in $\Pi_K(\mathcal{H})$ if it is connected by at least one cut net.

In $\mathcal{H}$, each net n_i is assigned a cost, which is denoted by $c(n_i)$. For a given partition $\Pi_K(\mathcal{H})$, there are two commonly-used cutsize definitions, which are explained as follows. The cutsize according to the cut-net metric corresponds to the sum of the costs of the cut nets in $\Pi_K(\mathcal{H})$, that is,

$$cutsize = \sum_{n_i:\lambda(n_i)>1} c(n_i).$$

The cutsize according to the connectivity-1 metric is formulated as

$$cutsize = \sum_{n_i \in \mathcal{N}} c(n_i)(\lambda(n_i) - 1).$$

Note that only the cut nets contribute to the summation given in this formulation since $\lambda(n_i) - 1 = 0$ for an uncut net n_i. Also note that for $K = 2$, the cutsizes given by these two different definitions are always equal to each other.

In $\mathcal{H}$, each vertex v_i is assigned a weight, which is denoted by $w(v_i)$. For a given partition $\Pi_K(\mathcal{H})$, the weight $W(\mathcal{V}_k)$ of part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$, that is, $W(\mathcal{V}_k) = \sum_{v_i \in \mathcal{V}_k} w(v_i)$. For a pre-determined ϵ value, a given partition $\Pi_K(\mathcal{H})$ is said to be balanced if the following condition holds for each part $\mathcal{V}_k \in \Pi_K(\mathcal{H})$:

$$W(\mathcal{V}_k) \leq W_{avg}(1 + \epsilon).$$

Here, W_{avg} denotes the average part weight, which is formulated as $W_{avg} = \sum_{v_i \in \mathcal{V}} w(v_i)/K$.

For given K and ϵ values, the hypergraph partitioning problem is defined as finding a K-way partition $\Pi_K(\mathcal{H})$ with the objective of minimizing the cutsize under the constraint of maintaining balance on the weights of the parts. Hypergraph partitioning problem is known to be NP-hard [21].

2.3 Multi-constraint graph/hypergraph partitioning

In the multi-constraint graph/hypergraph partitioning problem, $C > 1$ weights are assigned to each vertex instead of a single weight. The cth weight assigned to vertex v_i is denoted by $w^c(v_i)$, for $1 \leq c \leq C$. For a given partition $\Pi_K(G)/\Pi_K(\mathcal{H})$, the cth weight of a part $\mathcal{V}_k$ is defined as the sum of the cth weights of the vertices in $\mathcal{V}_k$. Then, for a predetermined ϵ^c value for each $c \in \{1, 2, \ldots, C\}$, a given partition $\Pi_K(G)/\Pi_K(\mathcal{H})$ is said to be balanced if the following condition holds for each part $\mathcal{V}_k$ and each $c \in \{1, 2, \ldots, C\}$:

$$W^c(\mathcal{V}_k) \leq W^c_{avg}(1 + \epsilon^c).$$

Here, W^c_{avg} denotes the average part weight for the cth vertex weight, which is formulated as $W^c_{avg} = \sum_{v_i \in \mathcal{V}} w^c(v_i)/K$.

The multi-constraint graph/hypergraph partitioning problem [22, 23] is then defined as finding a K-way partition of a given graph/hypergraph with the objective of minimizing the cutsize under the constraints of maintaining balance on each weight of the parts. Note that the graph/hypergraph partitioning problem given in Section 2.1/2.2 is a special case of the multi-constraint graph/hypergraph partitioning problem for $C = 1$.

2.4 Graph/hypergraph partitioning with fixed vertices

In case of graph/hypergraph partitioning with fixed vertices, we have an additional constraint on part assignment of some vertices, i.e., a number of vertices are assigned to parts prior to partitioning with the condition that, at the end of the partitioning, those vertices will remain in the part that they are assigned to.

2.5 Recursive bipartitioning paradigm

Recursive bipartitioning (RB) is a successful paradigm which has been commonly used for obtaining multi-way partitions. In this paradigm, the input graph/hypergraph is first bipartitioned (i.e., partitioned into two parts) and two new graphs/hypergraphs are formed using this bipartition information. Then, these two new graphs/hypergraphs are bipatitioned in a recursive manner until the desired number of parts is obtained.

In the RB process for partitioning a graph, while forming the two new graphs in each RB step, the vertex-induced subgraphs are obtained using the corresponding bipartition. In the RB process for partitioning a hypergraph, each internal net is included as is in the new hypergraph formed for the part which the corresponding net is internal to. For the cut-nets, there are two commonly-used techniques. In the cut-net removal technique, none of the cut nets is included in the new hypergraphs. In this technique, the sum of the cutsizes of the bipartitions encodes the cutsize of the resulting K-way partition according to the cut-net metric. In the cut-net splitting technique, each cut net is split into two new nets, where each split net connects the vertices in only one of the parts of the bipartition. It can be considered as bipartitioning the vertices connected by a cut net into two nets in a way conformal to the overall bipartition of the corresponding hypergraph. In this technique, the sum of the cutsizes of the bipartitions encodes the cutsize of the resulting K-way partition according to the connectivity-1 metric.

The RB process forms a hypothetical full binary tree, which we refer to as the RB tree, where each node represents a graph/hypergraph formed and/or bipartitioned throughout this process. For example, the topmost node of the RB tree represents the input graph/hypergraph. Note that it is not necessary to form the graph/hypergraph for a leaf node if that graph/hypergraph is not going to be bipartitioned, i.e., the recursion stops at that node. For a K-way partitioning, the overall RB process stops when there are K leaves in the RB tree. The vertex sets of the leaves then induce a K-way partition of the input graph/hypergraph.

Suppose that the required number of parts, i.e., K, is a power of 2. For obtaining a balanced K-way partition, all graphs/hypergraphs belonging to the same level of the RB tree should be bipartitioned. That is, the RB tree formed for a balanced K-way partition should be a complete binary tree with $\log_2 K + 1$ levels.

Throughout this thesis, the RB-tree levels are numbered as $0, 1, 2, \ldots, \log_2 K + 1$, from top to bottom. Likewise, the nodes belonging to a same level ℓ of the tree are numbered as $0, 1, 2, \ldots, 2^\ell - 1$, from left to right. A graph/hypergraph represented by the kth node in the ℓth level is denoted by $G_k^\ell / \mathcal{H}_k^\ell$. We use the same subscript and superscript notation to refer to the vertex and edge/net sets of these hypergraphs. That is, the vertex and net sets of $\mathcal{H}_k^\ell$ are denoted by $\mathcal{V}_k^\ell$ and $\mathcal{N}_k^\ell$, respectively. Then, the K-way partition resulting from the RB process can be formulated as

$$\{\mathcal{V}_0^{\log_2 K}, \mathcal{V}_1^{\log_2 K}, \ldots, \mathcal{V}_{K-1}^{\log_2 K}\}.$$

Chapter 3

Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems

Sparse matrix kernels form the computational basis of many scientific and engineering applications. An important kernel is the sparse matrix dense matrix multiplication (SpMM) of the form $Y = AX$, where A is a sparse matrix, and X and Y are dense matrices.

SpMM is already a common operation in computational linear algebra, usually utilized repeatedly within the context of block iterative methods. The practical benefits of block methods have been emphasized in several studies. These studies either focus on the block versions of certain solvers (i.e., conjugate gradient variants) which address multiple linear systems [2, 3, 4, 5], or the block methods for eigenvalue problems, such as block Lanczos [6] and block Arnoldi [7]. The column dimension of X and Y in block methods are usually very small compared to that of A [25].

see [24] for the original work

Along with other sparse matrix kernels, SpMM is also used in the emerging field of big data analytics. Graph algorithms are ubiquitous in big data analytics. Many graph analysis approaches such as centrality measures [14] rely on shortest path computations and use breadth-first search (BFS) as a building block. As indicated in several recent studies [8, 9, 10, 11, 12, 13], processing each level in BFS is actually equivalent to a sparse matrix vector "multiplication". Graph algorithms often necessitate BFS from multiple sources. In this case, processing each level becomes equivalent to multiplication of a sparse matrix with another sparse (the SpGEMM kernel [26]) or dense matrix. For a typical small world network [27], matrix X is sparse at the beginning of BFS, however it usually gets denser as BFS proceeds. Even in cases when it remains sparse, the changing pattern of this matrix throughout the BFS levels and the related sparse bookkeeping overhead make it plausible to store it as a dense matrix if there is memory available.

SpMM is provided in Intel MKL [28] and Nvidia cuSPARSE [29] libraries for multi-/many-core and GPU architectures. To optimize SpMM on distributed memory architectures for sparse matrices with irregular sparsity patterns, one needs to take communication bottlenecks into account. Communication bottlenecks are usually summarized by latency (message start-up) and bandwidth (message transfer) costs. The latency cost is proportional to the number of messages while the bandwidth cost is proportional to the number of words communicated, i.e., communication volume. These costs are usually addressed in the literature with intelligent graph and hypergraph partitioning models that can exploit irregular patterns quite well [30, 31, 19, 32, 33, 34]. Most of these models focus on improving the performance of parallel sparse matrix vector multiplication. Although one can utilize them for SpMM as well, SpMM necessitates the use of new models tailored to this kernel since it is specifically characterized with its high communication volume requirements because of the increased column dimensions of dense X and Y matrices. In this regard, the bandwidth cost becomes critical for overall performance, while the latency cost becomes negligible with increased average message size. Therefore, to get the best performance out of SpMM, it is vital to address communication cost metrics that are centered around volume

such as maximum send volume, maximum receive volume, etc.

3.0.1 Related work on multiple communication cost metrics

Total communication volume is the most widely optimized communication cost metric for improving the performance of sparse matrix operations on distributed memory systems [19, 32, 35, 36, 37]. There are a few works that consider communication cost metrics other than total volume [38, 39, 40, 41, 42, 43]. In an early work, Uçar and Aykanat [39] proposed hypergraph partitioning models to optimize two different cost metrics simultaneously. This work is a two-phase approach, where the partitioning in the first phase is followed by a latter phase in which they minimize total number of messages and achieve a balance on communication volumes of processors. In a related work, Uçar and Aykanat [38] adapted the mentioned model for two-dimensional fine-grain partitioning. A very recent work by Selvitopi and Aykanat aims to reduce the latency overhead in two-dimensional jagged and checkerboard partitioning [1].

Bisseling and Meesen [40] proposed a greedy heuristic for balancing communication loads of processors. This method is also a two-phase approach, in which the partitioning in the first phase is followed by a redistribution of communication tasks in the second phase. While doing so, they try to minimize the maximum send and receive volumes of processors while respecting the total volume obtained in the first phase.

The two-phase approaches have the flexibility of working with already existing partitions. However, since the first phase is oblivious to the cost metrics addressed in the second phase, they can get stuck in local optima. To remedy this issue, Deveci et al. [42] recently proposed a hypergraph partitioner called UMPa, which is capable of handling multiple cost metrics in a single partitioning phase. They consider various metrics such as maximum send volume, total number of messages, maximum number of messages, etc., and propose a different gain

computation algorithm specific to each of these metrics. In the center of their approach are the move-based iterative improvement heuristics which make use of directed hypergraphs. These heuristics consist of a number of refinement passes. To each pass, their approach is reported to introduce an $O(VK^2)$-time overhead, where V is the number of vertices in the hypergraph (number of rows/columns in A) and K is the number of parts/processors. They also report that the slowdown of UMPa increases with increasing K with respect to the native hypergraph partitioner PaToH due to this quadratic complexity.

3.0.2 Contributions

In this study, we propose a comprehensive and generic one-phase framework to minimize multiple volume-based communication cost metrics for improving the performance of SpMM on distributed memory systems. Our framework relies on the widely adopted recursive bipartitioning paradigm utilized in the context of graph and hypergraph partitioning. Total volume can already be effectively minimized with existing partitioners [19, 32, 35]. We focus on the other important volume-based metrics besides total volume, such as maximum send/receive volume, maximum sum of send and receive volumes, etc. The proposed model associates additional weights with boundary vertices to keep track of volume loads of processors during recursive bipartitioning. The minimization objectives associated with these loads are treated as constraints in order to make use of a readily available partitioner. Achieving a balance on these weights of boundary vertices through these constraints enables the minimization of target volume-based metrics. We also extend our model by proposing two practical enhancements to handle these constraints in partitioners more efficiently.

Our framework is unique and flexible in the sense that it handles multiple volume-based metrics through the same formulation in a generic manner. This framework also allows the optimization of any *custom* metric defined on send/receive volumes. Our algorithms are computationally lightweight: they only introduce an extra $O(nnz(A))$ time to each recursive bipartitioning level, where

$nnz(A)$ is the number of nonzeros in matrix A. To the best of our knowledge, it is the first portable one-phase method that can easily be integrated into any state-of-the-art graph and hypergraph partitioner. Our work is also the first work that addresses multiple volume-based metrics in the graph partitioning context.

Another important aspect is the *simultaneous* handling of multiple cost metrics. This feature is crucial as overall communication cost is simultaneously determined by multiple factors and the target parallel application may demand optimization of different cost metrics simultaneously for good performance (SpMM and multi-source BFS in our case). In this regard, Uçar and Aykanat [38, 39] accommodates this feature for two metrics, whereas Deveci et al. [42], although addresses multiple metrics, does not handle them in a completely simultaneous manner since some of the metrics may not be minimized in certain cases. Our models in contrast can optimize all target metrics simultaneously by assigning equal importance to each of them in the feasible search space. In addition, the proposed framework allows one to define and optimize as many volume-based metrics as desired.

For experiments, the proposed partitioning models for graphs and hypergraphs are realized using the widely-adopted partitioners Metis [32] and PaToH [19], respectively. We have tested the proposed models for 128, 256, 512 and 1024 processors on a dataset of 964 matrices containing instances from different domains. We achieve average improvements of up to 61% and 78% in maximum communication volume for graph and hypergraph models, respectively, in the categories of matrices for which maximum volume is most critical. Compared to the state-of-the-art partitioner UMPa, our graph model achieves an overall improvement of 5% in the partition quality 14.5x faster and our hypergraph model achieves an overall improvement of 11% in the partition quality 3.4x faster. Our average improvements for the instances that are bounded by maximum volume are even higher: 19% for the proposed graph model and 24% for the proposed hypergraph model.

We test the validity of the proposed models for both parallel SpMM and multi-source BFS kernels on large-scale HPC systems Cray XC40 and Lenovo

NeXtScale, respectively. For parallel SpMM, compared to the standard partitioning models, our graph and hypergraph partitioning models respectively lead to reductions of 14% and 22% in runtime, on average. For parallel BFS, we show on graphs with more than a billion edges that the scalability can significantly be improved with our models compared to a recently proposed two-dimensional partitioning model [11] for the parallelization of this kernel on distributed systems.

The rest of the chapter is organized as follows. Section 3.1 gives background for partitioning sparse matrices via graph and hypergraph models. Section 3.2 defines the problems regarding minimization of volume-based cost metrics. The proposed graph and hypergraph partitioning models to address these problems are described in Section 3.3. Section 3.4 proposes two practical extensions to these models. Section 5.4 gives experimental results for investigated partitioning schemes and parallel runtimes. Section 3.6 concludes.

3.1 Background

3.1.1 Parallel SpMM with one-dimensional sparse matrix partitioning

Consider the parallelization of sparse matrix dense matrix multiplication (SpMM) of the form $Y = AX$, where A is an $n \times n$ sparse matrix, and X and Y are $n \times s$ dense matrices. Assume that A is permuted into a K-way block structure of the form

$$A_{BL} = \begin{bmatrix} C_1 & \cdots & C_K \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_K \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1K} \\ \vdots & \ddots & \vdots \\ A_{K1} & \cdots & A_{KK} \end{bmatrix}, \tag{3.1}$$

for rowwise or columnwise partitioning, where K is the number of processors

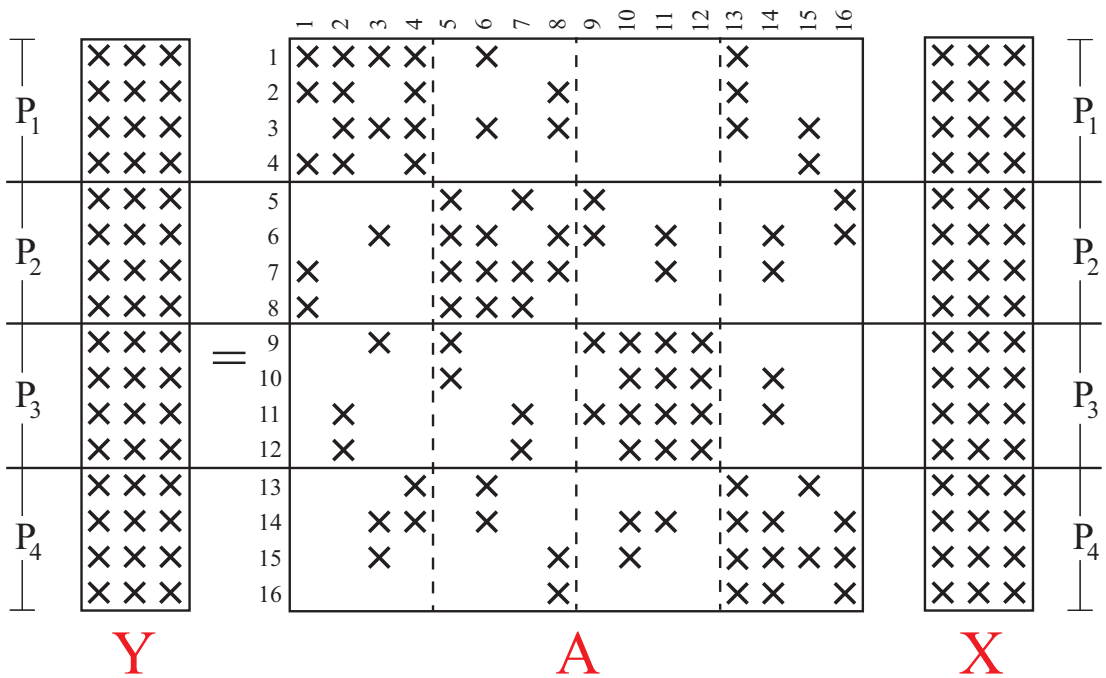


Figure 3.1: Row-parallel $Y = AX$ with $K = 4$ processors, $n = 16$ and $s = 3$.

in the parallel system. Processor P_k owns row stripe $R_k = [A_{k1} \cdots A_{kK}]$ for rowwise partitioning, whereas it owns column stripe $C_k = [A_{1k}^T \cdots A_{Kk}^T]^T$ for columnwise partitioning. We focus on rowwise partitioning in this work, however, all described models apply to columnwise partitioning as well. We use R_k and A_k interchangeably throughout the paper as we only consider rowwise partitioning.

In both block iterative methods and BFS-like computations, SpMM is performed repeatedly with the same input matrix A and changing X-matrix elements. The input matrix X of the next iteration is obtained from the output matrix Y of the current iteration via element-wise linear matrix operations. We focus on the case where the rowwise partitions of the input and output dense matrices are conformable to avoid redundant communication during these linear operations. Hence, a partition of A naturally induces partition $[Y_1^T \ldots Y_K^T]^T$ on the rows of Y, which is in turn used to induce a conformable partition $[X_1^T \ldots X_K^T]^T$ on the rows of X. In this regard, the row and column permutation mentioned in (3.1) should be conformable.

A nonzero column segment is defined as the nonzeros of a column in a specific

Algorithm 1: An SpMM iteration performed by processor P_k

Require: A_k and X_k

1: ▷ *Pre-communication phase — Expands on X-matrix rows*
2: P_k receives row i of X from P_ℓ for each nonzero column segment i in $A_{k\ell}$
3: P_k sends row j of X to P_m for each nonzero column segment j in A_{mk}

4: ▷ *Computations*
5: $Y_k \leftarrow A_k X$

6: **return** Y_k

submatrix block. For example in Figure 3.1, there are two nonzero column segments in A_{14} which belong to columns 13 and 15. In row-parallel $Y = AX$, which is given in Algorithm 1, P_k owns row stripes A_k and X_k of the input matrices, and is responsible for computing respective row stripe $Y_k = A_k X$ of the output matrix. P_k can perform computations regarding diagonal block A_{kk} locally using its own portion X_k without requiring any communication, where A_{kl} is called a diagonal block if $k = l$, and an off-diagonal block otherwise. Since P_k owns only X_k, it needs the remaining X-matrix rows that correspond to nonzero column segments in off-diagonal blocks of A_k. Hence, the respective rows must be sent to P_k by their owners in a pre-communication phase prior to SpMM computations. Specifically, to perform the multiplication regarding off-diagonal block A_{kl}, P_k needs to receive the respective X-matrix rows from P_l. For example, in Figure 3.1 for P_3, since there exists a nonzero column segment in A_{34}, P_3 needs to receive the corresponding three elements in row 14 of X from P_4. In a similar manner, it needs to receive the elements of X-matrix rows 2, 3 from P_1 and 5, 7 from P_2.

3.1.2 Sparse matrix partitioning models

In this section, we describe how to obtain a one-dimensional rowwise partitioning of matrix A for row-parallel $Y = AX$ using graph and hypergraph partitioning models. These models are the extensions of standard models used for sparse

matrix vector multiplication [19, 32, 44, 45, 46].

In the graph and hypergraph partitioning models, matrix A is represented as an undirected graph $G = (\mathcal{V}, \mathcal{E})$ and a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. In both, there exists a vertex $v_i \in \mathcal{V}$ for each row i of A, where v_i signifies the computational task of multiplying row i of A with X to obtain row i of Y. So, in both models, a single ($C = 1$) weight of s times the number of nonzeros in row i of A is associated with v_i to encode the load of this computational task. For example, in Figure 3.1, $w^1(v_5) = 4 \times 3 = 12$.

In G, each nonzero $a_{i,j}$ or a_{ji} (or both) of A is represented by an edge $e_{i,j} \in \mathcal{E}$. The cost of edge $e_{i,j}$ is assigned as $c(e_{i,j}) = 2s$ for each edge $e_{i,j}$ with $a_{i,j} \neq 0$ and $a_{ji} \neq 0$, whereas it is assigned as $c(e_{i,j}) = s$ for each edge $e_{i,j}$ with either $a_{i,j} \neq 0$ or $a_{ji} \neq 0$, but not both. In $\mathcal{H}$, each column j of A is represented by a net $n_j \in \mathcal{N}$, which connects the vertices that correspond to the rows that contain a nonzero in column j, i.e., $Pins(n_j) = \{v_i : a_{i,j} \neq 0\}$. The cost of net n_j is assigned as $c(n_j) = s$ for each net in $\mathcal{N}$.

In a K-way partition $\Pi(G)$ or $\Pi(\mathcal{H})$, without loss of generality, we assume that the rows corresponding to the vertices in part $\mathcal{V}_k$ are assigned to processor P_k. In $\Pi(G)$, each cut edge $e_{i,j}$, where $v_i \in \mathcal{V}_k$ and $v_j \in \mathcal{V}_\ell$, necessitates $c(e_{i,j})$ units of communication between processors P_k and P_ℓ. Here, P_ℓ sends row j of X to P_k if $a_{i,j} \neq 0$ and P_k sends row i of X to P_ℓ if $a_{ji} \neq 0$. In $\Pi(\mathcal{H})$, each cut net n_j necessitates $c(n_j)(\lambda(n_j) - 1)$ units of communication between processors that correspond to the parts in $\Lambda(n_j)$, where the owner of row j of X sends it to the remaining processors in $\Lambda(n_j)$. Hereinafter, $\Lambda(n_j)$ is interchangeably used to refer to parts and processors because of the identical vertex part to processor assignment.

Through these formulations, the problem of obtaining a good row partitioning of A becomes equivalent to the graph and hypergraph partitioning problems in which the objective of minimizing cutsize relates to minimizing total communication volume, while the constraint of maintaining balance on part weights corresponds to balancing computational loads of processors. The objective of

hypergraph partitioning problem is an exact measure of total volume, whereas the objective of graph partitioning problem is an approximation [19].

3.2 Problem definition

Assume that matrix A is distributed among K processors for parallel SpMM operation as described in Section 3.1.1. Let $\sigma(P_k, P_\ell)$ be the amount of data sent from processor P_k to P_ℓ in terms of X-matrix elements. This is equal to s times the number of X-matrix rows that are owned by P_k and needed by P_ℓ, which is also equal to s times the number of nonzero column segments in off-diagonal block $A_{\ell k}$. Since X_k is owned by P_k and computations on A_{kk} require no communication, $\sigma(P_k, P_k) = 0$. We use the function $ncs(.)$ to denote the number of nonzero column segments in a given block of matrix. $ncs(A_{k\ell})$ is defined to be the number of nonzero column segments in $A_{k\ell}$ if $k \neq \ell$, and 0 otherwise. This is extended to a row stripe R_k and a column stripe C_k, where $ncs(R_k) = \sum_\ell ncs(A_{k\ell})$ and $ncs(C_k) = \sum_\ell ncs(A_{\ell k})$. Finally, for the whole matrix, $ncs(A_{BL}) = \sum_k ncs(R_k) = \sum_k ncs(C_k)$. For example, in Figure 3.1, $ncs(A_{42}) = 2$, $ncs(R_3) = 5$, $ncs(C_3) = 4$ and $ncs(A_{BL}) = 21$.

The send and receive volumes of P_k are defined as follows:

- $SV(P_k)$, *send volume of* P_k*:* The total number of X-matrix elements sent from P_k to other processors. That is, $SV(P_k) = \sum_\ell \sigma(P_k, P_\ell)$. This is equal to $s \times ncs(C_k)$.
- $RV(P_k)$, *receive volume of* P_k*:* The total number of X-matrix elements received by P_k from other processors. That is, $RV(P_k) = \sum_\ell \sigma(P_\ell, P_k)$. This is equal to $s \times ncs(R_k)$.

Note that the total volume of communication is equal to $\sum_k SV(P_k) = \sum_k RV(P_k)$. This is also equal to s times the total number of nonzero column segments in all off-diagonal blocks, i.e., $s \times ncs(A_{BL})$.

In this study, we extend the sparse matrix partitioning problem in which the only objective is to minimize the total communication volume, by introducing four more minimization objectives which are defined on the following metrics:

1. $\max_k SV(P_k)$: maximum send volume of processors (equivalent to maximum $s \times ncs(C_k)$),
2. $\max_k RV(P_k)$: maximum receive volume of processors (equivalent to maximum $s \times ncs(R_k)$),
3. $\max_k(SV(P_k) + RV(P_k))$: maximum sum of send and receive volumes of processors (equivalent to maximum $s \times (ncs(C_k) + ncs(R_k))$),
4. $\max_k \max\{SV(P_k), RV(P_k)\}$: maximum of maximum of send and receive volumes of processors (equivalent to maximum $s \times \max\{ncs(C_k), ncs(R_k)\}$).

Under the objective of minimizing the total communication volume, minimizing one of these volume-based metrics (e.g., $\max_k SV(P_k)$) relates to *minimizing imbalance* on the respective quantity (e.g., imbalance on $SV(P_k)$ values). For instance, the imbalance on $SV(P_k)$ values is defined as

$$\frac{\max_k SV(P_k)}{\sum_k SV(P_k)/K}.$$

Here, the expression in the denominator denotes the average send volume of processors.

A parallel application may necessitate one or more of these metrics to be minimized. These metrics are considered besides total volume since minimization of them is plausible only when total volume is also minimized as mentioned above. Hereinafter, these metrics except total volume are referred to as volume-based metrics.

3.3 Models for minimizing multiple volume-based metrics

This section describes the proposed graph and hypergraph partitioning models for addressing volume-based cost metrics defined in the previous section. Our models have the capability of addressing a single, a combination or all of these metrics *simultaneously* in a single phase. Moreover, they have the flexibility of handling custom metrics based on volume other than the already defined four metrics. Our approach relies on the widely adopted recursive bipartitioning (RB) framework utilized in a breadth-first manner and can be realized by any graph and hypergraph partitioning tool.

3.3.1 Recursive bipartitioning

In the RB paradigm, the initial graph/hypergraph is partitioned into two subgraphs/subhypergraphs. These two subgraphs/subhypergraphs are further bipartitioned recursively until K parts are obtained. This process forms a full binary tree, which we refer to as an RB tree, with $\lg_2 K$ levels, where K is a power of 2. Without loss of generality, graphs and hypergraphs at level r of the RB tree are numbered from left to right and denoted as $G_0^r, \ldots, G_{2^r-1}^r$ and $\mathcal{H}_0^r, \ldots, \mathcal{H}_{2^r-1}^r$, respectively. From bipartition $\Pi(G_k^r) = \{\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1}\}$ of graph $G_k^r = (\mathcal{V}_k^r, \mathcal{E}_k^r)$, two vertex-induced subgraphs $G_{2k}^{r+1} = (\mathcal{V}_{2k}^{r+1}, \mathcal{E}_{2k}^{r+1})$ and $G_{2k+1}^{r+1} = (\mathcal{V}_{2k+1}^{r+1}, \mathcal{E}_{2k+1}^{r+1})$ are formed. All cut edges in $\Pi(G_k^r)$ are excluded from the newly formed subgraphs. From bipartition $\Pi(\mathcal{H}_k^r) = \{\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1}\}$ of hypergraph $\mathcal{H}_k^r = (\mathcal{V}_k^r, \mathcal{N}_k^r)$, two vertex-induces subhypergraphs are formed similarly. All cut nets in $\Pi(\mathcal{H}_k^r)$ are split to correctly encode the cutsize metric [19].

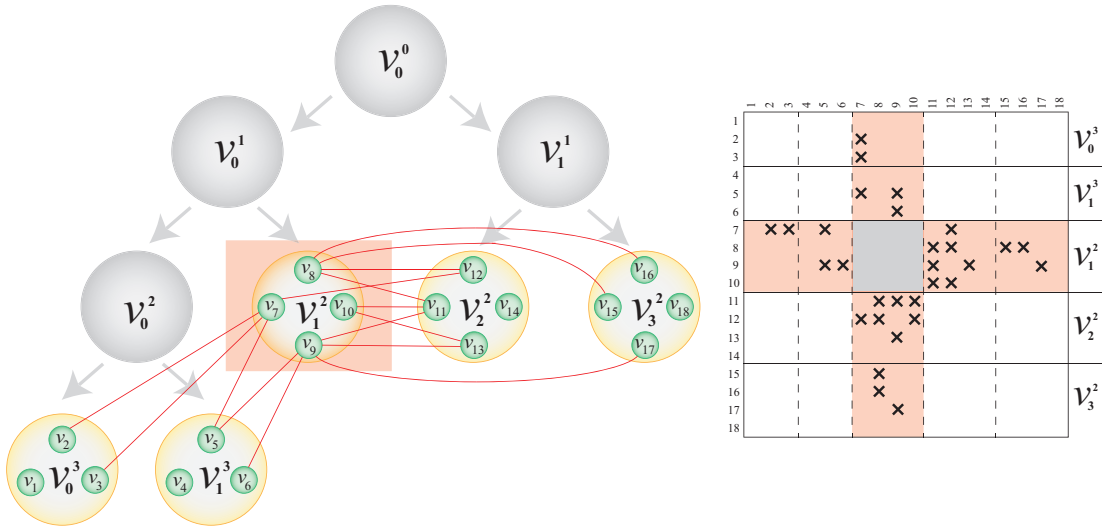


Figure 3.2: The state of the RB tree prior to bipartitioning G_1^2 and the corresponding sparse matrix. Among the edges and nonzeros, only the external (cut) edges of $\mathcal{V}_1^2$ and their corresponding nonzeros are shown.

3.3.2 Graph model

Consider the use of the RB paradigm for partitioning the standard graph representation $G = (\mathcal{V}, \mathcal{E})$ of A for row-parallel $Y = AX$ to obtain a K-way partition. We assume that the RB proceeds in a breadth-first manner and RB process is at level r prior to bipartitioning kth graph G_k^r. Observe that the RB process up to this bipartitioning already induces a K'-way partition $\Pi(G) = \{\mathcal{V}_0^{r+1}, \ldots, \mathcal{V}_{2k-1}^{r+1}, \mathcal{V}_k^r, \ldots, \mathcal{V}_{2^r-1}^r\}$. $\Pi(G)$ contains $2k$ vertex parts from level $r+1$ and $2^r - k$ vertex parts from level r, making $K' = 2^r + k$. After bipartitioning G_k^r, a $(K'+1)$-way partition $\Pi'(G)$ is obtained which contains $\mathcal{V}_{2k}^{r+1}$ and $\mathcal{V}_{2k+1}^{r+1}$ instead of $\mathcal{V}_k^r$. For example, in Figure 3.2, the RB process is at level $r = 2$ prior to bipartitioning $G_1^2 = (\mathcal{V}_1^2, \mathcal{E}_1^2)$, so, the current state of the RB induces a five-way partition $\Pi(G) = \{\mathcal{V}_0^3, \mathcal{V}_1^3, \mathcal{V}_1^2, \mathcal{V}_2^2, \mathcal{V}_3^2\}$. Bipartitioning G_1^2 induces a six-way partition $\Pi'(G) = \{\mathcal{V}_0^3, \mathcal{V}_1^3, \mathcal{V}_2^3, \mathcal{V}_3^3, \mathcal{V}_2^2, \mathcal{V}_3^2\}$. P_k^r denotes the group of processors which are responsible for performing the tasks represented by the vertices in $\mathcal{V}_k^r$. The send and receive volume definitions $SV(P_k)$ and $RV(P_k)$ of individual processor P_k are easily extended to $SV(P_k^r)$ and $RV(P_k^r)$ for processor group P_k^r.

We first formulate the send volume of the processor group P_k^r to all other

processor groups corresponding to vertex parts in $\Pi(G)$. Let *connectivity set* of vertex $v_i \in \mathcal{V}_k^r$, $Con(v_i)$, denote the subset of parts in $\Pi(G) - \{\mathcal{V}_k^r\}$ in which v_i has at least one neighbor. That is,

$$Con(v_i) = \{\mathcal{V}_\ell^t \in \Pi(G) : Adj(v_i) \cap \mathcal{V}_\ell^t \neq \emptyset\} - \{\mathcal{V}_k^r\},$$

where t is either r or $r+1$. Vertex v_i is boundary if $Con(v_i) \neq \emptyset$, and once v_i becomes boundary, it remains boundary in all further bipartitionings. For example, in Figure 3.2, $Con(v_9) = \{\mathcal{V}_1^3, \mathcal{V}_2^2, \mathcal{V}_3^2\}$. $Con(v_i)$ signifies the communication operations due to v_i, where P_k^r sends row i of X to processor groups that correspond to the parts in $Con(v_i)$. The send load associated with v_i is denoted by $sl(v_i)$ and is equal to

$$sl(v_i) = s \times |Con(v_i)|$$

The total send volume of P_k^r is then equal to the sum of the send loads of all vertices in $\mathcal{V}_k^r$, i.e., $SV(P_k^r) = \sum_{v_i \in \mathcal{V}_k^r} sl(v_i)$. In Figure 3.2, the total send volume of P_1^2 is equal to $sl(v_7)+sl(v_8)+sl(v_9)+sl(v_{10}) = 3s+2s+3s+s = 9s$. Therefore, during bipartitioning G_k^r, minimizing

$$\max\left\{ \sum_{v_i \in \mathcal{V}_{2k}^{r+1}} sl(v_i), \sum_{v_i \in \mathcal{V}_{2k+1}^{r+1}} sl(v_i) \right\}$$

is equivalent to minimizing the maximum send volume of the two processor groups P_{2k}^{r+1} and P_{2k+1}^{r+1} to the other processor groups that correspond to the vertex parts in $\Pi(G)$.

In a similar manner, we formulate the receive volume of the processor group P_k^r from all other processor groups corresponding to the vertex parts in $\Pi(G)$. Observe that for each boundary $v_j \in \mathcal{V}_\ell^t$ that has at least one neighbor in $\mathcal{V}_k^r$, P_k^r needs to receive the corresponding row j of X from P_ℓ^t. For instance, in Figure 3.2, since $v_5 \in \mathcal{V}_1^3$ has two neighbors in $\mathcal{V}_1^2$, P_1^2 needs to receive the corresponding fifth row of X from P_1^3. Hence, P_k^r receives a subset of X-matrix rows whose cardinality is equal to the number of vertices in $\mathcal{V} - \mathcal{V}_k^r$ that have at least one neighbor in $\mathcal{V}_k^r$, i.e., $|\{v_j \in \{\mathcal{V} - \mathcal{V}_k^r\} : v_i \in \mathcal{V}_k^r \text{ and } e_{ji} \in \mathcal{E}\}|$. The size of this set for $\mathcal{V}_1^2$ in Figure 3.2 is equal to 10. Note that each such v_j contributes s words to

the receive volume of P_k^r. This quantity can be captured by evenly distributing it among v_j's neighbors in $\mathcal{V}_k^r$. In other words, a vertex $v_j \in \mathcal{V}_l^t$ that has at least one neighbor in $\mathcal{V}_k^r$ contributes $s/|Adj(v_j) \cap \mathcal{V}_k^r|$ to the receive load of each vertex $v_i \in \{Adj(v_j) \cap \mathcal{V}_k^r\}$. The receive load of v_i, denoted by $rl(v_i)$, is given by considering all neighbors of v_i that are not in $\mathcal{V}_k^r$, that is,

$$rl(v_i) = \sum_{e_{ji} \in \mathcal{E} \text{ and } v_j \in \mathcal{V}_\ell^t} \frac{s}{|Adj(v_j) \cap \mathcal{V}_k^r|}.$$

The total receive volume of P_k^r is then equal to the sum of the receive loads of all vertices in $\mathcal{V}_k^r$, i.e., $RV(P_k^r) = \sum_{v_i \in \mathcal{V}_k^r} rl(v_i)$. In Figure 3.2, the vertices v_{11}, v_{12}, v_{15} and v_{16} respectively contribute $s/3$, $s/2$, s and s to the receive load of v_8, which makes $rl(v_8) = 17s/6$. The total receive volume of P_1^2 is equal to $rl(v_7)+rl(v_8)+rl(v_9)+rl(v_{10}) = 3s+17s/6+10s/3+5s/6 = 10s$. Note that this is also equal to the s times the number of neighboring vertices of $\mathcal{V}_1^2$ in $\mathcal{V} - \mathcal{V}_1^2$. Therefore, during bipartitioning G_k^r, minimizing

$$\max\left\{ \sum_{v_i \in \mathcal{V}_{2k}^{r+1}} rl(v_i), \sum_{v_i \in \mathcal{V}_{2k+1}^{r+1}} rl(v_i) \right\}$$

is equivalent to minimizing maximum receive volume of the two processor groups P_{2k}^{r+1} and P_{2k+1}^{r+1} from the other processor groups that correspond to the vertex parts in $\Pi(G)$.

Although these two formulations correctly encapsulate the send/receive volume loads of P_{2k}^{r+1} and P_{2k+1}^{r+1} to/from all other processor groups in $\Pi(G)$, they overlook the send/receive volume loads between these two processor groups. Our approach tries to refrain from this small deviation by immediately utilizing the newly generated partition information while computing volume loads in the upcoming bipartitionings. That is, the computation of send/receive loads for bipartitioning G_k^r utilizes the most recent K'-way partition information, i.e., $\Pi(G)$. This deviation becomes negligible with increasing number of subgraphs in the latter levels of the RB tree. The encapsulation of send/receive volumes between P_{2k}^{r+1} and P_{2k+1}^{r+1} during bipartitioning G_k^r necessitates implementing a new partitioning tool.

Algorithm 2: `GRAPH-COMPUTE-VOLUME-LOADS`

Input: $G = (\mathcal{V}, \mathcal{E}),\ G_k^r = (\mathcal{V}_k^r, \mathcal{E}_k^r),\ part,\ s$

1 **foreach** boundary vertex $v_i \in \mathcal{V}_k^r$ **do**

$\triangleright$ Compute the send load

2 $Con(v_i) \leftarrow \emptyset$

3 **foreach** boundary vertex $v_j \in Adj(v_i)$ **and** $v_j \notin \mathcal{V}_k^r$ **do**

4 $Con(v_i) \leftarrow Con(v_i) \cup \{part[v_j]\}$

5 $sl(v_i) \leftarrow s \times |Con(v_i)|$

$\triangleright$ Compute the receive load

6 $rl(v_i) \leftarrow 0$

7 **foreach** boundary vertex $v_j \in Adj(v_i)$ **and** $v_j \notin \mathcal{V}_k^r$ **do**

8 $rl(v_i) \leftarrow rl(v_i) + s/|Adj(v_j) \cap \mathcal{V}_k^r|$

Algorithm 2 presents the computation of send and receive loads of vertices in G_k^r prior to its bipartitioning. As its inputs, the algorithm needs the original graph $G = (\mathcal{V}, \mathcal{E})$, graph $G_k^r = (\mathcal{V}_k^r, \mathcal{E}_k^r)$, and the up-to-date partition information of vertices, which is stored in *part* array of size $V = |\mathcal{V}|$. To compute the send load of a vertex $v_i \in \mathcal{V}_k^r$, it is necessary to find the set of parts in which v_i has at least one neighbor. For this purpose, for each $v_j \notin \mathcal{V}_k^r$ in $Adj(v_i)$, $Con(v_i)$ is updated with the part that v_j is currently in (lines 2-4). $Adj(\cdot)$ lists are the adjacency lists of the vertices in the original graph G. Next, the send load of v_i, $sl(v_i)$, is simply set to s times the size of $Con(v_i)$ (line 5). To compute the receive load of $v_i \in \mathcal{V}_k^r$, it is necessary to visit the neighbors of v_i that are not in $\mathcal{V}_k^r$. For each such neighbor v_j, the receive load of v_i, $rl(v_i)$, is updated by adding v_i's share of receive load due to v_j, which is equal to $s/|Adj(v_j) \cap \mathcal{V}_k^r|$ (lines 6-8). Observe that only the boundary vertices in $\mathcal{V}_k^r$ will have nonzero volume loads at the end of this process.

Algorithm 3 presents the overall partitioning process to obtain a K-way partition utilizing breadth-first RB. For each level r of the RB tree, the graphs in this level are bipartitioned from left to right, G_0^r to $G_{2^r-1}^r$ (lines 3-4). Prior to bipartitioning of G_k^r, the send load and the receive load of each vertex in G_k^r

are computed with GRAPH-COMPUTE-VOLUME-LOADS (line 5). Recall that in the original sparse matrix partitioning with graph model, each vertex v_i has a single weight $w^1(v_i)$, which represents the computational load associated with it. To address the minimization of maximum send/receive volume, we associate an extra weight with each vertex. Specifically, to minimize the maximum send volume, the send load of v_i is assigned as its second weight, i.e., $w^2(v_i) = sl(v_i)$. In a similar manner, to minimize the maximum receive volume, the receive load of v_i is assigned as its second weight, i.e., $w^2(v_i) = rl(v_i)$. Observe that only the boundary vertices have nonzero second weights. Next, G_k^r is bipartitioned to obtain $\Pi(G_k^r) = \{\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1}\}$ using *multi-constraint* partitioning to handle multiple vertex weights (line 7). Then, two new subgraphs G_{2k}^{r+1} and G_{2k+1}^{r+1} are formed from G_k^r using $\Pi(G_k^r)$ (line 8). In partitioning, minimizing imbalance on the second part weights corresponds to minimizing imbalance on send (receive) volume if these weights are set to send (receive) loads. In other words, under the objective of minimizing total volume in this bipartitioning, minimizing

$$\frac{\max\{W^2(\mathcal{V}_{2k}^{r+1}), W^2(\mathcal{V}_{2k+1}^{r+1})\}}{(W^2(\mathcal{V}_{2k}^{r+1}) + W^2(\mathcal{V}_{2k+1}^{r+1}))/2}$$

relates to minimizing $\max\{SV(P_{2k}^{r+1}), SV(P_{2k+1}^{r+1})\}$ ($\max\{RV(P_{2k}^{r+1}), RV(P_{2k+1}^{r+1})\}$) if the second weights are set to send (receive) loads. Then *part* array is updated after each bipartitioning to keep track of the most up-to-date partition information of all vertices (line 9). Finally, the resulting K-way partition information is returned in *part* array (line 10). Note that in the final K-way partition, processor group $P_k^{\lg_2 K}$ denotes the individual processor P_k, for $0 \leq k \leq K-1$.

In order to efficiently maintain the send and receive loads of vertices, we make use of the RB paradigm in a breadth-first order. Since these loads are not known in advance and depend on the current state of the partitioning, it is crucial to act proactively by avoiding high imbalances on them. Compare this to computational loads of vertices, which is known in advance and remains the same for each vertex throughout the partitioning. Hence, utilizing a breadth-first or a depth-first RB does not affect the quality of the obtained partition in terms of computational

Algorithm 3: `GRAPH-PARTITION`

Input: $G = (\mathcal{V}, \mathcal{E}),\ K,\ s$

Let *part* be an array of size $|\mathcal{V}|$
$G_0^0 \leftarrow G$
for $r \leftarrow 0$ **to** $\lg_2 K - 1$ **do**
    for $k \leftarrow 0$ **to** $2^r - 1$ **do**
        `GRAPH-COMPUTE-VOLUME-LOADS`$(G, G_k^r, part, s)$
        Set volume-based vertex weights using $sl(v_i)$ and/or $rl(v_i)$
        Bipartition $G_k^r = (\mathcal{V}_k^r, \mathcal{E}_k^r)$ to obtain $\Pi(G_k^r) = (\mathcal{V}_{2k}^{r+1}, \mathcal{V}_{2k+1}^{r+1})$
        Form new graphs G_{2k}^{r+1} and G_{2k+1}^{r+1} using $\Pi(G_k^r)$
        Update *part* using $\Pi(G_k^r)$
return *part*

load. We prefer a breadth-first RB to a depth-first RB for minimizing volume-based metrics since operating on the parts that are at the same level of the RB tree (in order to compute send/receive loads) prevents the possible deviations from the target objective(s) by quickly adapting the current available partition to the changes that occur in send/receive volume loads of vertices.

The described methodology addresses the minimization of $\max_k SV(P_k)$ or $\max_k RV(P_k)$ separately. After computing the send and receive loads, we can also easily minimize $\max_k(SV(P_k) + RV(P_k))$ by associating the second weight of each vertex with the sum of send and receive loads, i.e., $w^2(v_i) = sl(v_i) + rl(v_i)$. For the minimization of $\max_k \max\{SV(P_k), RV(P_k)\}$, either the send loads or the receive loads are targeted at each bipartitioning. For this objective, the decision of minimizing which measure in a particular bipartitioning can be given according to the imbalance values on these measures for the current overall partition. If the imbalance on send loads is larger, then the second weights of vertices are set to the send loads, whereas if the imbalance on receive loads is larger, then the second weights of vertices are set to the receive loads. In this way, we try to control the high imbalance in $\max_k RV(P_k)$ that is likely to occur when minimizing solely $\max_k SV(P_k)$, and vice versa.

Apart from minimizing a single volume-based metric, our approach is very

flexible in the sense that it can address *any* combination of volume-based metrics simultaneously. This is achieved by simply associating even more weights with vertices. For instance, if one wishes to minimize $\max_k SV(P_k)$ and $\max_k RV(P_k)$ at the *same* time, it is enough to use two more weights in addition to the computational weight by setting $w^2(v_i) = sl(v_i)$ and $w^3(v_i) = rl(v_i)$ accordingly. Observe that one can utilize as many weights as desired with vertices. However, associating several weights with vertices does not come for free and has practical implications, which we address in the next section. Another important useful feature of our model is that, once the send and the receive loads are in hand, it is possible to define *custom* metrics regarding volume to best suit the needs of the target parallel application. For instance, although not sensible and just for demonstration purposes, one can address objectives like $\max_k \min\{SV(P_k), RV(P_k)\}$, $\max_k(SV(P_k)^2 + RV(P_k))$, etc. For our work, we have chosen the metrics which we believe to be the most crucial and definitive for a general application realized in message passing paradigm.

The arguments made so far are valid for the graph representation of symmetric matrices. To handle nonsymmetric matrices, it is necessary to modify the adjacency list definition by defining two adjacency lists for each vertex. This is because, the nonzeros $a_{i,j}$ and a_{ji} have different communication requirements in nonsymmetric matrices. Specifically, a nonzero a_{ji} signifies a send operation from P_k to P_ℓ no matter whether $a_{i,j}$ is nonzero or not, where v_i and v_j are respectively mapped to processors P_k and P_ℓ. Hence, the adjacency list definition regarding the send operations for v_i becomes $Adj_S(v_i) = \{v_j : a_{ji} \neq 0\}$. In a dual manner, a nonzero $a_{i,j}$ signifies a receive operation from P_ℓ to P_k no matter whether a_{ji} is nonzero or not. Thus, the adjacency list definition regarding the receive operations for v_i becomes $Adj_R(v_i) = \{v_j : a_{i,j} \neq 0\}$. Accordingly, in Algorithm 2, the adjacency lists in lines 4, 7, and 8 need to be replaced with $Adj_S(v_i)$, $Adj_R(v_i)$, and $Adj_S(v_j)$, respectively, to handle nonsymmetric matrices. Note that for all $v_i \in \mathcal{V}$, if the matrix is symmetric, then $Adj_S(v_i) = Adj_R(v_i) = Adj(v_i)$.

3.3.2.0.1 Complexity analysis Compared to the original RB-based graph partitioning model, our approach additionally requires computing and setting

volume loads (lines 5-6). Hence, we only focus on the runtime of these operations to analyze the additional cost introduced by our method. When we consider `GRAPH-COMPUTE-VOLUME-LOADS` for a single bipartitioning of graph G_k^r, the adjacency list of each boundary vertex ($Adj(v_i)$) in this graph is visited once. Note that although the lines 4 and 8 in this algorithm could be realized in a single for-loop, the computation of loads are illustrated with two distinct for-loops for the ease of presentation. In a single level of the RB tree (lines 4-9 of `GRAPH-PARTITION`), each edge $e_{i,j}$ of G is considered at most twice, once for computing loads of v_i, and once for computing loads of v_j. The efficient computation of $|Con(v_i)|$ in line 4 and $|Adj(v_j) \cap \mathcal{V}_k^r|$ in line 8 requires special attention. By maintaining an array of size $O(K)$ for each boundary vertex, we can retrieve these values in $O(1)$ time. In the computation of the send loads, the ℓth element of this array is one if v_i has neighbor(s) in $\mathcal{V}_\ell^r$, and zero otherwise. In the computation of the receive loads, it stands for the number of neighbors of v_i in $\mathcal{V}_\ell^r$. Since both of these operations can be performed in $O(1)$ time with the help of these arrays, the computation of volume loads in a single level takes $O(E)$ time in `GRAPH-PARTITION` (line 5). For lines 6 and 9, each vertex in a single level is visited only once, which takes $O(V)$ time. Hence, our method introduces an additional $O(V + E) = O(E)$ cost to each level of the RB tree. Note that $O(E) = O(nnz(A))$, where $nnz(A)$ is the number of nonzeros in A. The total runtime due to handling of volume-based loads thus becomes $O(E \lg_2 K)$. The space complexity of our algorithm is $O(V_B K)$ due to the arrays used to handle connectivity information of boundary vertices, where $V_B \subseteq \mathcal{V}$ denotes the set of boundary vertices in the final K-way partition. In practice $|\mathcal{V}_B|$ and K are much smaller than $|\mathcal{V}|$. In addition, for the send loads, these arrays contain only binary information which can be stored as bit vectors. Also note that the multi-constraint partitioning is expected to be costlier than its single-constraint counterpart.

3.3.3 Hypergraph model

Consider the use of the RB paradigm for partitioning the hypergraph representation $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ of A for row-parallel $Y = AX$ to obtain a K-way partition (Section 3.1.2) . Without loss of generality, we assume that the communication task represented by net n_i is performed by the processor that v_i is assigned to.

We assume that the assumptions made for the graph model also applies here so that we are at the stage of bipartitioning $\mathcal{H}_k^r$ for a given K'-way partition $\Pi(\mathcal{H})$. The hypergraph model for minimizing volume-based metrics resembles to the graph model. The only differences are the definitions regarding the send and receive loads of vertices. Recall that in the hypergraph model, n_i represents the communication task in which the processor that owns $v_i \in \mathcal{V}_k^r$ sends row i of X to the processors that correspond to the parts in $\Lambda(n_i) - \{\mathcal{V}_k^r\}$. So, in the hypergraph model, the connectivity set of vertex v_i is defined as the number of parts that n_i connects other than V_k^r, that is,

$$Con(v_i) = \{\mathcal{V}_\ell^t \in \Pi(\mathcal{H}) : Pins(n_i) \cap \mathcal{V}_\ell^t \neq \emptyset\} - \mathcal{V}_k^r.$$

Hence, in the hypergraph model, the send load $sl(v_i)$ of vertex v_i is given by

$$sl(v_i) = s \times |Con(v_i)| = s \times (\lambda(n_i) - 1).$$

Consider the communication task represented by a net n_j that connects $v_i \in \mathcal{V}_k^r$, where the vertex v_j associated with n_j is in $\mathcal{V}_\ell^t$. Recall that $\mathcal{V}_\ell^t$ is a part in $\Pi(\mathcal{H})$ other than V_k^r, where t is either r or $r+1$. For this task, the processor groups that correspond to the parts in $\Lambda(n_j) - \{\mathcal{V}_\ell^t\}$ receive row j of X from P_ℓ^t. This receive load of s words from P_ℓ^t to P_k^r is evenly distributed among the vertices in $Pins(n_j) \cap \mathcal{V}_k^r$. That is, n_j contributes $s/|Pins(n_j) \cap \mathcal{V}_k^r|$ amount to the receive load of v_i. Hence, the receive load $rl(v_i)$ of v_i is given by

$$rl(v_i) = \sum_{n_j \in Nets(v_i) - \{n_i\}} \frac{s}{|Pins(n_j) \cap \mathcal{V}_k^r|}.$$

Algorithm 4: `HYPERGRAPH-COMPUTE-VOLUME-LOADS`

Input: $\mathcal{H} = (\mathcal{V}, \mathcal{N}),\ \mathcal{H}_k^r = (\mathcal{V}_k^r, \mathcal{N}_k^r),\ part,\ s$

1 **foreach** boundary vertex $v_i \in \mathcal{V}_k^r$ **do**

$\triangleright$ Compute the send load

2 $Con(v_i) \leftarrow \emptyset$

3 **foreach** boundary vertex $v_j \in Pins(n_i)$ **and** $v_j \notin \mathcal{V}_k^r$ **do**

4 $Con(v_i) \leftarrow Con(v_i) \cup \{part[v_j]\}$

5 $sl(v_i) \leftarrow s \times |Con(v_i)|$

$\triangleright$ Compute the receive load

6 $rl(v_i) \leftarrow 0$

7 **foreach** $n_j \in Nets(v_i) - \{n_i\}$ **and** $v_j \notin \mathcal{V}_k^r$ **do**

8 $rl(v_i) \leftarrow rl(v_i) + s/|Pins(n_j) \cap \mathcal{V}_k^r|$

The remaining definitions regarding $SV(P_k^r)$, $RV(P_k^r)$ and the equivalence of minimization of the above-mentioned quantities with the defined metrics for the graph model hold as is for the hypergraph model. The algorithm `HYPERGRAPH-COMPUTE-VOLUME-LOADS` (Algorithm 4) computes the send and receive loads of vertices in the hypergraph model and resembles to that of graph model (Algorithm 2). In line 3 of this algorithm where we compute the send load of v_i, we traverse pin list of n_i instead of adjacency list of v_i. In line 7 where we compute the receive load of v_i, we traverse the nets that connect v_i instead of its adjacency list and in line 8, the receive load of v_i is updated by taking intersection of $\mathcal{V}_k^r$ with $Pins(n_j)$ instead of with $Adj(v_j)$. To compute a K-way partition of $\mathcal{H}$, Algorithm 3 can be used as is by replacing its graph terminology with the hypergraph terminology.

3.3.3.0.1 Complexity analysis The computation of volume loads in the hypergraph model differs from the graph model only in the sense that instead of visiting the adjacency lists of boundary vertices, the vertices connected by cut nets and the nets connecting boundary vertices are visited. Again, by associating an $O(K)$-size array with each boundary vertex, lines 4 and 8 in

HYPERGRAPH-COMPUTE-VOLUME-LOADS can be performed in $O(1)$ time. In the computation of the send loads, each vertex and the vertices connected by the net associated with that vertex are visited at most once in a single level of the RB tree. This requires visiting all vertices and pins of the hypergraph once in a single level in the worst case, which takes $O(V + P)$ time, where $P = \sum_{n \in \mathcal{N}} |Pins(n)|$. In the computation of the receive loads, each vertex and its net list are visited once. This also requires visiting all vertices and pins of the hypergraph once in a single level, which takes $O(V + P)$ time. Hence, our method introduces an additional $O(V + P) = O(P)$ cost to each level of the RB tree. Note that $O(P) = O(nnz(A))$. The total runtime due to handling of volume-based loads thus becomes $O(P \lg_2 K)$. The space complexity is $O(V_B K)$, where $V_B \subseteq \mathcal{V}$ denotes the set of boundary vertices in the final K-way partition. Observe that we introduce the same overhead both in graph and hypergraph models.

3.3.4 Partitioning tools

The multi-constraint graph and hypergraph partitioning tools associate multiple weights with vertices. These tools allow users to define different maximum allowed imbalance ratios $\epsilon^1, \ldots, \epsilon^C$ for each constraint, where ϵ^c denotes the maximum allowed imbalance ratio on the cth constraint. Recall that in our approach, minimizing the imbalance on a specific weight relates to minimizing the respective volume-based metric. Hence, by using the existing tools within our approach, it is possible to minimize the target volume-based metric(s).

The partitioning tools do not try to minimize the imbalance on a specific constraint. Rather, they aim to stay within the given threshold for any given ϵ^c. For this reason, the imbalance values provided to the tools should be as low as to the degree how much these metrics are important for optimization. Enforcing a very small value on ϵ^c can put a lot of strain on the partitioning tool, which in turn may cause the tool to intolerably loosen its objective. This may increase total volume drastically and make the minimization of target volume-based metrics pointless as they are defined on the amount of volume communicated. For this

reason, it is not sensible to use a very small value for ϵ^c.

3.4 Efficient handling of multiple constraints

In this section, we describe the two drawbacks of using multiple constraints within the context of our model and propose two practical schemes which enhance this model to overcome them.

Our approach introduces as many constraints as needed in order to address the desired volume-based cost metrics. Recall that the volume related weights are nonzero only for the boundary vertices because only these vertices incur communication. Since the objective of minimizing cutsize with partitioners also relates to minimizing the number of boundary vertices, only a small portion of all vertices will have nonzero volume related weights throughout the partitioning process. So, balancing the volume related weights of parts will have much less degree of freedom compared to balancing the computational weights of parts. That is, the partitioner will have difficulty in maintaining balance on volume-related weights of parts because of small number of vertices with nonzero volume-related weights.

Each introduced constraint puts an extra burden on the partitioning tool by restricting the solution space, where the more restricted the solution space, the worse the quality of the solutions generated by the partitioning tool. Hence, the additional constraint(s) used for minimizing volume-based metrics may lead to higher total volume (i.e., cutsize). This also has the side effect on the other factors that determine the overall communication cost, such as increasing contention on the network or increasing the latency overhead.

To address these shortcomings, in Section 3.4.1, we propose a scheme which selectively utilizes volume-related weights, and in Section 3.4.2, we propose another scheme which unifies multiple weights.

3.4.1 Delayed formation of volume loads

In this scheme, we utilize level information in the RB tree to form and make use of the volume related loads in a delayed manner. Specifically, in bipartitionings of the first ρ levels of the RB tree, we allow only a single constraint, i.e., regarding the computational load. In the remaining bipartitionings which belong to the latter $\lg_2 K - \rho$ levels, we consider volume-based metrics by introducing as many constraints as needed. This results in a level-based hybrid scheme in which either a single constraint or multiple constraints are utilized.

Our motivations for adopting this scheme are three-fold. First, we aim to improve the quality of the obtained solutions in terms of total volume by sacrificing from the quality of the volume-based metrics. Recall that the minimization of volume-based metrics is pointless unless the total volume is properly addressed. Next, the total volume changes as the partitioning progresses, and the volume-based metrics are defined over this changing quantity. As the ratio of boundary vertices increases in latter levels of the RB tree, addressing volume-based loads in bipartitionings of these levels leads to more efficient utilization of partitioners. Finally, utilization of volume-based loads in the latter levels rather than the earlier levels of the RB tree prevents the deviations on these loads which are likely to occur in the final solution if these constraints were utilized in the earlier levels rather than the latter levels.

This can be seen as an effort to achieve a tradeoff between minimizing total volume and minimizing target volume-based metrics. If we use multiple constraints in all bipartitionings, the target volume-based metrics will be optimized but the total obtained volume will be relatively high. On the other hand, if we use a single constraint (i.e., computational load), the total volume will be relatively low but the target metrics will not be addressed properly.

3.4.2 Unified weighting

In this scheme, we utilize only a *single* constraint by unifying multiple loads into a single load through a linear formula. Note that this scheme also refrains from the issue related with boundary vertices since the unified single weight for each vertex becomes almost always nonzero.

In order to use a single weight for vertices, it is required to establish a relation between distinct loads those are of interest. For SpMM, determining the relationship between the computational and communication loads is necessary to accurately estimate a single load for each vertex. In large-scale parallel architectures, per unit communication time is usually greater than per unit computation time. To unify the respective loads, we define a coefficient α that represents the per unit communication time in terms of per unit computation time. This coefficient depends on various factors such as clock rate, properties of the interconnect network, the requirements of the underlying parallel application, etc. The following code snippet constitutes the basic skeleton of the SpMM operations from processor P_k's point of view:

```
...
MPI_Irecv()
MPI_Send()
Perform local computations using A_kk
MPI_Waitall() // Wait all receives to complete
Perform non-local computations using A_kℓ, ℓ ≠ k
...
```

In this implementation, non-blocking receive operation is preferred to enable overlapping local SpMM computations $A_{kk}X_k$ and incoming messages. Blocking send operation is used since the performance gain from overlapping local computations and outgoing messages is very limited. The total load of a vertex v_i in this example can be captured with two distinct weights, where the first weight $w^1(v_i)$ and the second weight $w^2(v_i)$ respectively represent the computational load and the send load associated with v_i. The receive loads of vertices are neglected for this

implementation because of the non-blocking receive operations under the assumption that each processor has enough amount of local computation to overlap with the incoming messages. Then, with α in hand, we can easily unify these weights into a single weight as $w(v_i) = w^1(v_i) + \alpha w^2(v_i)$. Note that for non-boundary vertices $w(v_i) = w^1(v_i)$.

3.5 Experiments

3.5.1 Experimental setting

3.5.1.1 Datasets

We perform our experiments on three datasets. The first dataset is used to compare the proposed graph and hypergraph partitioning models (Sections 3.3 and 3.4) against the standard partitioning models (Section 3.1.2). Note that the standard models address only total volume. The second dataset is used to compare our models against the state-of-the-art partitioner `UMPa`, which addresses maximum send volume of processors. The third dataset is used to assess the strong and weak scaling performance of our models on multi-source breadth-first search (BFS) by comparing them against a recent two-dimensional partitioning model [11]. Tables 3.1, 3.2 and 3.3 describe the basic properties of these three datasets.

The first dataset is abbreviated as `ds-general` and contains all the square matrices from SuiteSparse Matrix Collection [47] (formerly known as the UFL Sparse Matrix Collection) with at least 5,000 rows/columns and between 50,000 and 50,000,000 nonzeros. At the time of the experiment, UFL had 964 such matrices, so `ds-general` contains 964 matrices. We categorized these matrices according to the maximum send volume of processors obtained by the standard partitioning models when they are partitioned among 128, 256, 512 and 1024 processors. The naming for the categories is in the format of *m*-K*k*-V*v*.

Table 3.1: Properties of the matrices in dataset `ds-general`. The values are the averages of the matrices in the respective category.

	matrix category	number of matrices	avg number of rows/cols	avg number of nonzeros	avg max row/col sparsity	matrix category	number of matrices	avg number of rows/cols	avg number of nonzeros	avg max row/col sparsity
categories obtained by G-TV	G-K128-V8000	50	303K	8,903K	0.0443	G-K512-V8000	32	350K	10,279K	0.0746
	G-K128-V4000	103	191K	5,513K	0.0233	G-K512-V4000	54	240K	8,296K	0.0598
	G-K128-V2000	167	175K	4,328K	0.0122	G-K512-V2000	98	179K	6,083K	0.0302
	G-K128-V1000	287	129K	3,067K	0.0099	G-K512-V1000	175	159K	4,304K	0.0146
	G-K128-V500	462	113K	2,487K	0.0056	G-K512-V500	329	108K	2,534K	0.0134
	G-K128-V250	646	83K	1,566K	0.0049	G-K512-V250	553	84K	1,769K	0.0084
	G-K128-V125	808	73K	1,227K	0.0040	G-K512-V125	709	73K	1,394K	0.0055
	G-K128-V0	964	60K	907K	0.0032	G-K512-V0	964	60K	907K	0.0032
	G-K256-V8000	40	303K	9,832K	0.0915	G-K1024-V8000	23	312K	9,907K	0.1171
	G-K256-V4000	73	195K	7,089K	0.0362	G-K1024-V4000	47	271K	8,053K	0.0867
	G-K256-V2000	135	182K	5,136K	0.0154	G-K1024-V2000	78	168K	5,885K	0.0628
	G-K256-V1000	200	149K	3,943K	0.0138	G-K1024-V1000	174	120K	3,216K	0.0360
	G-K256-V500	404	123K	2,851K	0.0063	G-K1024-V500	313	78K	1,520K	0.0330
	G-K256-V250	574	91K	1,874K	0.0057	G-K1024-V250	500	85K	1,823K	0.0105
	G-K256-V125	770	74K	1,279K	0.0045	G-K1024-V125	650	75K	1,502K	0.0068
	G-K256-V0	964	60K	907K	0.0032	G-K1024-V0	964	60K	907K	0.0032
categories obtained by H-TV	H-K128-V8000	50	185K	8,869K	0.0686	H-K512-V8000	39	155K	8,791K	0.0852
	H-K128-V4000	91	170K	6,253K	0.0250	H-K512-V4000	51	132K	7,726K	0.0848
	H-K128-V2000	167	166K	4,323K	0.0142	H-K512-V2000	89	165K	6,262K	0.0351
	H-K128-V1000	267	123K	3,122K	0.0133	H-K512-V1000	170	154K	4,383K	0.0135
	H-K128-V500	453	112K	2,478K	0.0059	H-K512-V500	303	111K	2,895K	0.0115
	H-K128-V250	624	86K	1,686K	0.0049	H-K512-V250	554	84K	1,700K	0.0078
	H-K128-V125	801	72K	1,246K	0.0041	H-K512-V125	696	74K	1,412K	0.0058
	H-K128-V0	964	60K	907K	0.0032	H-K512-V0	964	60K	907K	0.0032
	H-K256-V8000	41	152K	9,447K	0.0893	H-K1024-V8000	38	134K	6,529K	0.1061
	H-K256-V4000	69	155K	7,494K	0.0519	H-K1024-V4000	50	124K	6,564K	0.1042
	H-K256-V2000	128	164K	5,191K	0.0177	H-K1024-V2000	75	151K	6,240K	0.0567
	H-K256-V1000	214	135K	3,607K	0.0163	H-K1024-V1000	126	151K	5,090K	0.0267
	H-K256-V500	380	117K	2,808K	0.0071	H-K1024-V500	322	83K	1,654K	0.0261
	H-K256-V250	556	93K	1,965K	0.0053	H-K1024-V250	508	81K	1,644K	0.0103
	H-K256-V125	753	72K	1,300K	0.0048	H-K1024-V125	653	75K	1,470K	0.0067
	H-K256-V0	964	60K	907K	0.0032	H-K1024-V0	967	60K	907K	0.0032

Here, $m \in \{\texttt{G},\texttt{H}\}$ denotes the model, where $\texttt{m} = \texttt{G}$ if the partitions are obtained by the standard graph model and $\texttt{m} = \texttt{H}$ if the partitions are obtained by the standard hypergraph model. k denotes the number of processors, where $k \in \{128, 256, 512, 1024\}$. v denotes the lower bound for the maximum send volume obtained, where $v \in \{8000, 4000, 2000, 1000, 500, 250, 125, 0\}$. For example, `G-K512-V1000` denotes the set of matrices for which the standard graph model obtains a maximum send volume of at least 1000 units for $K = 512$ processors. Here and hereafter one unit of communication refers to an X-matrix row that contains s words. Our motivations for this categorization are two-fold: (i) to categorize the matrices according to their likelihood for which the maximum send volume is a bottleneck for parallel performance and (ii) to facilitate a better performance analysis of the proposed models. Table 3.1 displays various important properties of the matrices in those categories as the geometric averages. Note that the categories for the standard graph and hypergraph models are different as the maximum send volume values obtained by them are typically different. Also note that m-Kk-V8000 $\subseteq$ m-Kk-V4000 $\subseteq \cdots \subseteq$ m-Kk-V0 and m-Kk-V0 contains all the matrices in `ds-general`, for any m and k. "avg max row/col sparsity" column of Table 3.1 denotes the maximum number of nonzeros in a row or column divided by the number of rows/columns in the respective matrix, averaged over all the matrices in the respective category.

For the comparison against `UMPa`, we run our models on the matrices in Table 1 of [42], the work that propose `UMPa`. We use these matrices for our experiments since `UMPa` is not publicly available. The matrices in this dataset are obtained from 10th DIMACS Implementation Challenge [48] and contains 38 matrices from eight different classes. We exclude two synthetic matrices, namely `1,280,000` and `320,000`, in our experiments as the communicated items in these matrices have different sizes and the communication load formulation utilized in our models does not support varying sizes of communicated items. We abbreviate the resulting dataset of 36 matrices as `ds-dimacs`. For the matrices in this dataset, the number of rows/columns is between 100,000 and 1,585,478 and the number of nonzeros is between 119,666 and 38,354,076. The properties of the matrices in this dataset are given in Table 3.2.

Table 3.2: Properties of the matrices in dataset `ds-dimacs`.

matrix name	number of rows/cols	number of nonzeros	class	matrix name	number of rows/cols	number of nonzeros	class
citationCiteseer	268K	2,313K	Citation	rgg_n_2_19_s0	524K	6,540K	RandomGeometric
coAuthorsCiteseer	227K	1,628K	Citation	rgg_n_2_20_s0	1,049K	13,783K	RandomGeometric
coAuthorsDBLP	299K	1,955K	Citation	af_shell10	1,508K	52,672K	Sparse
coPapersCiteseer	434K	32,073K	Citation	af_shell9	505K	17,589K	Sparse
coPapersDBLP	540K	30,491K	Citation	audikw_1	944K	77,652K	Sparse
caidaRouterLevel	192K	1,218K	Clustering	ecology1	1,000K	4,996K	Sparse
cnr-2000	326K	3,216K	Clustering	ecology2	1,000K	4,996K	Sparse
eu-2005	863K	19,235K	Clustering	G3_circuit	1,585K	7,661K	Sparse
G_n_pin_pout	100K	1,002K	Clustering	ldoor	952K	46,522K	Sparse
in-2004	1,383K	16,917K	Clustering	thermal2	1,228K	8,580K	Sparse
pref.Att.	100K	1,000K	Clustering	belgium_osm	1,441K	3,100K	Street
smallworld	100K	1,000K	Clustering	luxembourg_osm	115K	239K	Street
delaunay_n17	131K	786K	Delaunay	144	145K	2,149K	Walshaw
delaunay_n18	262K	1,573K	Delaunay	598a	111K	1,484K	Walshaw
delaunay_n19	524K	3,146K	Delaunay	auto	449K	6,629K	Walshaw
delaunay_n20	1,049K	6,291K	Delaunay	fe_ocean	143K	819K	Walshaw
rgg_n_2_17_s0	131K	1,458K	RandomGeometric	m14b	215K	3,358K	Walshaw
rgg_n_2_18_s0	262K	3,095K	RandomGeometric	wave	156K	2,119K	Walshaw

Table 3.3: Properties of the matrices in `ds-large`.

matrix name	number of rows/cols	number of nonzeros	problem kind
arabic-2005	22,744K	640,000K	directed graph
nlpkkt240	27,994K	760,648K	optimization problem
uk-2005	39,460K	936,364K	directed graph
webbase-2001	118,142K	1,019,903K	directed graph
it-2004	41,292K	1,150,725K	directed graph

The third dataset is abbreviated as `ds-large` and contains five of the six largest matrices in the SuiteSparse Matrix Collection [47]. The properties of the matrices in this dataset are given Table 3.3. These are larger than the matrices in `ds-general` and `ds-dimacs`, with the number of rows/columns between 22.7 million and 118.1 million and the number of nonzeros between 640 million and 1.15 billion. We experiment with larger number of processors up to 2048 on this dataset.

3.5.1.2 Implementation and parallel systems

The parallel SpMM and multi-source BFS kernels are implemented within a parallel library [49] in C that uses MPI for interprocess communication. We use two systems in our experiments. The first system is a Cray XC40 machine. A node on this machine consists of 24 cores (two 12-core Intel Haswell Xeon processors) with 2.5 GHz clock frequency and 128 GB memory. The nodes are connected with a high speed Dragonfly network topology called CRAY Aries. The second system is used for scalability analysis and is a Lenovo NeXtScale machine. A node on this machine consists of 28 cores (two 14-core Intel Haswell Xeon processors) with 2.6 GHz clock frequency and 64 GB memory. The nodes are connected with an Infiniband non-blocking tree network topology.

The proposed models can be realized with any graph and hypergraph partitioning tool. For models that rely on using multiple constraints (i.e., the proposed model and its delayed version), the tool should support multiple weights on vertices. In our experiments, we used Metis [32] for partitioning graphs and PaToH [19] for partitioning hypergraphs, both in default settings. Metis and PaToH are respectively used to bipartition the graphs and hypergraphs in line 7 of Algorithm 3.

3.5.1.3 Compared schemes and models

We evaluate six proposed schemes that address the total volume and the maximum send volume simultaneously. Each of these schemes considers the minimization of the maximum send volume of processors, (i.e., $\max_k SV(P_k)$) in accordance with the discussions given in Section 3.4.2. These schemes are as follows:

- `G-TMV`: The proposed graph partitioning model which addresses both `T`otal and `M`aximum `V`olume metrics (Section 3.3.2). This scheme utilizes two weights, one for the computational loads and one for the send volume loads.
- `H-TMV`: The hypergraph counterpart of `G-TMV` (Section 3.3.3).

- `G-TMVd`: The variant of `G-TMV` with the send loads formed in a *delayed* manner (Section 3.4.1). ρ is set to $\lceil \lg_2 K/2 \rceil$.
- `H-TMVd`: The hypergraph counterpart of `G-TMVd`.
- `G-TMVu`: The variant of `G-TMV` with *unified* loads (Section 3.4.2). The coefficient that determines the relation between computational and communication loads is set to $\alpha = 10$.
- `H-TMVu`: The hypergraph counterpart of `G-TMVu`.

The baseline models that we compare our proposed models against are as follows:

- `G-TV`: The standard graph partitioning model which only addresses `T`otal `V`olume (Section 3.1.2). Recall that this is the most widely adopted model in the literature for sparse matrix/graph partitioning and there exists a single weight, which is on the computational loads. This scheme refers to the use of Metis as is for K-way partitioning.
- `H-TV`: The standard hypergraph partitioning model which only addresses `T`otal `V`olume (Section 3.1.2). This scheme refers to the use of PaToH as is for K-way partitioning.
- `UMPa`: The state-of-the-art partitioner that can minimize multiple communication cost metrics [42]. In our case, we consider $\texttt{UMPa}_{\text{MSV}}$ in which the single objective is to minimize the maximum send volume handled by a processor.
- `2D`: The two-dimensional partitioning model proposed for parallel level-synchronized BFS [11]. Although this method considers the single-source BFS, it is trivially extended to the multi-source BFS.

For the proposed schemes and the baseline schemes `G-TV` and `H-TV`, we use 10% as the maximum allowed imbalance for each of the constraints. Hereinafter,

we will refer to the maximum send volume of processors simply as maximum volume.

In Section 3.5.2, we compare the six proposed schemes among themselves and against standard models `G-TV` and `H-TV`. In Section 3.5.3, we compare the best performing proposed scheme for graph and hypergraph models against `UMPa`. In Section 3.5.4, we compare these best schemes against `2D` to analyze their scalability on parallel multi-source BFS.

3.5.2 Comparison against standard partitioning models

3.5.2.1 Partitioning results

In this section, we provide the results of the proposed partitioning schemes for both graph and hypergraph models in terms of maximum volume, total volume, maximum number of messages and total number of messages, on dataset `ds-general` following the categorization described in Section 3.5.1.1. Although bandwidth-related metrics are expected to be more important for parallel SpMM performance, latency-related metrics such as maximum and total number of messages can still affect the performance for certain matrices, hence we include them in our analysis. The results obtained by `G-TMV`, `G-TMVd` and `G-TMVu` are normalized with respect to those of `G-TV` and the results obtained by `H-TMV`, `H-TMVd` and `H-TMVu` are normalized with respect to those of `H-TV`. The obtained normalized values for $K = 1024$ processors are displayed as plots for each of the four metrics seperately for graph and hypergraph models, respectively in Figures 3.3 and 3.4. The detailed results for each $K \in \{128, 256, 512, 1024\}$ are given in Tables 3.4 and 3.5. In the plots in Figures 3.3 and 3.4, the x-axis denotes the eight respective matrix categories in order of increasing maximum volume whereas the y-axis denotes the normalized values. Each value in the plots and the tables is the geometric average of the normalized values obtained for the matrices in the respective category. For example, the three values reported for category `G-K1024-V2000` in the plot for maximum volume for graph model (the top left

one in Figure 3.3) denote the geometric averages of the normalized values obtained by G-TMV, G-TMVd and G-TMVu with respect to those obtained by G-TV on 78 matrices in this category.

As seen in Figure 3.3, G-TMV, G-TMVd and G-TMVu perform better than G-TV in terms of maximum volume. These three schemes perform drastically better in the matrices for which maximum volume is a bottleneck for performance. The improvements obtained by all three schemes increase with increasing significance of maximum volume. For example, for category G-K1024-V500, G-TMV, G-TMVd and G-TMVu respectively achieve improvements of 21%, 21% and 14%, whereas for category G-K1024-V8000, these improvements increase to respectively, 61%, 59% and 39%. In addition, for category G-K1024-Vv with varying v values of 0, 125, 250, 500, 1000, 2000, 4000 and 8000, the improvement of G-TMV over G-TV gradually increases as 5%, 10%, 14%, 21%, 33%, 48%, 56% and 61%, respectively. The reason for this increase in the improvement is that there exists more room for improvement in partitioning the matrices for which the standard graph model yields high maximum volume.

When we compare G-TMV, G-TMVd and G-TMVu among themselves, G-TMV usually obtains the best improvements in maximum volume since it addresses this metric as a stand-alone objective during the entire partitioning process. In this sense, it differs from G-TMVd and G-TMVu, where G-TMVd addresses maximum volume only in latter bipartitionings and G-TMVu, in order to address maximum volume, uses a single unified constraint that also involves the computational load. Compared to G-TV, G-TMV causes an increase of 17% and 7% in total volume for G-K1024-V0 and G-K1024-V8000, respectively, and G-TMVd causes an increase of 15% and 4%. This is due to the additional constraint utilized in G-TMV and G-TMVd. Recall that utilizing multiple constraints degrades the quality of the solutions obtained by the partitioner in terms of total volume. The reason for smaller degradation rates in total volume for categories with high maximum volume can be attributed to the matrices in these categories having a high total volume, which leaves less room for degradation as a significant fraction of the edges were already in the cut in the partitions obtained by the standard model. G-TMVu, which is proposed to remedy this problem, does not increase the total

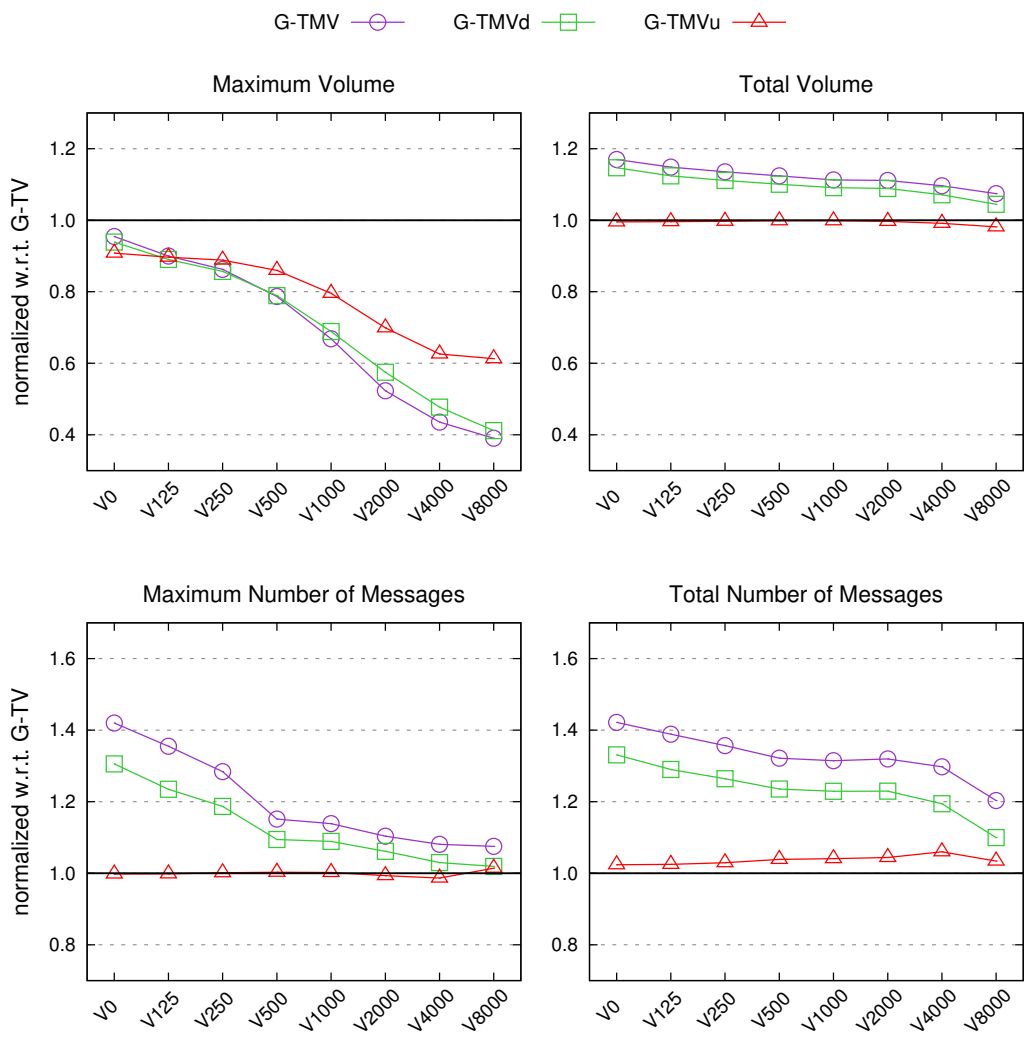


Figure 3.3: Maximum volume, total volume, maximum number of messages and total number of messages of the proposed graph schemes `G-TMV`, `G-TMVd` and `G-TMVu` normalized with respect to those of `G-TV` for $K = 1024$, averaged on matrices in each category `G-K1024-V`υ.

volume in contrast to G-TMV and G-TMVd, and attains comparable results with G-TV in this metric.

As seen in Figure 3.3, G-TMV obtains the worst results in terms of maximum number of messages, followed by G-TMVd. These two schemes in this metric respectively cause increases of 42% and 33% in category G-K1024-V0, and 20% and 10% in category G-K1024-V8000 over G-TV. G-TMVu on the other hand causes a very slight increase over G-TV. The same observations hold for the total number of messages as well. Observe that while maximum volume is substantially improved by G-TMVu, other important factors that determine the communication time such as maximum and total number of messages and total volume are kept almost intact.

As seen in Figure 3.4, most of the above observations and discussions made for the graph model hold for the hypergraph model as well. In the hypergraph model for maximum and total volume, the improvement and deterioration rates of the proposed schemes over H-TV are magnified compared to those of the graph models over G-TV. For example, for category H-K1024-V500, H-TMV, H-TMVd and H-TMVu respectively achieve improvements of 28%, 25% and 26%, whereas for category H-K1024-V8000, these improvements become 78%, 66% and 63%. H-TMV causes increases of 28% and 14% in total volume for H-K1024-V0 and H-K1024-V8000, respectively, and H-TMVd causes increases of 25% and 11%. H-TMVu on the other hand obtains slightly smaller total volume than H-TV. In terms of maximum number of messages, H-TMV and H-TMVd perform worse than H-TV while H-TMVu performs slightly better. In terms of total number of messages, all three schemes perform worse than H-TV.

As seen in Tables 3.4 and 3.5, the improvements of the proposed models in maximum volume increase as the number of processors increases. For example, for category G-Kk-V2000 with increasing number of k processors $128, 256, 512$ and 1024, the improvements of G-TMVu over G-TV respectively increase as 19%, 25%, 27% and 30%. The improvements of H-TMVu over H-TV for the same setting respectively increase as 18%, 25%, 38% and 51%. The reason for the better performance of the proposed models in larger number of processors is the increased

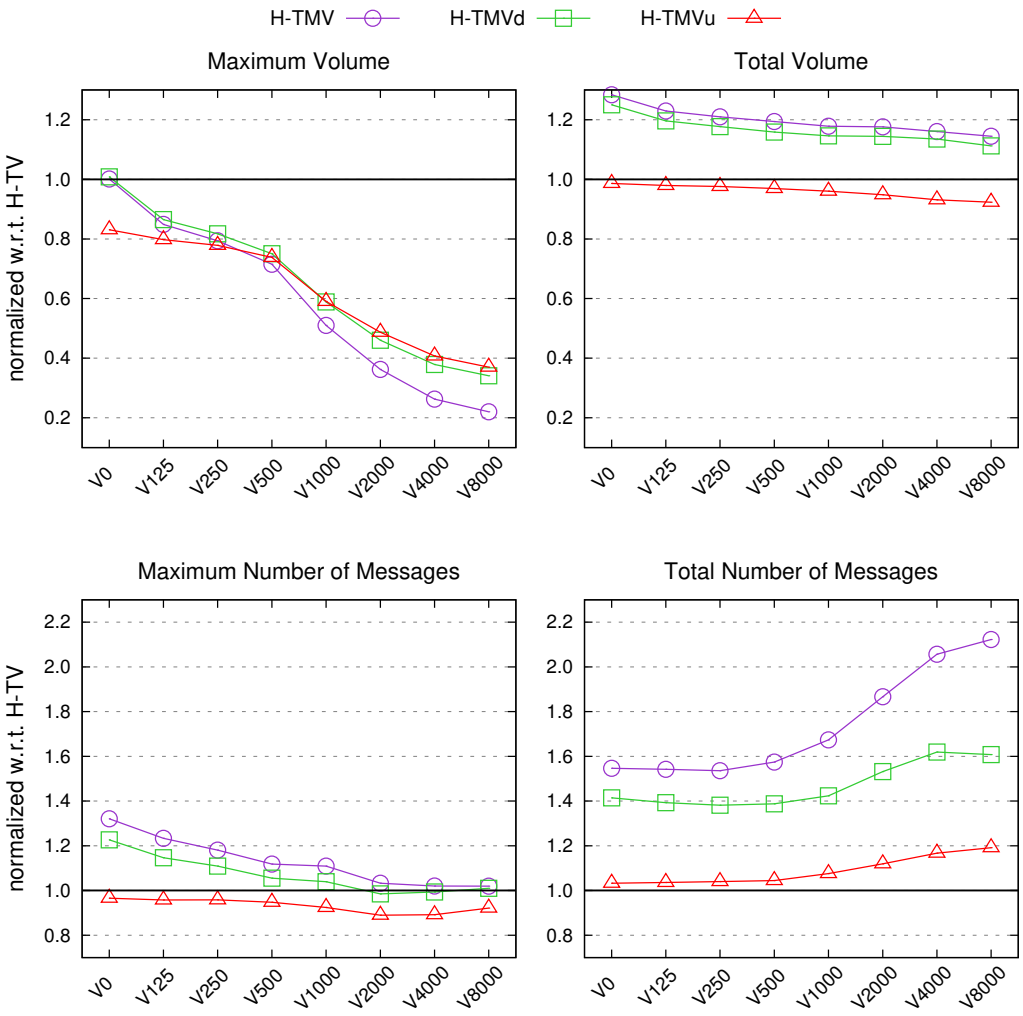


Figure 3.4: Maximum volume, total volume, maximum number of messages and total number of messages of the proposed hypergraph schemes `H-TMV`, `H-TMVd` and `H-TMVu` normalized with respect to those of `H-TV` for $K = 1024$, averaged on matrices in each category `H-K1024-V`υ.

Table 3.4: Normalized values of maximum volume, total volume, maximum number of messages and total number of messages of the proposed graph models `G-TMV`, `G-TMVd` and `G-TMVu` with respect to those of `G-TV` for $K \in \{128, 256, 512, 1024\}$.

		normalized values w.r.t those of `G-TV`											
		max volume			total volume			max message			total message		
K	category	`G-TMV`	`G-TMVd`	`G-TMVu`	`G-TMV`	`G-TMVd`	`G-TMVu`	`G-TMV`	`G-TMVd`	`G-TMVu`	`G-TMV`	`G-TMVd`	`G-TMVu`
128	`G-K128-V8000`	0.69	0.74	0.73	1.10	1.07	1.00	1.05	1.03	0.99	1.22	1.14	1.04
	`G-K128-V4000`	0.73	0.82	0.76	1.12	1.09	0.99	1.09	1.04	1.00	1.27	1.17	1.03
	`G-K128-V2000`	0.78	0.85	0.81	1.10	1.08	0.99	1.13	1.06	1.02	1.29	1.16	1.05
	`G-K128-V1000`	0.84	0.88	0.85	1.10	1.07	0.99	1.16	1.09	1.01	1.28	1.16	1.03
	`G-K128-V500`	0.86	0.89	0.87	1.11	1.08	0.99	1.18	1.10	1.01	1.31	1.19	1.03
	`G-K128-V250`	0.89	0.90	0.88	1.12	1.09	0.99	1.24	1.12	1.01	1.35	1.21	1.03
	`G-K128-V125`	0.91	0.92	0.89	1.13	1.10	1.00	1.30	1.16	1.01	1.40	1.24	1.03
	`G-K128-V0`	0.95	0.95	0.90	1.16	1.12	1.00	1.43	1.24	1.00	1.48	1.29	1.02
256	`G-K256-V8000`	0.54	0.69	0.65	1.09	1.07	0.99	1.07	1.02	1.01	1.26	1.16	1.08
	`G-K256-V4000`	0.63	0.73	0.68	1.12	1.08	0.99	1.10	1.05	1.00	1.31	1.18	1.04
	`G-K256-V2000`	0.70	0.81	0.75	1.10	1.07	0.99	1.12	1.07	1.00	1.31	1.17	1.04
	`G-K256-V1000`	0.76	0.84	0.79	1.10	1.07	0.99	1.13	1.05	0.99	1.30	1.16	1.03
	`G-K256-V500`	0.82	0.86	0.85	1.11	1.08	0.99	1.21	1.10	1.00	1.32	1.18	1.03
	`G-K256-V250`	0.86	0.88	0.87	1.12	1.08	0.99	1.24	1.10	0.99	1.35	1.20	1.03
	`G-K256-V125`	0.90	0.90	0.89	1.13	1.09	0.99	1.30	1.13	0.99	1.39	1.22	1.03
	`G-K256-V0`	0.95	0.94	0.90	1.16	1.11	1.00	1.43	1.21	0.99	1.47	1.27	1.02
512	`G-K512-8000`	0.49	0.59	0.68	1.09	1.06	0.99	1.09	1.03	1.00	1.27	1.13	1.06
	`G-K512-V4000`	0.52	0.60	0.67	1.11	1.07	0.99	1.10	1.04	0.99	1.30	1.17	1.08
	`G-K512-V2000`	0.60	0.70	0.73	1.11	1.08	0.99	1.13	1.07	0.99	1.35	1.21	1.06
	`G-K512-V1000`	0.68	0.74	0.78	1.11	1.08	0.99	1.15	1.08	1.00	1.32	1.21	1.04
	`G-K512-V500`	0.77	0.80	0.84	1.12	1.09	0.99	1.23	1.12	1.01	1.33	1.21	1.04
	`G-K512-V250`	0.85	0.86	0.88	1.13	1.10	0.99	1.28	1.15	1.00	1.38	1.24	1.03
	`G-K512-V125`	0.89	0.89	0.89	1.14	1.11	0.99	1.34	1.19	0.99	1.40	1.26	1.03
	`G-K512-V0`	0.94	0.93	0.90	1.17	1.13	0.99	1.43	1.27	0.99	1.46	1.31	1.02
1024	`G-K1024-V8000`	0.39	0.41	0.61	1.07	1.04	0.98	1.08	1.02	1.01	1.20	1.10	1.03
	`G-K1024-V4000`	0.44	0.48	0.63	1.10	1.07	0.99	1.08	1.03	0.99	1.30	1.19	1.06
	`G-K1024-V2000`	0.52	0.57	0.70	1.11	1.09	1.00	1.10	1.06	0.99	1.32	1.23	1.04
	`G-K1024-V1000`	0.67	0.69	0.80	1.11	1.09	1.00	1.14	1.09	1.00	1.31	1.23	1.04
	`G-K1024-V500`	0.79	0.79	0.86	1.12	1.10	1.00	1.15	1.09	1.00	1.32	1.24	1.04
	`G-K1024-V250`	0.86	0.86	0.89	1.14	1.11	1.00	1.28	1.19	1.00	1.36	1.26	1.03
	`G-K1024-V125`	0.90	0.89	0.90	1.15	1.12	1.00	1.36	1.23	1.00	1.39	1.29	1.02
	`G-K1024-V0`	0.95	0.94	0.91	1.17	1.15	1.00	1.42	1.31	1.00	1.42	1.33	1.02

Table 3.5: Normalized values of maximum volume, total volume, maximum number of messages and total number of messages of the proposed hypergraph models `H-TMV`, `H-TMVd` and `H-TMVu` with respect to those of `H-TV` for $K \in \{128, 256, 512, 1024\}$.

		normalized values w.r.t those of `H-TV`											
		max volume			total volume			max message			total message		
K	category	`H-TMV`	`H-TMVd`	`H-TMVu`	`H-TMV`	`H-TMVd`	`H-TMVu`	`H-TMV`	`H-TMVd`	`H-TMVu`	`H-TMV`	`H-TMVd`	`H-TMVu`
128	`H-K128-V8000`	0.50	0.66	0.72	1.15	1.10	0.97	1.07	1.03	0.99	1.58	1.24	1.02
	`H-K128-V4000`	0.62	0.73	0.79	1.16	1.10	0.98	1.09	1.06	1.00	1.48	1.22	1.02
	`H-K128-V2000`	0.71	0.80	0.82	1.15	1.11	0.98	1.13	1.07	1.00	1.49	1.23	1.03
	`H-K128-V1000`	0.78	0.85	0.85	1.17	1.12	0.99	1.15	1.09	0.99	1.46	1.24	1.02
	`H-K128-V500`	0.85	0.89	0.87	1.19	1.15	0.99	1.21	1.11	1.00	1.48	1.26	1.02
	`H-K128-V250`	0.89	0.91	0.87	1.21	1.16	0.99	1.25	1.13	0.99	1.49	1.27	1.02
	`H-K128-V125`	0.93	0.95	0.89	1.24	1.19	0.99	1.30	1.16	1.00	1.51	1.29	1.03
	`H-K128-V0`	1.01	1.01	0.90	1.28	1.23	0.99	1.35	1.20	1.00	1.54	1.32	1.02
256	`H-K256-V8000`	0.38	0.53	0.65	1.16	1.12	0.96	1.06	1.04	0.98	1.84	1.41	1.04
	`H-K256-V4000`	0.50	0.62	0.72	1.18	1.13	0.97	1.10	1.07	0.98	1.76	1.43	1.04
	`H-K256-V2000`	0.61	0.69	0.75	1.18	1.15	0.98	1.12	1.08	0.99	1.61	1.38	1.04
	`H-K256-V1000`	0.71	0.77	0.79	1.17	1.13	0.98	1.16	1.09	0.99	1.54	1.36	1.04
	`H-K256-V500`	0.79	0.83	0.82	1.19	1.16	0.98	1.19	1.11	0.99	1.52	1.36	1.03
	`H-K256-V250`	0.84	0.86	0.84	1.21	1.17	0.98	1.24	1.14	0.98	1.51	1.36	1.03
	`H-K256-V125`	0.90	0.91	0.86	1.23	1.20	0.99	1.27	1.16	0.98	1.52	1.37	1.03
	`H-K256-V0`	1.00	1.00	0.88	1.28	1.24	0.99	1.35	1.22	0.98	1.55	1.40	1.03
512	`H-K512-V8000`	0.32	0.50	0.52	1.15	1.11	0.95	1.05	1.04	0.95	1.95	1.43	1.09
	`H-K512-V4000`	0.34	0.50	0.53	1.19	1.14	0.96	1.05	1.03	0.94	2.00	1.51	1.11
	`H-K512-V2000`	0.47	0.60	0.62	1.19	1.14	0.97	1.08	1.05	0.96	1.78	1.44	1.09
	`H-K512-V1000`	0.64	0.74	0.73	1.18	1.14	0.98	1.14	1.08	0.96	1.60	1.35	1.05
	`H-K512-V500`	0.73	0.79	0.76	1.19	1.15	0.98	1.17	1.09	0.97	1.56	1.34	1.04
	`H-K512-V250`	0.83	0.86	0.82	1.21	1.17	0.98	1.20	1.11	0.97	1.52	1.33	1.03
	`H-K512-V125`	0.87	0.89	0.83	1.23	1.19	0.98	1.25	1.13	0.98	1.53	1.34	1.03
	`H-K512-V0`	1.00	1.01	0.85	1.28	1.23	0.99	1.33	1.20	0.98	1.55	1.36	1.03
1024	`H-K1024-V8000`	0.22	0.34	0.37	1.14	1.11	0.92	1.02	1.01	0.92	2.12	1.61	1.19
	`H-K1024-V4000`	0.26	0.38	0.41	1.16	1.14	0.93	1.02	0.99	0.89	2.06	1.62	1.17
	`H-K1024-V2000`	0.36	0.46	0.49	1.18	1.14	0.95	1.03	0.98	0.89	1.87	1.53	1.12
	`H-K1024-V1000`	0.51	0.59	0.59	1.18	1.15	0.96	1.11	1.04	0.92	1.67	1.42	1.08
	`H-K1024-V500`	0.72	0.75	0.74	1.19	1.16	0.97	1.12	1.06	0.95	1.57	1.39	1.04
	`H-K1024-V250`	0.79	0.82	0.78	1.21	1.18	0.98	1.18	1.11	0.96	1.54	1.38	1.04
	`H-K1024-V125`	0.85	0.86	0.80	1.23	1.20	0.98	1.23	1.15	0.96	1.54	1.39	1.04
	`H-K1024-V0`	1.00	1.01	0.83	1.28	1.25	0.99	1.32	1.23	0.97	1.55	1.41	1.03

number of bipartitions in which our model is applied throughout the recursive bipartitioning process.

Table 3.6: Comparison of partitioning times averaged over 964 matrices.

		actual (miliseconds)				normalized w.r.t. G-TV		
model	K	G-TV	G-TMV	G-TMVd	G-TMVu	G-TMV	G-TMVd	G-TMVu
graph	128	597	1307	1192	636	2.19	2.00	1.06
	256	801	2136	1897	865	2.67	2.37	1.08
	512	1143	3148	2933	1251	2.75	2.57	1.09
	1024	1073	3662	3502	1184	3.41	3.26	1.10
		actual (miliseconds)				normalized w.r.t. H-TV		
model	K	H-TV	H-TMV	H-TMVd	H-TMVu	H-TMV	H-TMVd	H-TMVu
hypergraph	128	5601	5225	5124	5939	0.93	0.91	1.06
	256	6602	5973	6786	6758	0.90	1.03	1.02
	512	7720	7012	8249	7933	0.91	1.07	1.03
	1024	8932	7956	9161	9669	0.89	1.03	1.08

Table 3.6 displays the partitioning times of the compared schemes for the graph and hypergraph models. For each $K \in \{128, 256, 512, 1024\}$, we present the actual time and the normalized time (with respect to G-TV for graph schemes G-TMV, G-TMVd and G-TMVu, and H-TV for hypergraph schemes H-TMV, H-TMVd and H-TMVu), which are both geometric averages of 964 matrices. Recall that the proposed schemes introduce the same partitioning overhead of $O(nnz(A))$ in both models. This overhead can be extracted from the normalized values of G-TMVu over G-TV and H-TMVu over H-TV, and is only 6%–10% for the graph model and 2%–8% for the hypergraph model. For the graph model, among the proposed schemes, G-TMVu introduces the lowest partitioning overhead compared to G-TV. The worse performances of G-TMV and G-TMVd compared to G-TMVu are expected since multi-constraint partitioning is more expensive than single-constraint partitioning because of the additional feasibility conditions. Although one expects the same for the hypergraph model, the multi-constraint partitioning times of PaToH are surprisingly better than those of single-constraint partitioning.

Judging from the partitioning results, G-TMVu among the graph schemes and H-TMVu among the hypergraph schemes always achieve significant reductions in maximum volume while keeping the change in other three metrics as small as possible compared to G-TV and H-TV, respectively. For this reason, we only consider G-TMVu and H-TMVu among the proposed schemes in the rest of the experimentation.

3.5.2.2 Parallel SpMM runtime results

We have run parallel SpMM [49] on a Cray XC40 machine for 128, 256 and 512 processors, with $s = 10$. Due to the quota limitations on our core-hours on this system, we have tested the performance of G-TMVu over G-TV and H-TMVu over H-TV for 30 test matrices. Whenever we use the phrase "parallel SpMM with G-TMVu/H-TMVu/G-TV/H-TV", we refer to the parallel SpMM execution when the matrices in SpMM are partitioned with G-TMVu/H-TMVu/G-TV/H-TV.

Table 3.7 presents the parallel SpMM runtime results with G-TMVu and H-TMVu normalized with respect to those of G-TV and H-TV, respectively. The 30 test matrices are a subset of dataset ds-general with 964 matrices whose partitioning results are given in the previous section.

Observe that the improvements obtained by G-TMVu and H-TMVu in maximum volume (Tables 3.4 and 3.5) are reflected upon the parallel SpMM runtimes. In most instances, these two schemes lead to a lower runtime compared to the standard models. On the average, parallel SpMM with G-TMVu runs 4%, 11% and 14% faster than parallel SpMM with G-TV, whereas parallel SpMM with H-TMVu runs 14%, 13% and 22% faster than parallel SpMM with H-TV for 128, 256 and 512 processors, respectively. There are two important observations that can be inferred from these results. (i) The improvements in parallel SpMM runtimes attained both by G-TMVu and H-TMVu increase with increasing number of processors. This can be attributed to the increased importance of communication costs with increasing number of processors. (ii) H-TMVu attains higher improvements in parallel SpMM runtime compared to G-TMVu. This conforms with the experimental

Table 3.7: Parallel SpMM runtimes attained by `G-TMVu` and `H-TMVu` normalized with respect to parallel SpMM runtimes attained by `G-TV` and `H-TV`, respectively, for 128, 256 and 512 processors.

matrix name	number of rows/cols	number of nonzeros	problem kind	G-TMVu			H-TMVu		
				128	256	512	128	256	512
144	145K	2,149K	undirected graph	1.08	0.78	0.78	0.81	0.89	1.01
598a	111K	1,484K	undirected graph	1.02	0.85	0.88	0.81	0.89	0.84
bauru5727	40K	145K	eigenvalue/model reduction	0.97	0.70	0.73	0.75	0.73	0.83
bcsstk25	40K	145K	structural	1.16	1.16	0.84	0.97	0.91	0.63
big	13K	92K	directed weighted graph	0.76	0.87	0.78	1.07	1.00	0.90
bips07_3078	21K	76K	eigenvalue/model reduction	0.87	0.69	0.93	0.94	0.88	0.77
bodyy4	18K	122K	structural	1.03	0.92	1.00	0.74	1.00	0.90
chipcool0	20K	281K	model reduction	1.03	0.76	0.96	1.18	0.92	0.95
copter1	17K	211K	computational fluid dynamics	0.81	0.88	1.09	1.30	1.04	0.96
ford1	19K	102K	structural	0.97	1.00	0.71	0.70	0.76	1.05
fv1	10K	85K	2D/3D	1.57	0.90	1.00	1.00	1.11	1.00
hcircuit	106K	513K	circuit simulation	0.96	0.93	0.80	0.64	1.01	1.02
hvdc1	25K	158K	power network	0.86	0.82	0.73	1.00	1.03	1.03
jan99jac040	14K	73K	economic	0.56	0.91	0.90	0.97	0.64	0.68
juba40k	40K	145K	eigenvalue/model reduction	0.98	1.13	0.55	1.18	1.08	0.50
lhr11	11K	232K	chemical process simulation	0.83	0.82	0.52	0.95	0.80	0.50
m14b	215K	3,358K	undirected graph	1.00	0.85	0.92	0.95	0.90	0.70
offshore	260K	4,243K	electromagnetics	0.98	0.93	0.80	0.92	0.87	0.91
OPF_3754	15K	142K	power network	1.00	0.88	1.14	1.08	0.80	0.67
pattern1	19K	9,323K	optimization	0.77	0.81	0.65	0.62	0.67	0.60
pds10	17K	150K	optimization	0.93	1.09	0.94	1.21	0.83	0.75
pesa	12K	80K	directed weighted graph	0.97	0.74	0.90	0.76	1.00	0.86
rail_20209	20K	139K	model reduction	1.17	1.00	1.18	1.03	0.90	0.94
ri2010	25K	126K	undirected weighted graph	0.95	0.83	0.85	0.83	0.96	0.83
skirt	13K	197K	structural	1.42	0.78	1.10	0.80	1.04	1.00
std1_Jac2	22K	1,248K	chemical process simulation	0.82	0.97	0.62	0.47	0.63	0.54
tandem_vtx	19K	253K	structural	1.31	0.98	1.09	0.93	0.82	1.14
TSOPF_RS_b2383	38K	16,171K	power network	0.92	1.01	1.01	0.62	0.82	0.37
xingo3012	21K	74K	eigenvalue/model reduction	0.93	0.86	0.79	0.96	1.00	0.63
Zd_Jac3	23K	1,916K	chemical process simulation	0.85	0.92	1.00	0.45	0.54	0.69
			geomean	**0.96**	**0.89**	**0.86**	**0.86**	**0.87**	**0.78**
			best	0.56	0.69	0.52	0.45	0.54	0.37
			worst	1.57	1.16	1.18	1.30	1.11	1.14

finding that H-TMVu attains higher improvements in maximum volume compared to G-TMVu as also seen in Tables 3.4 and 3.5.

We analyze the scalability of parallel SpMM with G-TMVu and H-TMVu in Figure 3.5 on 10 matrices by respectively comparing them against those with G-TV and H-TV. The results of each matrix are grouped for graph and hypergraph model. Each bar chart in the figure belongs to a different matrix and indicates the parallel SpMM runtime obtained with the respective scheme and the number of processors. The three consecutive bars for each scheme and each matrix denote the respective parallel SpMM runtimes obtained on 128, 256 and 512 processors. These matrices are chosen in such a way that they illustrate and summarize different scalability characteristics of both schemes. For matrices that already scale well with G-TV and H-TV such as 144, 598a, bauru5727 and m14b, G-TMVu and H-TMVu almost always lead to lower SpMM runtimes for all K values and improve the scalability. For matrices such as bcsstk25, juba40k and pattern1, which have an elbow while moving from 256 to 512 processors, communication costs become a bottleneck and hinder scalability. Addressing the right bottleneck for these matrices via G-TMVu and H-TMVu pays off with improved scalability by the decreased runtimes with increasing number of processors. For harder instances such as lhr11, although none of the two schemes scales, G-TMVu and H-TMVu are still able to reduce the parallel SpMM runtime drastically. For example, for lhr11 on 512 processors, G-TMVu and H-TMVu respectively lead to 48% and 50% better SpMM runtimes compared to G-TV and H-TV.

3.5.3 Comparison against UMPa

In this section, we compare our models against UMPa [42] and present the results in Table 3.8. Each instance reported in the table is the geometric average of the results of five partitioning runs. The comparison is performed in terms of the partition quality and the partitioning time. The partition quality is measured in terms of the maximum volume in number of words and reported as the actual values for UMPa in the second column (as reported in [42]). The third and fourth

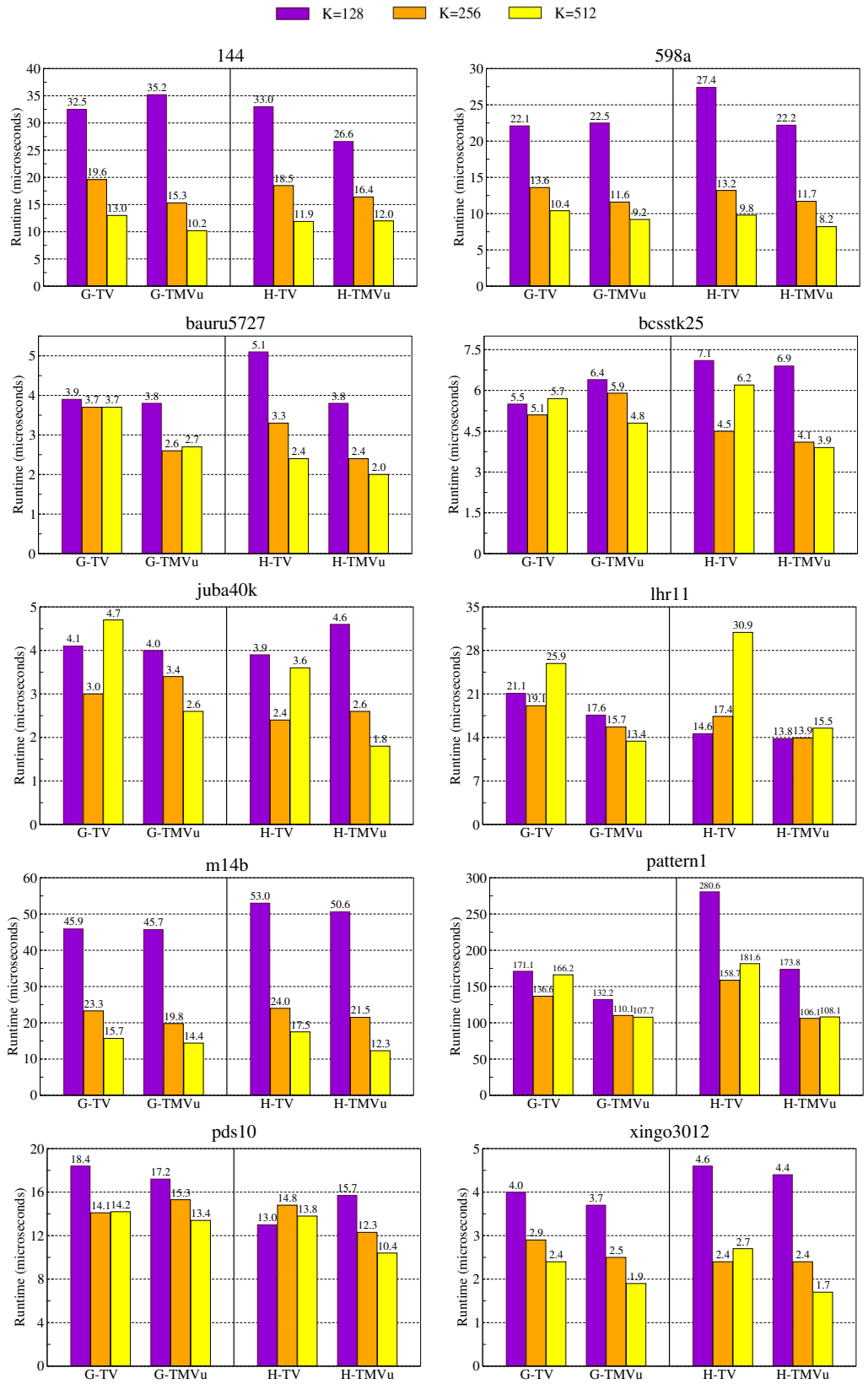


Figure 3.5: Strong scaling analysis of parallel SpMM with `G-TMVu` and `H-TMVu` schemes compared to those with `G-TV` and `H-TV`, respectively.

Table 3.8: Comparison of G-TMVu and H-TMVu against UMPa in terms of maximum volume (number of words communicated) and partitioning time for $K = 512$ processors. The matrices are sorted according to the maximum volume values obtained by UMPa.

		max volume			partitioning time w.r.t. PaToH		
		actual	norm. w.r.t. UMPa				
name		UMPa	G-TMVu	H-TMVu	UMPa	G-TMVu	H-TMVu
eu-2005		8544	1.18	0.27	6.68	0.18	1.05
coPapersDBLP		7229	1.03	0.96	3.40	0.10	0.91
in-2004		4962	0.74	0.60	2.83	0.17	0.87
preferentialAttachment		3938	0.45	1.62	8.48	0.33	1.27
coPapersCiteseer		3927	0.83	0.78	1.56	0.07	1.06
cnr-2000		3096	0.41	0.32	6.93	0.22	1.05
caidaRouterLevel		2932	0.29	0.29	6.93	0.38	1.32
audikw_1		2655	1.04	1.04	1.57	0.09	1.03
citationCiteseer		2003	1.06	1.08	6.64	0.26	1.24
coAuthorsDBLP		1489	0.71	0.61	8.72	0.44	1.35
G_n_pin_pout		1283	0.96	1.12	7.01	0.40	1.46
auto		717	1.22	1.17	6.96	0.21	1.34
coAuthorsCiteseer		688	0.85	0.89	7.17	0.44	1.33
af_shell10		621	1.03	0.97	1.13	0.20	1.10
ldoor		582	0.98	0.98	1.61	0.14	1.07
m14b		501	1.01	0.99	6.94	0.23	1.42
max volume of UMPa > 500	**avg:**	–	**0.81**	**0.76**	**4.35**	**0.21**	**1.17**
	impr:	–	**19%**	**24%**	–	**20.7x**	**3.7x**
smallworld		497	0.93	0.93	7.84	0.50	1.35
G3_circuit		406	1.02	0.99	5.40	0.45	1.41
144		402	0.89	0.86	6.64	0.25	1.48
wave		375	1.09	1.10	6.87	0.27	1.51
af_shell9		352	1.06	1.04	1.61	0.20	1.16
598a		327	1.04	1.02	6.93	0.27	1.48
rgg_n_2_20_s0		253	1.17	1.02	2.31	0.30	1.25
thermal2		248	0.98	0.96	3.12	0.39	1.31
fe_ocean		232	1.29	1.01	7.36	0.45	1.95
delaunay_n20		209	1.02	1.01	3.49	0.43	1.36
ecology1		203	1.16	0.97	4.58	0.51	1.45
ecology2		201	1.18	1.01	4.58	0.50	1.45
rgg_n_2_19_s0		172	1.17	1.01	3.51	0.31	1.26
delaunay_n19		147	1.02	1.00	4.38	0.43	1.41
rgg_n_2_18_s0		123	1.13	0.99	4.06	0.33	1.31
delaunay_n18		108	0.99	0.97	5.15	0.42	1.46
rgg_n_2_17_s0		85	1.13	1.01	5.70	0.35	1.34
delaunay_n17		77	1.01	0.97	6.05	0.42	1.47
belgium_osm		65	1.29	1.29	2.06	1.03	1.72
luxembourg_osm		17	1.21	1.10	4.83	1.14	1.90
overall	**avg:**	–	**0.95**	**0.89**	**4.40**	**0.30**	**1.31**
	impr:	–	**5%**	**11%**	–	**14.5x**	**3.4x**

columns display the maximum volume values obtained by G-TMVu and H-TMVu as normalized values with respect to those of UMPa, respectively. The last three columns display the partitioning times of the compared models as normalized values with respect to the runtime of the partitioner PaToH [19] in default setting. The rows of the table are sorted with respect to the maximum volume obtained by UMPa and are divided into two according to the matrices for which UMPa obtains a maximum volume higher/lower than 500 words.

Both G-TMVu and H-TMVu are able to obtain better quality partitions than UMPa. On the average, G-TMVu and H-TMVu obtain improvements of 5% and 11% in maximum volume compared to UMPa, respectively. For the matrices for which UMPa obtains higher maximum volume, i.e., at least 500 words, the improvements attained by G-TMVu and H-TMVu are more apparent: 19% for G-TMVu and 24% for H-TMVu. Recall that for such matrices, maximum volume is more likely to be a critical factor in determining the overall performance. The examples for such matrices are seen in classes such as "Citation" and "Clustering" (see the classes of the matrices in Table 3.2), for which both G-TMVu and H-TMVu perform significantly better than UMPa in terms of partition quality. These are usually hard instances that are scale-free. In the remaining classes, G-TMVu produces slightly worse quality partitions, while H-TMVu produces comparable quality partitions.

Both G-TMVu and H-TMVu are significantly faster than UMPa. The average partitioning time of UMPa is 4.40x that of PaToH, whereas the average partitioning times of G-TMVu and H-TMVu are respectively 0.30x and 1.31x that of PaToH. As a result, remarkably, G-TMVu is on the average 14.5x faster than UMPa. H-TMVu is 3.4x faster than UMPa.

3.5.4 Scalability analysis

We thoroughly evaluate the scalability of the multi-source level-synchronized BFS kernel executed on the five graphs in dataset ds-large. We compare our models G-TMVu and H-TMVu against 2D [11] in terms of parallel runtime of multi-source BFS and communication statistics. The runtimes reported in this

section are the execution times of parallel multi-source BFS on these graphs and not the partitioning times. Whenever we use the phrase "parallel BFS with `G-TMVu`/`H-TMVu`/`2D`", we refer to the parallel BFS execution when the vertices/edges of the input graph are partitioned using `G-TMVu`/`H-TMVu`/`2D`. We investigate the scalability performance on the multi-source BFS and not on SpMM since `2D` was originally proposed for the former and all three models exhibit similar behaviors in both. There are four different number of processors, $K \in \{256, 512, 1024, 2048\}$ and four different number of source vertices, $s \in \{5, 10, 20, 40\}$ in our experiments. Recall that the number of source vertices in the multi-source BFS is equivalent to the number of columns of input dense matrix X in SpMM. We investigate both strong and weak scaling performances of multi-source BFS with these three models. The experiments were performed on the Lenovo NeXtScale system.

We present the strong scaling results in Figure 3.6. Each row in the figure belongs to a different graph and each column belongs to a different s value. Each plot contains three lines comparing `G-TMVu`, `H-TMVu` and `2D` for a specific graph and s. The x-axis and y-axis respectively denote the number of processors and the runtime of the operations in a single level of parallel multi-source BFS in miliseconds. Both axes are in logarithmic scale.

As seen in the plots in Figure 3.6, for all instances, both parallel BFS with `G-TMVu` and parallel BFS with `H-TMVu` run much faster than parallel BFS with `2D`. For example, for 256, 512, 1024 and 2048 processors, parallel BFS with `G-TMVu` respectively runs 5.3x, 6.9x, 8.0x and 10.8x faster than `2D`, for $s = 20$, on the average. Again for $s = 20$ and for the same numbers of processors, parallel BFS with `H-TMVu` respectively runs 6.6x, 8.4x, 10.3x and 10.3x faster than `2D`. This is simply because the communication cost of parallel multi-source BFS is largely dominated by the bandwidth costs and our models aim at reducing bandwidth-related metrics total and maximum volume, whereas `2D` only aims to provide an upper bound on latency-related metrics. This results in our models to achieve lower communication overhead and hence better performance. The scalability

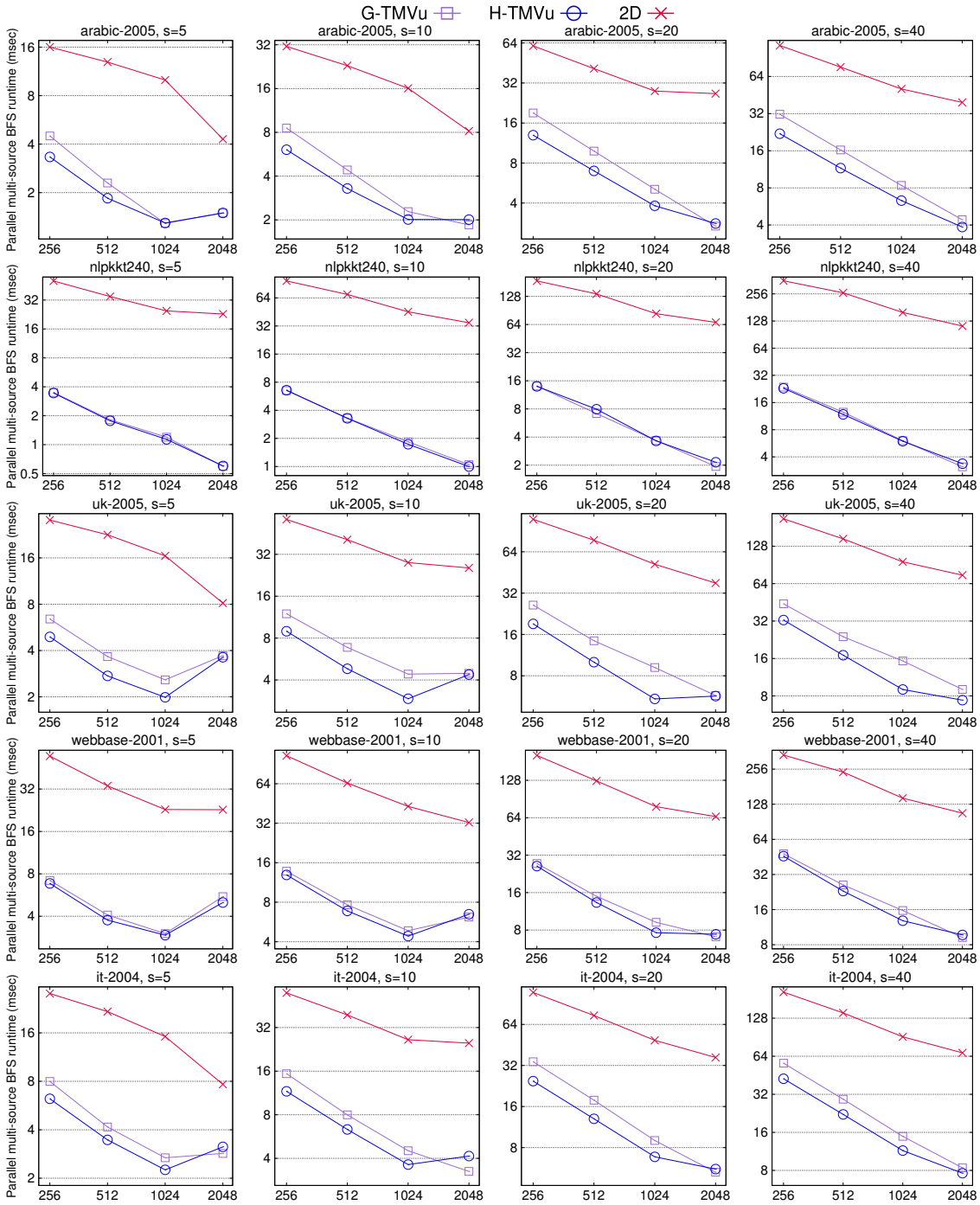


Figure 3.6: Strong scaling analysis of parallel multi-source BFS with `G-TMVu`, `H-TMVu` and `2D`. The x-axis denotes the number of processors.

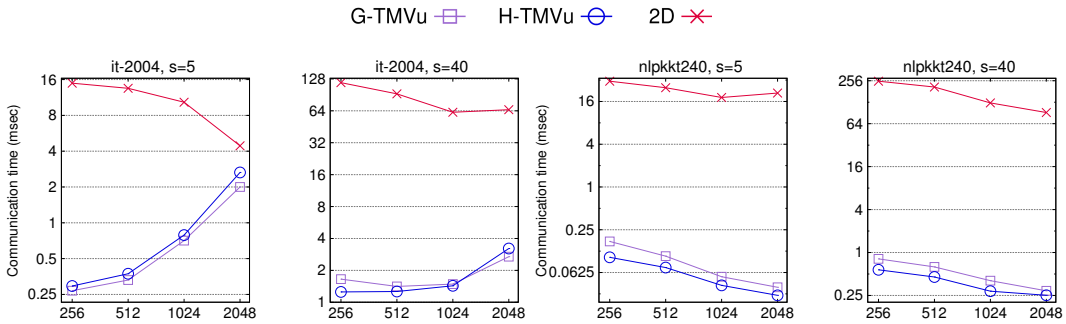


Figure 3.7: Communication times in parallel multi-source BFS with `G-TMVu`, `H-TMVu` and `2D` for `it-2004` and `nlpkkt240` both with $s \in \{5, 40\}$. The x-axis denotes the number of processors.

of our models becomes more apparent with increasing s. The performance gap between `2D` and our models in terms of scalability turns into favor of our models with increasing s. For example, for `it-2004`, compared to `2D`, parallel BFS with `G-TMVu` runs 3.5x faster on 256 processors and 2.7x faster on 2048 processors for $s = 5$, whereas for $s = 40$ it runs 3.6x faster on 256 processors and 8.1x faster on 2048 processors. Similar improvements are observed for parallel BFS with `H-TMVu` as well, where these two values are 4.5x and 2.4x for $s = 5$ and they are 4.8x and 9.0x for $s = 40$. The close performances of `G-TMVu` and `H-TMVu` for `nlpkkt240` are due to the regular sparsity pattern of this matrix which hides the flaw of the graph model [44, 19] to a large extent.

We investigate the communication performance of parallel BFS with `G-TMVu`, `H-TMVu` and `2D`. The volume and message count statistics obtained by these models are given in Table 3.9 for $s = 5$ and $s = 40$, and for 256, 512, 1024 and 2048 processors. Both the volume and message count statistics include maximum and average values. We focus on two graphs `it-2004` and `nlpkkt240` since `it-2004` is the largest graph in `ds-large` and for the other graphs in `ds-large` except `nlpkkt240`, we observe similar findings with those for `it-2004`. Figure 3.7 illustrates variation of the communication times with varying number of processors for parallel BFS.

In all instances, both of our models obtain lower communication times than

Table 3.9: Volume and message count statistics of `it-2004` and `nlpkkt240` for $s = 5$ and $s = 40$. Note that message count statistics are the same for $s = 5$ and $s = 40$.

volume statistics (in megabytes)

		$s = 5$						$s = 40$					
		maximum			average			maximum			average		
graph	K	2D	G-TMVu	H-TMVu	2D	G-TMVu	H-TMVu	2D	G-TMVu	H-TMVu	2D	G-TMVu	H-TMVu
it-2004	256	37.71	0.83	0.75	37.45	0.16	0.11	301.70	6.63	6.00	299.61	1.31	0.84
	512	21.11	0.71	0.57	20.86	0.09	0.06	168.85	5.64	4.59	166.89	0.75	0.49
	1024	13.93	0.64	0.32	13.70	0.06	0.04	111.45	5.14	2.59	109.63	0.50	0.30
	2048	7.70	0.50	0.52	7.48	0.05	0.03	61.58	3.98	4.15	59.81	0.38	0.27
nlpkkt240	256	54.53	0.64	0.48	54.44	0.47	0.35	436.26	5.15	3.86	435.52	3.72	2.80
	512	33.45	0.39	0.31	33.38	0.29	0.23	267.63	3.10	2.50	267.07	2.32	1.83
	1024	19.84	0.25	0.21	19.77	0.18	0.15	158.69	2.03	1.69	158.19	1.48	1.21
	2048	11.08	0.16	0.15	11.02	0.12	0.10	88.62	1.31	1.21	88.15	0.95	0.79

message count statistics

		maximum			average		
graph	K	2D	G-TMVu	H-TMVu	2D	G-TMVu	H-TMVu
it-2004	256	30	245	240	30	172	139
	512	46	463	442	46	267	184
	1024	62	724	731	62	347	192
	2048	94	1171	1488	94	324	231
nlpkkt240	256	30	20	22	30	12	12
	512	46	23	21	46	13	13
	1024	62	22	23	62	14	14
	2048	94	21	25	94	14	14

2D. The significantly better performance of our models can be explained with the significant reductions obtained in both maximum and average volume, as seen in Table 3.9. For example, for 512 processors and $s = 40$, the maximum volume values obtained by 2D for 512 processors and $s = 40$ are 168.85 MB and 267.63 MB for it-2004 and nlpkkt240, respectively, whereas these two values are 5.64 MB and 3.10 MB for G-TMVu, respectively, and 4.59 MB and 2.50 MB for H-TMVu, respectively. There are similar significant reductions in average volume.

For it-2004, it is seen from the left two plots in Figure 3.7 that the performance gap between G-TMVu/H-TMVu and 2D is much higher for $s = 40$ than the gap for $s = 5$, especially with larger number processors. This is mainly because the volume-based metrics for $s = 40$ are more determinant in communication times compared to $s = 5$, as message count statistics do not change while volume statistics increase with increasing s. For example, the average volume obtained by 2D for 1024 processors is 13.70 MB for $s = 5$ while it is 109.63 MB for $s = 40$ and the average message count is 62 regardless of s. For $s = 5$, the decrease in the performance gap between our models and 2D for larger number of processors can be explained by the increased importance of latency-related metrics in communication time and since 2D model provides an upper bound on the maximum and total message counts, it achieves a lower latency overhead compared to G-TMVu and H-TMVu.

Compared to it-2004, nlpkkt240 exhibits a more regular structure as it is obtained by PDE discretization while it-2004 is a web graph. This can be seen in Table 3.9 by comparing the maximum or average message counts obtained by our models. For example, for 1024 processors, the maximum message count obtained by G-TMVu for it-2004 is 724 while it is only 22 for nlpkkt240. The regular structure of nlpkkt240 is successfully exploited by our models as G-TMVu and H-TMVu always obtain lower maximum and average message counts than 2D. As opposed to it-2004, there always exists a big performance gap between our models and 2D regardless of s and number of processors since both G-TMVu and H-TMVu perform much better in terms of both bandwidth and latency costs.

The different behavior of G-TMVu and H-TMVu for it-2004 and nlpkkt240 can

be explained by the varying importance of latency costs in communication times. For both of these graphs, with increasing number of processors, maximum and average volume values tend to decrease, however maximum and average message count values for `it-2004` increase while they remain the same for `nlpkkt240`.

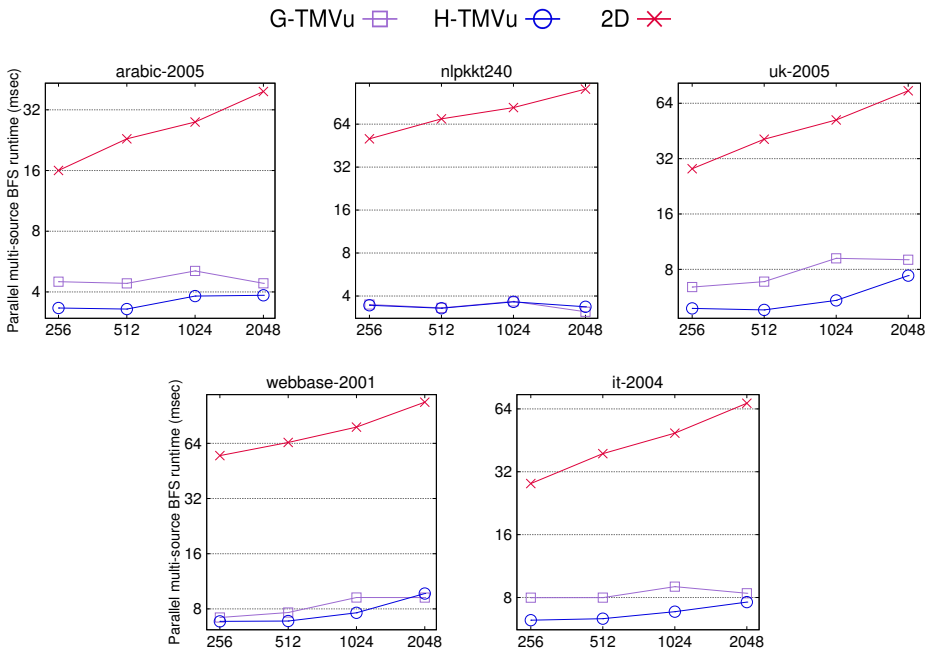


Figure 3.8: Weak scaling analysis of parallel multi-source BFS with `G-TMVu`, `H-TMVu` and `2D`. The x-axis denotes the number of processors.

We present the weak scaling results in Figure 3.8. To keep the computational load of each processor fixed when we double the number of processors, we double the number of source vertices while using the same input graph. In this way, when we double the number of processors, we double the total amount of computation while keeping the structure of the input graph same. Ideally, the number of edges assigned to each processor is halved when the number of processors is doubled. In other words, we use $s = 5$ at 256 processors, $s = 10$ at 512 processors, etc. Using different number of source vertices for the BFS enables us to seamlessly perform weak scaling analysis. The five plots in Figure 3.8 show that our models exhibit superior weak scaling performance compared to `2D`. This is mainly because the communication costs incurred by our models tend to increase less than those incurred by `2D` when the number of processors is doubled. The lines that belong

to our models in these plots sometimes have a negative slope when the number of processors is doubled. This behavior can be attributed to the following two reasons: (i) unstable computational load imbalances in the partitions obtained with the partitioners, leading to number of edges owned by the processors to not always halve when the number of processors is doubled, and (ii) the increased cache utilization in local computations due to the good reorderings generated by the partitioners. Nonetheless, it can be said that our models more than often exhibit consistently good weak scaling performance overall.

3.6 Conclusion

This work aimed to improve the performance of sparse matrix dense matrix multiplication on distributed memory systems. We addressed the high communication volume requirements of this kernel by proposing graph and hypergraph partitioning models which can minimize multiple volume-based communication cost metrics simultaneously in a single partitioning phase. Relying on a formulation that makes use of multiple constraints in recursive bipartitioning framework, we additionally proposed two practical schemes to efficiently utilize the existing partitioning tools. The experiments performed with this kernel and a level-synchronized multi-source parallel breadth-first search kernel on a large-scale high performance computing system up to 2048 processors validate the benefits of optimizing multiple volume-based metrics via our models by improving scalability.

As future work, we plan to try out different orders for bipartitionings in recursive bipartitioning. Moreover, we also consider using other partitioners to realize our models. Among them, Scotch [35] is the first to consider due to its high quality partitions in terms of load balance.

Chapter 4

Improving performance of sparse matrix vector multiplication on large-scale parallel systems

Sparse matrix vector multiplication (SpMV), which is denoted by $y = Ax$, is one of the most common operations which arise in many scientific and engineering problems. In iterative applications such as iterative solvers, SpMV is usually performed once at each iteration. Therefore, performance of SpMV is quite crucial to the overall performance of the application.

The irregularity of the coefficient matrix A plays an important role in the parallelization of SpMV for distributed-memory systems, since the distribution of the nonzero entries greatly affect the runtime cost of parallel SpMV. This cost consists of two major components:

- computational cost: the cost of performing the computation operations,
- communication cost: the cost of exchanging data among processors.

The unit of the computational cost in SpMV is an individual multiply-and-add operation performed on a nonzero entry of A, i.e., $y_i \leftarrow y_i + a_{i,j}x_j$. Hence,

the computational cost of the processor which is assigned the maximum number of nonzero entries equate to the computational cost of the parallel SpMV. The communication cost is usually assumed to have the following components:

- bandwidth cost: the cost of transferring messages,
- latency cost: the start-up cost of preparing messages.

The bandwidth cost is proportional to the number of words transmitted among processors, i.e., the communication volume, whereas the latency cost is proportional to the number of messages. The distribution of the nonzeros of input coefficient matrix A as well as the distribution of the entries of input and output vectors x and y determine both of these costs.

For an effective optimization of the runtime cost of parallel SpMV, the computational cost should be minimized by assigning balanced computational loads (i.e., number of nonzeros) to processors and the communication cost should be minimized by considering both of bandwidth and latency costs. For these minimization purposes, graph and hypergraph partitioning models have been utilized. The partitioning models in [31, 19, 32, 33, 22, 44, 45, 50, 51, 52, 53] minimize the computational and the bandwidth costs while disregarding the latency cost. Among these models, the hypergraph-based ones have a correct encoding for the minimization objective of the bandwidth cost while the graph-based ones only relate to that objective.

There are numerous different hypergraph partitioning models that address the bandwidth cost. These models basically vary on the dimensionality and the granularity of the partition. In the coarse-grain models [19], an atomic task is defined as the multiplication of a row/column of A by vector x. The distribution of the tasks induce a rowwise/columnwise one-dimensional (1D) partitioning of A. In the checkerboard and jagged models [22, 54], an atomic task is defined as the multiplication of a subset of nonzeros in a row/column of A by the corresponding x-vector entries. In the fine-grain model [55], an atomic task is defined as the multiplication of a single nonzero entry of A by the corresponding x-vector entry.

The checkerboard, jagged and fine-grain models all obtain a two-dimensional (2D) partition of A. Note that the fine-grain model defines the tasks in the smallest possible granularity. Hence among the mentioned models, the fine-grain model is the most flexible one in distributing the nonzeros of A and has the largest solution space for the minimization objective. The fine-grain model having the largest solution space results in the lowest communication volume and the imbalance on the computational loads of the processors. Therefore, the lowest bandwidth and computational costs for parallel SpMV is attained by this partitioning model.

The flexibility of independently assigning the tasks defined on nonzero entries comes with the expense of losing the row/column coherences of the nonzeros. Thus, the fine-grain model usually results in a larger number of messages, i.e., high latency cost, compared to the other models. In this work, we propose a hypergraph partitioning model based on the fine-grain model, where the partitioning objective of minimizing the cutsize simultaneously addresses both the bandwidth and latency costs.

Since each nonzero entry of A introduces a different task in the fine-grain model, it has the largest partitioning cost among all partitioning models due the larger number of tasks. A recent work, the medium-grain model [52] proposed for partitioning sparse matrices, remedies this problem by obtaining partitions with a quality comparable to that of fine-grain model in a faster manner. This model involves a heuristic in order to group the nonzeros of A in a more intelligent way compared to the coarser-grain models. The partitioning cost of the medium-grain model is less than that of the fine-grain model and comparable to those of the coarse-grain models, hence, this model is a good alternative to the fine-grain model. Hence, we propose another hypergraph partitioning model based on the medium-grain model, with the objective of minimizing both the bandwidth and latency costs.

The proposed models utilize the recursive bipartitioning paradigm in order to encode the messages with a special type of nets called *message nets*. The nets of the original fine- and medium-grain models are called *volume nets*, and the proposed models utilize both types of nets to reduce the bandwidth and latency costs

simultaneously. The model proposed in [1] uses message nets for applications with a single communication phase and conformal input and output partitions. In the current work, the proposed models optimize the two communication phases found in the parallel SpMV partitioned with fine-/medium-grain model. Moreover, the proposed models are able to produce nonconformal input and output partitions (i.e., the partitions of x and y vectors) as well as conformal ones. The experiments conducted on a dataset containing almost one thousand matrices show that the proposed models drastically improve the latency costs of the original fine-grain and medium-grain models. The experiments also show that compared to the baseline model proposed in [1], the proposed models obtain better results in volume-related metrics and computational imbalance. The runtime results of parallel SpMV validate the proposed models.

The rest of the chapter is organized as follows. Section 4.1 gives background on parallel SpMV with two-dimensional partitioning and original fine-grain and medium-grain hypergraph partitioning models. Sections 4.2 and 4.3 present the proposed models based on fine-grain and medium-grain models, respectively. Section 4.4 presents two techniques for further improving the quality of the partitions obtained by the proposed models. Section 4.5 gives the experimental results and Section 4.6 concludes the chapter.

4.1 Background

4.1.1 Parallel SpMV with two-dimensional sparse matrix partitioning

Consider an SPMV operation denoted by $y = Ax$, where A is a sparse matrix of size $n_r \times n_c$. $a_{i,j}$ denotes the entry in the ith row and the jth column of A, whereas r_i and c_j respectively denote the ith row and the jth column of A. x_j and y_i denote the jth entry of vector x and the ith entry of vector y, respectively.

Let $\mathcal{A}$ denote the set of nonzero entries in matrix A. That is,

$$A = \{a_{i,j} : a_{i,j} \neq 0\}.$$

Similarly, let $\mathcal{X}$ and $\mathcal{Y}$ denote the set of entries in input vector x and output vector y, respectively. That is,

$$\mathcal{X} = \{x_1, x_2, \ldots, x_{n_c}\} \text{ and}$$

$$\mathcal{Y} = \{y_1, y_2, \ldots, y_{n_r}\}.$$

Assume that there are K processors denoted by $P_1, P_2, \ldots, P_K$ in the parallel system. Let $\Pi_K(\mathcal{A}) = \{\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_K\}$, $\Pi_K(\mathcal{X}) = \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_K\}$ and $\Pi_K(\mathcal{Y}) = \{\mathcal{Y}_1, \mathcal{Y}_2, \ldots, \mathcal{Y}_K\}$ denote K-way partitions of sets A, x and y, respectively.

The set of nonzeros in A_k are assigned to processor P_k, for each $1 \leq k \leq K$. P_k performs the computation task of the multiply-and-add operation in the form of $y_i \leftarrow y_i + a_{i,j}x_j$ for each $a_{i,j} \in \mathcal{A}_k$. The vector entries in $\mathcal{X}_k$ and $\mathcal{Y}_k$ are also assigned to P_k. The partitions on vectors x and y, i.e., $\Pi_K(\mathcal{X})$ and $\Pi_K(\mathcal{Y})$, are said to be conformal if $n_r = n_c$ and the same processor owns x_i and y_i for each $1 \leq i \leq n_r = n_c$. Otherwise, the vector partitions are said to be non-conformal.

There are two communication phases in each iteration of parallel SpMV. One of them is performed before the computations of the multiply-and-add operations and called the *pre-communicaton* phase. The aim of this phase is to communicate the x-vector entries so that each processor P_k has the updated x-vector entry x_j for each $a_{i,j}$ in $\mathcal{A}_k$. The owner of an x-vector entry sending it to possibly multiple processors is called the *expand* operation on the respective x-vector entry. The other communication phase is performed after the computations and called the *post-communication* phase. The aim of this phase is to communicate the partial results computed for y-vector entries so that the owner processor of each y-vector entry y_i has all partial results computed for y_i. The owner of a y-vector entry receiving partial results for it from possibly multiple processors is called the *fold* operation on the respective y-vector entry. An outline of the operations performed

Algorithm 5: An SpMV iteration performed by processor P_k

Require: $\mathcal{A}_k$ and $\mathcal{X}_k$

1: $\triangleright$ *Pre-communication phase — Expands on x-vector entries*
2: P_k receives the needed x-vector entries that are not in $\mathcal{X}_k$
3: P_k sends the x-vector entries in $\mathcal{X}_k$ that are needed by other processors

4: $\triangleright$ *Computations*
5: $y_i^{(k)} \leftarrow y_i^{(k)} + a_{i,j}x_j$ for each $a_{i,j} \in \mathcal{A}_k$

6: $\triangleright$ *Post-communication phase — Folds on y-vector entries*
7: P_k receives the partial results for y-vector entries that are in $\mathcal{Y}_k$
8: P_k sends the partial results for y-vector entries that are not in $\mathcal{Y}_k$

9: $y_i \leftarrow \sum y_i^{(t)}$ for each partial result $y_i^{(t)}$ for $y_i \in \mathcal{Y}_k$

10: **return** $\mathcal{Y}_k$

by P_k during parallel SpMV is given in Algorithm 5. In this algorithm, the partial result produced by a processor P_t for a vector entry y_i is denoted by $y_i^{(t)}$. As seen in the algorithm, no computation starts until all needed x-vector entries are received. Similarly, no produced partial results are sent to the owners of the respective y-vector entries until all computation finishes. Hence, this is an example for the parallel SpMV with non-overlapping computation and communication.

For an efficient parallelization, the objective is to find K-way partitions $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$ and $\Pi_K(\mathcal{Y})$ so that the communication overhead is minimized while a balance on the computational loads of the processors is maintained. The next subsections describe the fine-grain [22] and medium-grain [52] hypergraph partitioning models, in which this objective is met only partially. That is, those models aim at minimizing the communication overhead by only minimizing the bandwidth cost, i.e., the total communication volume.

4.1.2 Fine-grain hypergraph model

Let $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ denote the fine-grain hypergraph of a given SpMV instance $y = Ax$. In $\mathcal{H}$, the vertex set $\mathcal{V}$ contains three types of vertices:

- the vertices representing the nonzero entries in A,
- the vertices representing the entries in $\mathcal{X}$ and
- the vertices representing the entries in $\mathcal{Y}$.

$\mathcal{V}$ contains a vertex $v^a_{i,j}$ for each nonzero entry $a_{i,j}$ in matrix A, a vertex v^x_j for each entry x_j in vector x and a vertex v^y_i for each entry y_i in vector y. That is,

$$\mathcal{V} = \{v^a_{i,j} : a_{i,j} \in \mathcal{A}\} \cup \{v^x_j : x_j \in \mathcal{X}\} \cup \{v^y_i : y_i \in \mathcal{Y}\}.$$

Note that each vertex represents a data entry. However, for each vertex $v^a_{i,j} \in \mathcal{V}$, $v^a_{i,j}$ additionally represents the computational task of the multiply-and-add operation $y_i \leftarrow y_i + a_{i,j}x_j$.

The net set $\mathcal{N}$ contains two types of nets:

- the nets representing the input dependencies of the computational tasks to x-vector entries and
- the nets representing the output dependencies of the computational tasks to y-vector entries.

$\mathcal{N}$ contains a net n^x_j for each entry x_j in x and a net n^y_i for each entry in y. That is,

$$\mathcal{N} = \{n^x_j : x_j \in \mathcal{X}\} \cup \{n^y_i : y_i \in \mathcal{Y}\}.$$

Since n^x_j represents the input dependency of the tasks on x_j, it connects v^x_j and the vertices that represent the tasks that need x_j. That is,

$$Pins(n^x_j) = \{v^x_j\} \cup \{v^a_{t,j} : a_{t,j} \neq 0\}.$$

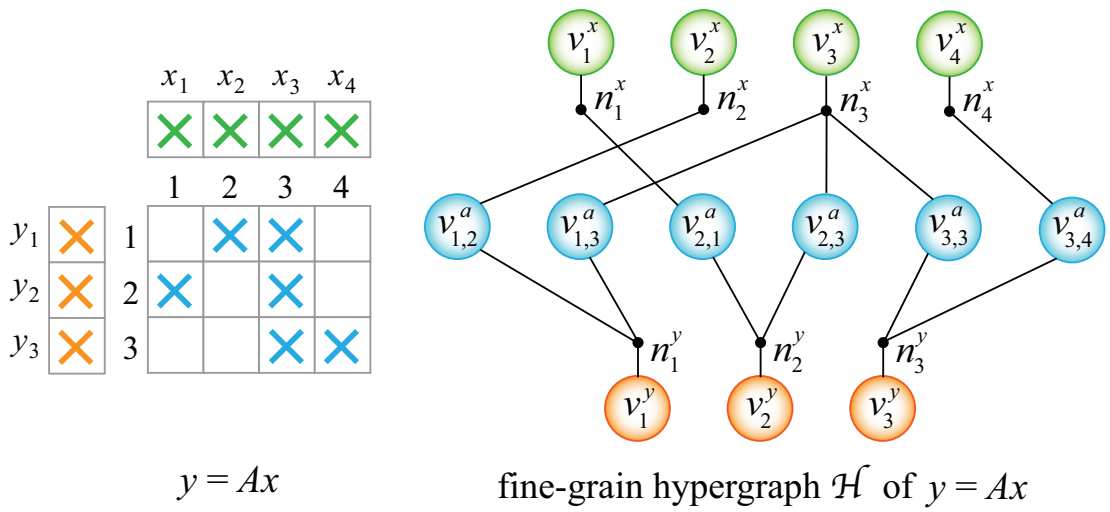


Figure 4.1: A sample $y = Ax$ instance and its corresponding fine-grain hypergraph.

Since n_i^y represents the output dependency of the tasks on y_i, it connects v_i^y and the vertices that represent the tasks that produce partial results for y_i. That is,

$$Pins(n_i^y) = \{v_i^y\} \cup \{v_{i,t}^a : a_{i,t} \neq 0\}.$$

$\mathcal{H}$ contains $n_{nz} + n_c + n_r$ vertices, $n_c + n_r$ nets and $2n_{nz} + n_c + n_r$ pins. A sample SpMV instance and the fine-grain hypergraph corresponding to this SpMV instance are shown in Figure 4.1.

In the fine-grain model, the weight $w(v)$ of vertex v is set to the computation load associated to v. So, a unit weight $w(v_{i,j}^a) = 1$ is assigned to each $v_{i,j}^a \in \mathcal{V}$ and zero weights $w(v_j^x) = w(v_i^y) = 0$ are assigned to each $v_j^x \in \mathcal{V}$ and $v_i^y \in \mathcal{V}$.

For parallel SpMV, a K-way partition $\Pi_K(\mathcal{H})$ of $\mathcal{H}$ is decoded as follows. The nonzeros and the responsibilities of performing their corresponding multiply-and-add operations represented by the vertices in part $\mathcal{V}_k$ are assigned to processor P_k, as well as the vector entries represented by the vertices in $\mathcal{V}_k$. That is,

$$\mathcal{A}_k = \{a_{i,j} : v_{i,j}^a \in \mathcal{V}_k\},$$
$$\mathcal{X}_k = \{x_j : v_j^x \in \mathcal{V}_k\},$$
$$\mathcal{Y}_k = \{y_i : v_i^y \in \mathcal{V}_k\}.$$

Consider a cut net n_j^x in $\Pi_K(\mathcal{H})$ and let $v_j^x \in \mathcal{V}_k$ for some $\mathcal{V}_k \in \lambda(n_j^x)$. The pin of n_j^x to a vertex $v_{i,j}^a \in \mathcal{V}_\ell$ with $\ell \neq k$ signifies the need of processor P_ℓ to vector entry x_j due to the computation task $y_i \leftarrow y_i + a_{i,j}x_j$. Hence in the pre-communication phase, P_k sends x_j to the processors that correspond to the parts in $\lambda(n_j^x) - \mathcal{V}_k$. The total volume of communication in the pre-communication phase is then the sum of $(\lambda(n_j^x) - 1)$ values over the cut nets representing input dependencies on x-vector entries. Similarly, consider a cut net n_t^y in $\Pi_K(\mathcal{H})$ and let $v_t^y \in \mathcal{V}_q$ for some $\mathcal{V}_q \in \lambda(n_t^y)$. The pin of n_t^y to a vertex $v_{t,s}^a \in \mathcal{V}_r$ with $r \neq q$ signifies the partial results produced by processor P_r for vector entry y_t due to the computation task $y_t \leftarrow y_t + a_{t,s}x_s$. Hence in the post-communication phase, P_q receives partial results for y_t from the processors that correspond to the parts in $\lambda(n_t^y) - \mathcal{V}_q$. The total volume of communication in the post-communication phase is the sum of $(\lambda(n_t^y) - 1)$ values over the cut nets representing output dependencies on y-vector entries. As a result, the total volume of communication in an SpMV iteration equates to the total cutsize of $\Pi_K(\mathcal{H})$. Therefore, minimizing the total cutsize of $\Pi_K(\mathcal{H})$ corresponds to minimizing the total communication volume of the parallel SpMV with the corresponding partitions $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$ and $\Pi_K(\mathcal{Y})$.

For each part $\mathcal{V}_k$ in $\Pi_K(\mathcal{H})$, the weight $W(\mathcal{V}_k)$ of $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in $\mathcal{V}_k$. Since only the vertices representing nonzero entries in matrix A are assigned (unit) weights, $W(\mathcal{V}_k)$ equates to the number of the multiply-and-add operations performed by P_k. Then, maintaining balance on the weights of the parts in $\Pi_K(\mathcal{H})$ corresponds to maintaining balance on the computational loads of the processors.

4.1.3 Medium-grain hypergraph model

Given an SpMV instance $y = Ax$, the medium-grain hypergraph model applies the recursive bipartitioning (RB) paradigm on the nonzero entries of matrix A and the entries of vectors x and y. As in graph/hypergraph partitioning described in the previous chapters, this procedure forms a hypothetical RB tree whose nodes represent the parts consisting of matrix and vector entries. Let $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$

denote the parts of sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$ corresponding to the kth node of the ℓth level of the RB tree. Here, ℓ ranges from 0 to $\log K$, where K is the number of parts/processors. Note that the topmost node of the RB tree represents the original sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$. That is, $\mathcal{A}_0^0 = \mathcal{A}$, $\mathcal{X}_0^0 = \mathcal{X}$ and $\mathcal{Y}_0^0 = \mathcal{Y}$.

At each step of bipartitioning a node in the RB process, the medium-grain model forms a new hypergraph using the A-matrix nonzero entries and the vector entries. The hypergraph formed while bipartitioning the parts $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$ is denoted by $\mathcal{H}_k^\ell$, for $0 \leq \ell \leq \log K$ and $0 \leq k \leq 2^\ell - 1$. Before forming $\mathcal{H}_k^\ell$, a two-way splitting of the nonzeros in $\mathcal{A}_k^\ell$ is obtained. In this splitting procedure, if row i contains at least one nonzero entry in $\mathcal{A}_k^\ell$ or $\mathcal{Y}_k^\ell$ contains y-vector entry y_i, then a group row_i represents these entries, for each $1 \leq i \leq n_r$. Note that $\mathcal{A}_k^\ell$ does not have to contain a nonzero entry in row i just because $y_i \in \mathcal{Y}_k^\ell$. If $\mathcal{A}_k^\ell$ does not contain any nonzero entries in row i and $y_i \in \mathcal{Y}_k$, group row_i represents only y_i. In a dual manner, $\mathcal{Y}_k^\ell$ does not have to contain y_i just because A_k^ℓ contains some nonzeros in row i. If $y_i \notin \mathcal{Y}_k^\ell$ and $\mathcal{A}_k^\ell$ contains at least one nonzero entry in row i, group row_i represents those nonzeros in row i in $\mathcal{A}_k^\ell$. Similarly, if column j contains at least one nonzero entry in $\mathcal{A}_k^\ell$ or $\mathcal{X}_k^\ell$ contains x-vector entry x_j, then a group $column_j$ represents these entries, for each $1 \leq j \leq n_c$.

In the medium-grain model, each nonzero entry $a_{i,j} \in \mathcal{A}_k^\ell$ is assigned either to group row_i or to group $column_j$. This assignment is determined by a simple heuristic, which was proven to be effective [52]. In this heuristic, a nonzero entry $a_{i,j} \in \mathcal{A}_k^\ell$ is assigned to row_i if row i has fewer nonzero entries in $\mathcal{A}_k^\ell$ than column j. It is assigned to $column_j$, if column j has fewer nonzero enries in $\mathcal{A}_k^\ell$ than row i. The motivation is that the nonzero entries of a row/column with fewer nonzeros are less likely to be scattered to multiple processors, hence it is better to keep those nonzeros together for the partitioning procedure. If column j and row i have equal number of nonzeros in $\mathcal{A}_k$, then $a_{i,j}$ is assigned to the group for the longer dimension. For a matrix with $n_r = n_c$, a global decision of always assigning such nonzeros to either their row groups or their column groups is followed.

After each nonzero in $\mathcal{A}_k^\ell$ is assigned to a group corresponding to its row or column, the corresponding hypergraph $\mathcal{H}_k^\ell = (\mathcal{V}_k^\ell, \mathcal{N}_k^\ell)$ is formed as follows. For

each group $column_j$, vertex set $\mathcal{V}$ contains a vertex v_j^x, which represents the nonzeros assigned to $column_j$ as well as the vector entry x_j. Similarly for each group row_i, $\mathcal{V}$ contains a vertex v_i^y, which represents the nonzeros assigned to row_i as well as the vector entry y_i. That is,

$$\mathcal{V}_k^\ell = \{v_j^x : x_j \in \mathcal{X}_k^\ell \text{ or } a_{t,j} \in \mathcal{A}_k^\ell\} \cup \{v_i^y : y_i \in \mathcal{Y}_k^\ell \text{ or } a_{i,t} \in \mathcal{A}_k^\ell\}$$

For each group $row_i/column_j$, net set $\mathcal{N}$ contains a net n_i^y/n_j^x. That is,

$$\mathcal{N} = \{n_j^x : x_j \in \mathcal{X}_k^\ell \text{ or } a_{t,j} \in \mathcal{A}_k^\ell\} \cup \{n_i^y : y_i \in \mathcal{Y}_k^\ell \text{ or } a_{i,t} \in \mathcal{A}_k^\ell\}$$

Net n_j^x represents the dependency of the groups to x-vector entry x_j. Note that group $column_j$ always depends on x_j because it either represents x_j or contains A-matrix nonzeros each of whose multiply-and-add operation requires x_j, or both. Also note that $column_j$ is the only column group that can depend on x_j. However, multiple row groups can depend on x_j, since multiple nonzeros in column j happen to be assigned to their column groups. That is, for each nonzero $a_{t,j}$ which is assigned to group row_t, row_t depends on x_j due to the multiply-and-add operation of $a_{t,j}$. Let $map(\cdot)$ function denote the assignment of nonzeros to their row or column groups. Then, the set of vertices connected by n_j^x can be formulated as

$$Pins(n_j^x) = \{v_j^x\} \cup \{v_t^y : map(a_{t,j}) = row_t\}.$$

Similary, net n_i^y represents the dependency of the groups to y-vector entry y_i. Row group row_i as well as each column group $column_r$ where $a_{i,r}$ is assigned to $column_r$ depend on y_i. Hence, the set of vertices connected by n_i^y can be formulated as

$$Pins(n_i^y) = \{v_i^y\} \cup \{v_r^x : map(a_{i,r}) = column_r\}.$$

In the medium-grain model, each net is assigned unit cost and each vertex is assigned a weight equal to the number of nonzeros assigned to the corresponding

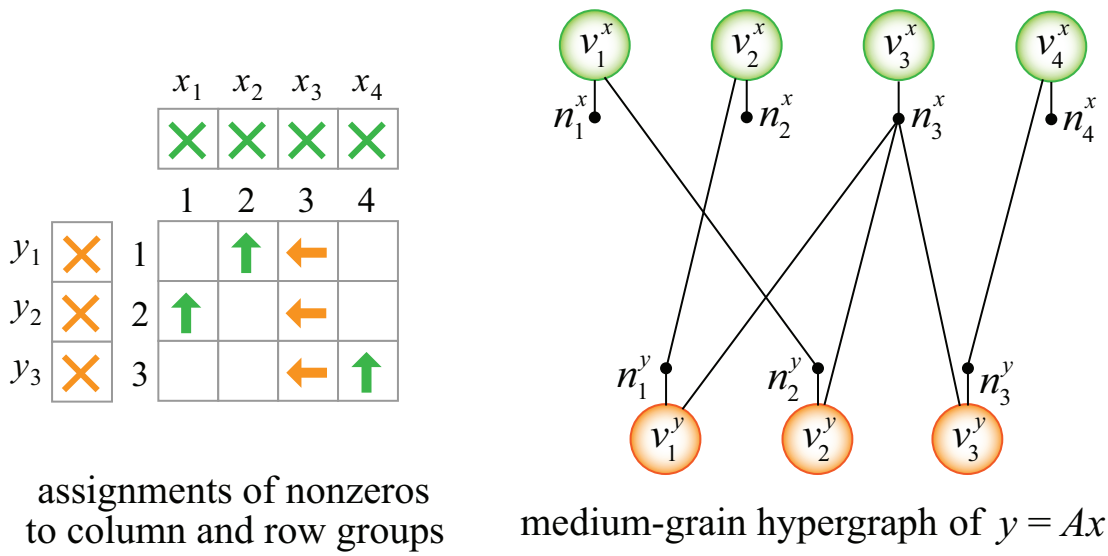


Figure 4.2: The nonzero assignments of the sample $y = Ax$ and the corresponding medium-grain hypergraph.

row/column group. That is,

$$w(v_j^x) = |\{a_{t,j} : map(a_{t,j}) = column_j\}|,$$
$$w(v_i^y) = |\{a_{i,t} : map(a_{i,r}) = row_r\}|.$$

Then, $\mathcal{H}_k^\ell$ is bipartitioned with the objective of minimizing the cutsize and the constraint of maintaining balance on the part weights. As in the fine-grain model, the cutsize-minimization objective in all RB steps corresponds to the objective of minimizing the total volume of communication. The constraint of maintaining balance on the weights of the parts corresponds to the constraint of maintaining balance on the computational loads of the processors.

A sample SpMV instance is given in Figure 4.2 together with the arrows pointing the direction of the assignment which is performed in the medium-grain model according to the described heuristic. Note that this assignment takes places before the first bipartition, hence it covers all of the nonzeros in A, i.e., $\mathcal{A}_0^0$. As an assignment example, nonzero $a_{1,2}$ is assigned to group $column_2$ since row 1 has two nonzeros while column 2 has only one nonzero. The corresponding medium-grain hypergraph $\mathcal{H}_0^0$ is also given in the figure. For example, net n_3^x connects vertex v_3^x as well as vertices v_1^y, v_2^y and v_3^y since all nonzeros in column 3 are

assigned to their corresponding row groups.

4.2 Reducing latency in fine-grain model

In this section, the proposed fine-grain hypergraph partitioning model by which the bandwidth and latency costs of parallel SpMV are *simultaneously* minimized is described. The proposed model utilizes the RB paradigm and is built upon the fine-grain hypergraph, which is described in Section 4.1.2. In each bipartitioning step, our model forms and adds new nets to the hypergraph to be bipartitioned in addition to the existing nets in the original fine-grain model. The newly added nets encode the minimization of the latency cost by minimizing the total number of messages, so, these nets are called *message nets*. The nets that alredy exist in the original fine-grain model are called *volume nets* since they encode the minimization of the bandwidth cost by minimizing the total volume of communication.

Message nets motivate the matrix nonzeros and vector entries that altogether cause communicating a message to stay together throughout the RB process. In other words, the total cutsize defined over the message nets relate to the total number of messages. The message net concept was used in [1] for iterative applications with one communication phase and conformal input and output partitions. In the current work, the message nets are used for improving the performance of parallel SpMV with two communication phases and possibly nonconformal vector partitions.

The existence of the message nets in our model relies on the fine-grain model which is partitioned using the RB paradigm. As in models described in the previous sections and chapters, the RB process forms a hypothetical full binay tree called RB tree. The naming of the nodes in the RB tree is the same as the previous models, that is, the kth hypergraph in the ℓth level of the RB tree is denoted by $\mathcal{H}_k^\ell$. So, the topmost hypergraph in the RB tree, i.e., $\mathcal{H}_0^0$, is the original fine-grain hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. Again as in the previous models,

the hypergraphs are assumed to be bipartitioned in the breadth-first order of the nodes in the RB tree. At any step of the RB process, the vertex sets of the leaf hypergraphs of the current RB tree induce a k-way partition, where k denotes the number of leaves. Recall that a k-way partition induced by the leaves of the RB tree then induces a k-way partition on the processors. The kth node in the ℓth level also represents the processor group denoted by $\mathcal{P}_k^\ell$. The computation tasks (multiply-and-add operations) and data elements (A-matrix nonzeros and x- and y-vector entries) that are represented by the vertices in $\mathcal{H}_k^\ell$, or equivalently $\mathcal{V}_k^\ell$ are assigned to the processors in the group $\mathcal{P}_k^\ell$.

Throughout the RB process, the nets of the original fine-grain hypergraph $\mathcal{H}_0^0$, i.e., the volume nets, are maintained by the cut-net splitting technique [19]. Consider the bipartition $\Pi(\mathcal{H}_k^\ell) = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{V}_{2k+1}^{\ell+1}\}$ of hypergraph $\mathcal{H}_k^\ell$. In hypergraphs $\mathcal{H}_{2k}^{\ell+1}$ and $\mathcal{H}_{2k+1}^{\ell+1}$, the volume nets in $\mathcal{H}_k^\ell$ are split as follows. If a net n_i is internal to $\mathcal{V}_{2k}^{\ell+1}$, i.e., $Pins(n_i) \subseteq \mathcal{V}_{2k}^{\ell+1}$, then n_i is included in $\mathcal{H}_{2k}^{\ell+1}$ as is, i.e., without changing any of its connections. Similarly if n_i is internal to $\mathcal{V}_{2k+1}^{\ell+1}$, then it is included in $\mathcal{H}_{2k+1}^{\ell+1}$ as is. If n_i is a cut net, then two *split* versions of n_i, namely n_i' and n_i'', are respectively included in $\mathcal{H}_{2k}^{\ell+1}$ and $\mathcal{H}_{2k+1}^{\ell+1}$. Split net n_i' only connects the subset of vertices connected by n_i that are assigned to $\mathcal{H}_{2k}^{\ell+1}$. That is,

$$Pins(n_i') = \{v : v \in Pins(n_i) \cap \mathcal{V}_{2k}^{\ell+1}\}.$$

Similarly, split net n_i'' only connects the subset of vertices connected by n_i that are assigned to $\mathcal{H}_{2k+1}^{\ell+1}$. That is,

$$Pins(n_i'') = \{v : v \in Pins(n_i) \cap \mathcal{V}_{2k+1}^{\ell+1}\}.$$

In this technique, the sum of the number of cut volume nets over the RB steps amounts to the total cutsize of the volume nets in the final K-way partition which results from the RB process.

Note that the maintainance of the volume nets throughout the RB process only requires the local bipartition information for each RB step. For this reason, the volume nets of a hypergraph can be formed as soon as the vertex set is

Algorithm 6: MAIN

Require: $\mathcal{A}$, $\mathcal{X}$, $\mathcal{Y}$ and K

1: $\mathcal{A}_0^0 \leftarrow \mathcal{A}$, $\mathcal{X}_0^0 \leftarrow \mathcal{X}$, $\mathcal{Y}_0^0 \leftarrow \mathcal{Y}$
2: $\mathcal{H}_0^0 \leftarrow \mathcal{H} \leftarrow$FINE-GRAIN-HYPERGRAPH$(\mathcal{A}, \mathcal{X}, \mathcal{Y})$
3: $K' \leftarrow 1$
4: $\Pi_{K'}(\mathcal{A}) \leftarrow \{\mathcal{A}_0^0\}$, $\Pi_{K'}(\mathcal{X}) \leftarrow \{\mathcal{X}_0^0\}$, $\Pi_{K'}(\mathcal{Y}) \leftarrow \{\mathcal{Y}_0^0\}$
5: **for** $\ell \leftarrow 0$ **to** $\log_2 K - 1$ **do**
6: **for** $k \leftarrow 0$ **to** $2^\ell - 1$ **do**
7: **if** $\ell > 0$ **then**
8: ADD-MESSAGE-NETS$(\mathcal{H}_k^\ell, \Pi_{K'}(\mathcal{A}), \Pi_{K'}(\mathcal{X}), \Pi_{K'}(\mathcal{Y}))$
9: $\Pi \leftarrow$ BIPARTITION$(\mathcal{H}_k^\ell)$ $\triangleright \Pi = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{V}_{2k+1}^{\ell+1}\}$
 $\triangleright$ *Derive bipartitions for* $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ *and* $\mathcal{Y}_k^\ell$ *using* Π
10: $\mathcal{A}_{2k}^{\ell+1} \leftarrow \{a_{i,j} : v_{i,j}^a \in \mathcal{V}_{2k}^{\ell+1}\}$ and $\mathcal{A}_{2k+1}^{\ell+1} \leftarrow \{a_{i,j} : v_{i,j}^a \in \mathcal{V}_{2k+1}^{\ell+1}\}$
11: $\mathcal{X}_{2k}^{\ell+1} \leftarrow \{x_j : v_j^x \in \mathcal{V}_{2k}^{\ell+1}\}$ and $\mathcal{X}_{2k+1}^{\ell+1} \leftarrow \{x_j : v_j^x \in \mathcal{V}_{2k+1}^{\ell+1}\}$
12: $\mathcal{Y}_{2k}^{\ell+1} \leftarrow \{y_i : v_i^y \in \mathcal{V}_{2k}^{\ell+1}\}$ and $\mathcal{Y}_{2k+1}^{\ell+1} \leftarrow \{y_i : v_i^y \in \mathcal{V}_{2k+1}^{\ell+1}\}$
 $\triangleright$ *Update partitions*
13: $\Pi_{K'+1}(\mathcal{A}) \leftarrow \Pi_{K'}(\mathcal{A}) - \{\mathcal{A}_k^\ell\} \cup \{\mathcal{A}_{2k}^{\ell+1}, \mathcal{A}_{2k+1}^{\ell+1}\}$
14: $\Pi_{K'+1}(\mathcal{X}) \leftarrow \Pi_{K'}(\mathcal{X}) - \{\mathcal{X}_k^\ell\} \cup \{\mathcal{X}_{2k}^{\ell+1}, \mathcal{X}_{2k+1}^{\ell+1}\}$
15: $\Pi_{K'+1}(\mathcal{Y}) \leftarrow \Pi_{K'}(\mathcal{Y}) - \{\mathcal{Y}_k^\ell\} \cup \{\mathcal{Y}_{2k}^{\ell+1}, \mathcal{Y}_{2k+1}^{\ell+1}\}$
16: Split cut-volume nets of $\mathcal{N}_k^\ell$ to obtain $\mathcal{H}_{2k}^{\ell+1}$ and $\mathcal{H}_{2k+1}^{\ell+1}$
17: $K' \leftarrow K' + 1$
18: **return** $\Pi_K(\mathcal{A})$, $\Pi_K(\mathcal{X})$ and $\Pi_K(\mathcal{Y})$

formed. For example, volume nets of $\mathcal{H}_{2k}^{\ell+1}$ and $\mathcal{H}_{2k+1}^{\ell+1}$ can be formed just after the bipartition $\Pi(\mathcal{H}_k^\ell) = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{V}_{2k+1}^{\ell+1}\}$ of hypergraph $\mathcal{H}_k^\ell$ is obtained. However, the construction of the message nets needs a more global view as they involve information about all of the processor groups, which are represented by the whole set of leaf nodes in the RB tree. Hence, the message nets are added to $\mathcal{H}_k^\ell$ just before it is bipartitioned for the most up-to-date state of the processor groups to be covered.

Algorithm 6 displays the basic steps of the proposed partitioning algorithm. As inputs, it takes the sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$ and the number of processors/parts K. As seen in line 1, these sets are represented by the topmost node of the RB tree. In line 2, the fine-grain hypergraph $\mathcal{H} = \mathcal{H}_0^0$ is formed as described in Section 4.1.2. The running number of parts (i.e., the number of leaves in the RB tree) is denoted by K' and its initial value is set to 1 in line 3. The partitions of sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$ are all initialized as their singleton partition, as seen in line 4. The RB process takes place in lines 5-17. As seen in lines 7-9, the message nets are added to hypergraph $\mathcal{H}_k^\ell$, which already contains the volume nets, just before it is bipartitioned. The bipartition Π of $\mathcal{H}_k^\ell$ is then utilized to form the bipartitions of $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$, as seen in lines 10-12. Then in lines 13-15, the global partitions of $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$ are updated by adding the new parts and removing the old parts that have just been bipartitioned. In line 16, the cut-net splitting technique is applied to the volume nets of $\mathcal{H}_k^\ell$ and two new hypergraphs $\mathcal{H}_{2k}^{\ell+1} = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{N}_{2k}^{\ell+1}\}$ and $\mathcal{H}_{2k+1}^{\ell+1} = \{\mathcal{V}_{2k+1}^{\ell+1}, \mathcal{N}_{2k+1}^{\ell+1}\}$ are obtained. As seen in line 18, the K-way partitions

$$\Pi_K(\mathcal{A}) = \{\mathcal{A}_0^{\log_2 K}, \mathcal{A}_1^{\log_2 K}, \ldots, \mathcal{A}_{K-1}^{\log_2 K}\}$$

$$\Pi_K(\mathcal{X}) = \{\mathcal{X}_0^{\log_2 K}, \mathcal{X}_1^{\log_2 K}, \ldots, \mathcal{X}_{K-1}^{\log_2 K}\}$$

$$\Pi_K(\mathcal{Y}) = \{\mathcal{Y}_0^{\log_2 K}, \mathcal{Y}_1^{\log_2 K}, \ldots, \mathcal{Y}_{K-1}^{\log_2 K}\}$$

are returned.

Algorithm 7 displays the message-net addition algorithm. As input, it takes hypergraph $\mathcal{H}_k^\ell$ to which message nets are going to be added. It also takes the K'-way partitions of the sets $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$. Recall that K' denotes the number of

Algorithm 7: ADD-MESSAGE-NETS

Require: $\mathcal{H}_k^\ell = (\mathcal{V}_k^\ell, \mathcal{N}_k^\ell)$, $\Pi_{K'}(\mathcal{A})$, $\Pi_{K'}(\mathcal{X})$, $\Pi_{K'}(\mathcal{Y})$.

$\triangleright$ *Form expand-send nets*
1: **for** each $x_j \in \mathcal{X}_k^\ell$ **do**
2: **for** each $a_{t,j} \in \mathcal{A}_q^r$ where $\mathcal{A}_q^r \neq \mathcal{A}_k^\ell$ **do**
3: **if** $es_q^r \notin \mathcal{N}_k^\ell$ **then**
4: $Pins(es_q^r) \leftarrow \{v_j^x\}$ and $\mathcal{N}_k^\ell \leftarrow \mathcal{N}_k^\ell \cup \{es_q^r\}$
5: **else**
6: $Pins(es_q^r) \leftarrow Pins(es_q^r) \cup \{v_j^x\}$

$\triangleright$ *Form expand-receive nets*
7: **for** each $a_{t,j} \in \mathcal{A}_k^\ell$ **do**
8: **for** each $x_j \in \mathcal{X}_q^r$ where $\mathcal{X}_q^r \neq \mathcal{X}_k^\ell$ **do**
9: **if** $er_q^r \notin \mathcal{N}_k^\ell$ **then**
10: $Pins(er_q^r) \leftarrow \{v_{t,j}^a\}$ and $\mathcal{N}_k^\ell \leftarrow \mathcal{N}_k^\ell \cup \{er_q^r\}$
11: **else**
12: $Pins(er_q^r) \leftarrow Pins(er_q^r) \cup \{v_{t,j}^a\}$

$\triangleright$ *Form fold-send nets*
13: **for** each $a_{i,t} \in \mathcal{A}_k^\ell$ **do**
14: **for** each $y_i \in \mathcal{Y}_q^r$ where $\mathcal{Y}_q^r \neq \mathcal{Y}_k^\ell$ **do**
15: **if** $fs_q^r \notin \mathcal{N}_k^\ell$ **then**
16: $Pins(fs_q^r) \leftarrow \{v_{i,t}^a\}$ and $\mathcal{N}_k^\ell \leftarrow \mathcal{N}_k^\ell \cup \{fs_q^r\}$
17: **else**
18: $Pins(fs_q^r) \leftarrow Pins(fs_q^r) \cup \{v_{i,t}^a\}$

$\triangleright$ *Form fold-receive nets*
19: **for** each $y_i \in \mathcal{Y}_k^\ell$ **do**
20: **for** each $a_{i,t} \in \mathcal{A}_q^r$ where $\mathcal{A}_q^r \neq \mathcal{A}_k^\ell$ **do**
21: **if** $fr_q^r \notin \mathcal{N}_k^\ell$ **then**
22: $Pins(fr_q^r) \leftarrow \{v_i^y\}$ and $\mathcal{N}_k^\ell \leftarrow \mathcal{N}_k^\ell \cup \{fr_q^r\}$
23: **else**
24: $Pins(fr_q^r) \leftarrow Pins(fr_q^r) \cup \{v_i^y\}$

leaves of the current RB tree. Hence, the input partitions are up-to-date. Also recall that the elements in $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$ and the corresponding multiply-and-add operations are assigned to processor group $\mathcal{P}_k^\ell$, and there exists a processor group for each leaf node of the current RB tree.

There are four types of message nets: expand-send nets, expand-receive nets, fold-send nets and fold-receive nets. To differentiate the message nets from the volume nets, we denote them with their initials. We use es, er, fs and fr to denote expand-send, expand-receive, fold-send and fold-receive nets. For each processor group $\mathcal{P}_q^r$ different than $\mathcal{P}_k^\ell$, these four different message nets can be added to $\mathcal{H}_k^\ell$ depending on the distribution of the matrix and vector elements. es_q^r, er_q^r, fs_q^r and fr_q^r respectively denote the expand-send, expand-receive, fold-send and fold-receive nets which are added for processor group $\mathcal{P}_q^r$.

Expand-send net es_q^r signifies the message sent by $\mathcal{P}_k^\ell$ to $\mathcal{P}_q^r$ regarding the expand operations in the pre-communication phase. es_q^r is added to $\mathcal{H}_k^\ell$ only if such a message exists. It connects the vertices that represent the x-vector entries to be expanded to $\mathcal{P}_q^r$. Since each $x_j \in \mathcal{X}_k^\ell$ for each $a_{t,j} \in \mathcal{A}_q^r$ should be sent to $\mathcal{P}_q^r$ by $\mathcal{P}_k^\ell$, the vertices connected by es_q^r can be formulated as

$$Pins(es_q^r) = \{v_j^x : x_j \in \mathcal{X}_k^\ell \text{ and } a_{t,j} \in \mathcal{A}_q^r\}.$$

The addition of the expand-send nets to $\mathcal{H}_k^\ell$ is given in lines 1–6 of Algorithm 7. If es_q^r becomes cut in bipartiton Π, then both $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ send a message to $\mathcal{P}_q^r$ in the pre-communication phase. The sets of x-vector entries sent from $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ to $\mathcal{P}_q^r$ are $\{x_j : v_j^x \in \mathcal{V}_{2k}^{\ell+1} \text{ and } a_{t,j} \in \mathcal{A}_q^r\}$ and $\{x_j : v_j^x \in \mathcal{V}_{2k+1}^{\ell+1} \text{ and } a_{t,j} \in \mathcal{A}_q^r\}$, respectively.

Expand-receive net er_q^r signifies the message received by $\mathcal{P}_k^\ell$ from $\mathcal{P}_q^r$ regarding the expand operations in the pre-communication phase. er_q^r is added to $\mathcal{H}_k^\ell$ only if such a message exists. It connects the vertices that represent the A-matrix nonzeros that need x-vector entries to be received from $\mathcal{P}_q^r$. Since each $x_i \in \mathcal{X}_q^r$ for each $a_{i,t} \in \mathcal{A}_k^\ell$ should be received from $\mathcal{P}_q^r$ by $\mathcal{P}_k^\ell$, the vertices connected by

er_q^r can be formulated as

$$Pins(er_q^r) = \{v_{t,j}^a : a_{t,j} \in \mathcal{A}_k^\ell \text{ and } x_j \in \mathcal{X}_q^r\}.$$

The addition of the expand-receive nets to $\mathcal{H}_k^\ell$ is given in lines 7–12 of Algorithm 7. If er_q^r becomes cut in bipartiton Π, then both $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ receive a message from $\mathcal{P}_q^r$ in the pre-communication phase. The sets of x-vector entries received by $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ from $\mathcal{P}_q^r$ are $\{x_j : v_j^x \in \mathcal{V}_q^r \text{ and } a_{t,j} \in \mathcal{A}_{2k}^{\ell+1}\}$ and $\{x_j : v_j^x \in \mathcal{V}_q^r \text{ and } a_{t,j} \in \mathcal{A}_{2k+1}^{\ell+1}\}$, respectively.

Fold-send net fs_q^r signifies the message sent by $\mathcal{P}_k^\ell$ to $\mathcal{P}_q^r$ regarding the fold operations in the post-communication phase. fs_q^r is added to $\mathcal{H}_k^\ell$ only if such a message exists. It connects the vertices that represent the A-matrix nonzeros that produce partial results for the y-vector entries in $\mathcal{P}_q^r$. Since each partial result of $a_{i,t} \in \mathcal{A}_k^\ell$ for each $y_i \in \mathcal{Y}_q^r$ should be sent to $\mathcal{P}_q^r$ by $\mathcal{P}_k^\ell$, the vertices connected by fs_q^r can be formulated as

$$Pins(fs_q^r) = \{v_{i,t}^a : a_{i,t} \in \mathcal{A}_k^\ell \text{ and } y_i \in \mathcal{Y}_q^r\}.$$

The addition of the fold-send nets to $\mathcal{H}_k^\ell$ is given in lines 13–18 of Algorithm 7. If fs_q^r becomes cut in bipartiton Π, then both $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ send a message to $\mathcal{P}_q^r$ in the post-communication phase. The sets of partial results sent by $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ to $\mathcal{P}_q^r$ are $\{a_{i,t}x_t : v_i^y \in \mathcal{V}_q^r \text{ and } a_{i,t} \in \mathcal{A}_{2k}^{\ell+1}\}$ and $\{a_{i,t}x_y : v_i^y \in \mathcal{V}_q^r \text{ and } a_{i,t} \in \mathcal{A}_{2k+1}^{\ell+1}\}$, respectively.

Fold-receive net fr_q^r signifies the message received by $\mathcal{P}_k^\ell$ from $\mathcal{P}_q^r$ regarding the fold operations in the post-communication phase. fr_q^r is added to $\mathcal{H}_k^\ell$ only if such a message exists. It connects the vertices that represent the y-vector entries that need partial results from $\mathcal{P}_q^r$. Since a partial result for each $y_i \in \mathcal{Y}_k^\ell$ for each $a_{i,t} \in \mathcal{A}_q^r$ should be received from $\mathcal{P}_q^r$ by $\mathcal{P}_k^\ell$, the vertices connected by fr_q^r can be formulated as

$$Pins(fr_q^r) = \{v_i^y : y_i \in \mathcal{Y}_k^\ell \text{ and } a_{i,t} \in \mathcal{A}_q^r\}.$$

The addition of the fold-receive nets to $\mathcal{H}_k^\ell$ is given in lines 19–24 of Algorithm 7. If fr_q^r becomes cut in bipartiton Π, then both $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ receive a message

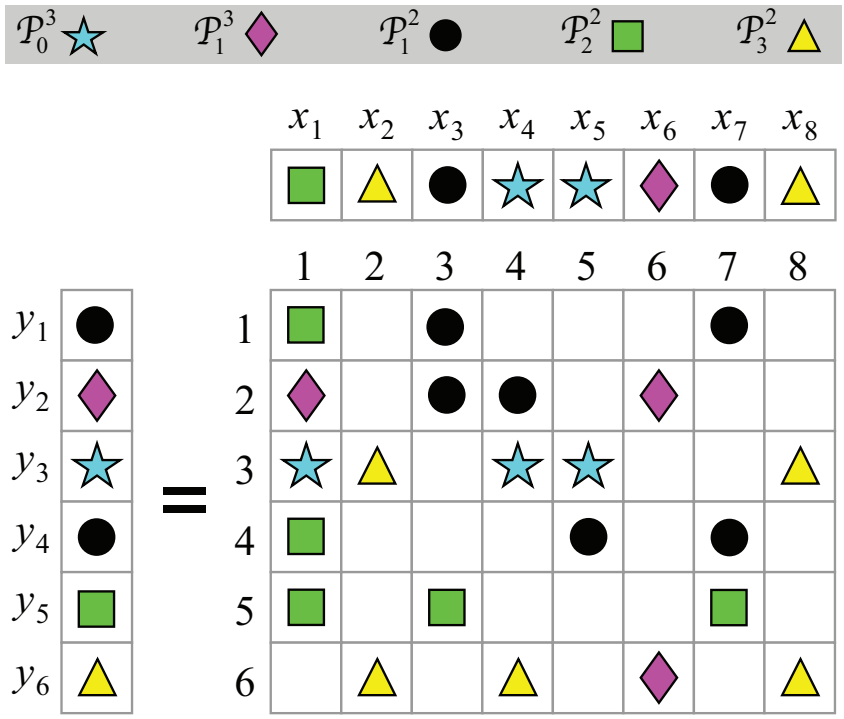


Figure 4.3: 5-way partitions of $\mathcal{A}$, $\mathcal{X}$ and $\mathcal{Y}$.

from $\mathcal{P}_q^r$ in the post-communication phase. The sets of partial results received by $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$ from $\mathcal{P}_q^r$ are $\{a_{i,t}x_t : v_i^y \in \mathcal{V}_{2k}^{\ell+1} \text{ and } a_{i,t} \in \mathcal{A}_q^r\}$ and $\{a_{i,t}x_t : v_i^y \in \mathcal{V}_{2k+1}^{\ell+1} \text{ and } a_{i,t} \in \mathcal{A}_q^r\}$, respectively.

Note that at most four message nets can be added to $\mathcal{H}_k^\ell$ to encapsulate the messages between $\mathcal{P}_k^\ell$ and $\mathcal{P}_q^r$. Since there are at most $K' - 1$ processor groups that $\mathcal{P}_k^\ell$ may communicate with, the maximum number of message nets in $\mathcal{H}_k^\ell$ amounts to $4(K' - 1)$.

Figure 4.3 displays an SpMV instance with a 6×8 matrix A. Consider the RB process of partitioning the fine-grain hypergraph $\mathcal{H}$ corresponding to this SpMV instance. Assume that the RB process is at the state before bipartitioning $\mathcal{H}_1^2$, so, there are five leaf hypergraphs: $\mathcal{H}_0^3$, $\mathcal{H}_1^3$, $\mathcal{H}_1^2$, $\mathcal{H}_2^2$ and $\mathcal{H}_3^2$. The figure displays the assignments of the matrix nonzeros and vector entries to processor groups $\mathcal{P}_0^3$, $\mathcal{P}_1^3$, $\mathcal{P}_1^2$, $\mathcal{P}_2^2$ and $\mathcal{P}_3^2$ induced by the leaf hypergraphs. Each symbol

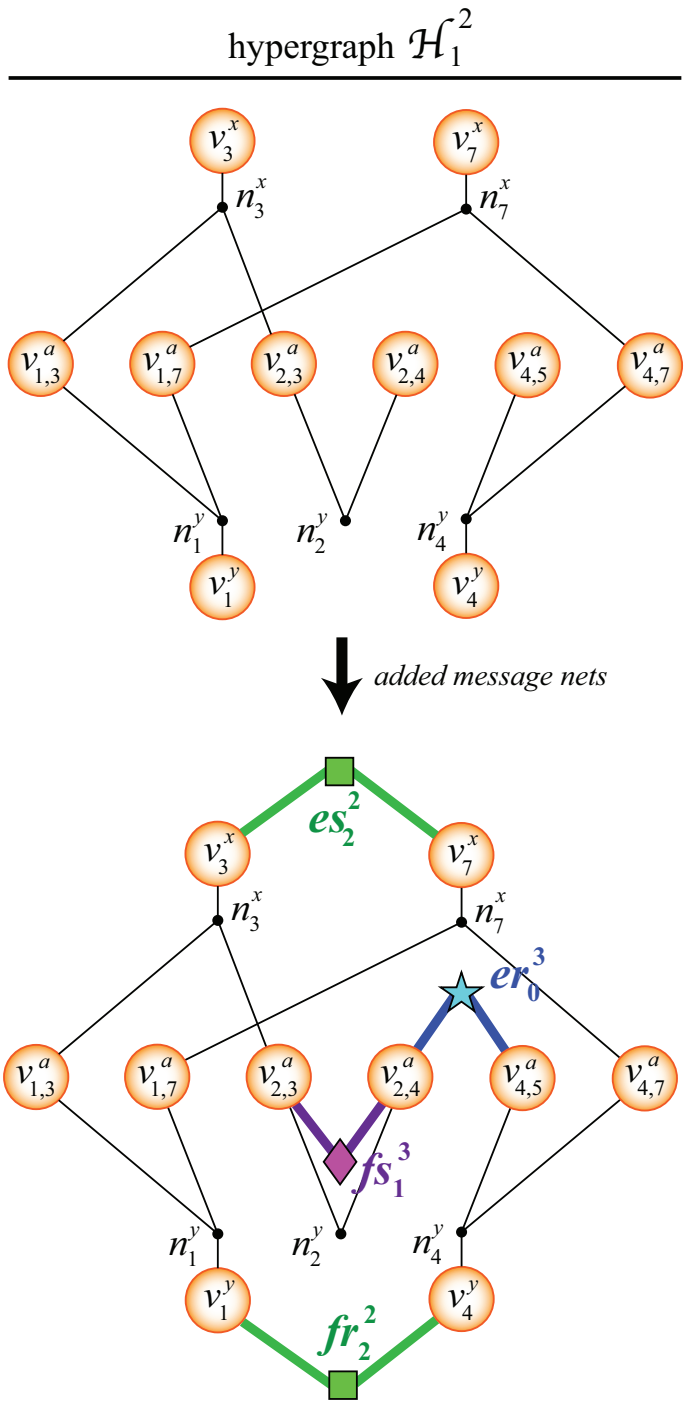


Figure 4.4: Hypergraph $\mathcal{H}_1^2$ with only volume nets (top) and with both types of nets (bottom).

in the figure represents a distinct processor group and a symbol inside a cell signifies the assignment of the corresponding matrix nonzero or vector entry to the processor group represented by that symbol. For example, nonzeros $a_{1,3}$, $a_{1,7}$, $a_{2,3}$, $a_{2,4}$, $a_{4,5}$, $a_{4,7}$, x-vector entries x_3, x_7, and y-vector entries y_1, y_4 are assigned to $\mathcal{P}_1^2$. Figure 4.4 displays hypergraph $\mathcal{H}_1^2$ before and after message-net addition.

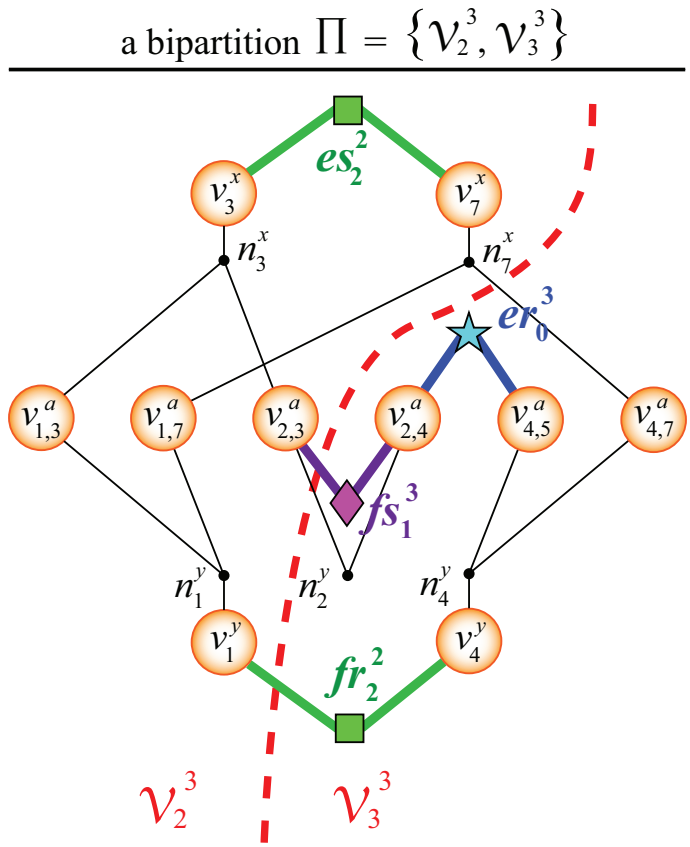


Figure 4.5: A bipartition Π of hypergraph $\mathcal{H}_1^2$.

Figure 4.5 displays a sample bipartition Π for $\mathcal{H}_1^2$ with both net types. As seen in the figure, message nets es_2^2 and er_0^3 are both internal nets, whereas message nets fs_1^3 and fr_2^2 are both cut nets. Net es_2^2 being internal to part $\mathcal{V}_2^3$ represents that among $\mathcal{P}_2^3$ and $\mathcal{P}_3^3$, only $\mathcal{P}_2^3$ sends x-vector entries to $\mathcal{P}_2^2$ regarding the expand operations in the pre-communication phase. Net er_0^3 being internal to part $\mathcal{V}_3^3$ represents that among $\mathcal{P}_2^3$ and $\mathcal{P}_3^3$, only $\mathcal{P}_3^3$ receives x-vector entries from $\mathcal{P}_0^3$ regarding the expand operations in the pre-communication phase. Net fs_1^3 being

Table 4.1: The messages communicated by processor group $\mathcal{P}_1^2$ (equivalently, $\mathcal{P}_2^3$ and $\mathcal{P}_3^3$) in pre- and post-communication phases before and after bipartitioning $\mathcal{H}_1^2$.

RB state	phase	message	due to
before Π	pre	$\mathcal{P}_1^2$ sends $\{x_3, x_7\}$ to $\mathcal{P}_2^2$	$a_{5,3}, a_{5,7}$
		$\mathcal{P}_1^2$ receives $\{x_4, x_5\}$ from $\mathcal{P}_0^3$	$a_{2,4}, a_{4,5}$
	post	$\mathcal{P}_1^2$ sends $\{a_{2,3}x_3 + a_{2,4}x_4\}$ to $\mathcal{P}_1^3$	$a_{2,3}, a_{2,4}$
		$\mathcal{P}_1^2$ receives $\{a_{1,1}x_1, a_{4,1}x_1\}$ from $\mathcal{P}_2^2$	$a_{1,1}, a_{4,1}$
after Π	pre	$\mathcal{P}_2^3$ sends $\{x_3, x_7\}$ to $\mathcal{P}_2^2$	$a_{5,3}, a_{5,7}$
		$\mathcal{P}_3^3$ receives $\{x_4, x_5\}$ from $\mathcal{P}_0^3$	$a_{2,4}, a_{4,5}$
	post	$\mathcal{P}_2^3$ sends $\{a_{2,3}x_3\}$ to $\mathcal{P}_1^3$	$a_{2,3}$
		$\mathcal{P}_3^3$ sends $\{a_{2,4}x_4\}$ to $\mathcal{P}_1^3$	$a_{2,4}$
		$\mathcal{P}_2^3$ receives $\{a_{1,1}x_1\}$ from $\mathcal{P}_2^2$	$a_{1,1}$
		$\mathcal{P}_3^3$ receives $\{a_{4,1}x_1\}$ from $\mathcal{P}_2^2$	$a_{4,1}$

cut represents that both $\mathcal{P}_2^3$ and $\mathcal{P}_3^3$ send partial results to $\mathcal{P}_1^3$ regarding the fold operations in the post-communication phase. Net fr_2^2 being cut represents that $\mathcal{P}_1^3$ receive partial results from both $\mathcal{P}_2^3$ and $\mathcal{P}_3^3$ regarding the fold operations in the post-communication phase. Table 4.1 displays the messages communicated by $\mathcal{P}_1^2$ with the other processor groups. The top half of the table displays the message traffic before $\mathcal{H}_1^3$ (equivalently, $\mathcal{P}_1^2$) is bipartitioned, whereas the bottom half displays the traffic after $\mathcal{H}_1^2$ is bipartitioned as given in Figure 4.5.

In each hypergraph bipartitioning throughout the proposed RB process, the cost of each volume net is set to be the per-word transfer time, t_w, whereas the cost of each message net is set to be the message start-up time, t_s. For a given bipartition Π, let m and v respectively denote the number of cut-message nets and the number of cut-volume nets. Then, the cutsize of Π can be formulated as

$$mt_s + vt_w.$$

As exemplified with Figure 4.5 and Table 4.1, m is equal to the increase in the number of messages that $\mathcal{P}_k^\ell$ communicates with the other processor groups. Since the sum of the cutsizes of the RB-steps when partitioning the original fine-grain

hypergraph is equal to the total communication volume, v is equal to the increase in the total volume of communication. The increase in the overall total number of messages after the bipartition is equal to $m+\delta$, where δ denotes the number of messages between $\mathcal{P}_{2k}^{\ell+1}$ and $\mathcal{P}_{2k+1}^{\ell+1}$. δ is bounded by 2 and emprically found to be equal to 2 in almost all of the bipartitions. Let M and V respectively denote the total number of messages and the total communication volume of the resulting K-way partition. Assume that the communication cost of the resulting partition is formulated as

$$Mt_s + Vt_w.$$

Then, minimizing the cutsize of the bipartitions in the RB process corresponds to minimizing the communication cost of the resulting K-way partition.

The analysis for deriving the complexity of adding message nets are given as follows. While adding expand-send nets to $\mathcal{H}_k^\ell$, all external nonzeros $a_{t,j} \in \mathcal{A}_q^r$ with $x_j \in \mathcal{X}_k^\ell$ are visited once (lines 1–6). Since the parts for x-vector entries represented by the leaf nodes of the RB tree are mutually disjoint and exhaustive, each nonzero of A is visited once during the bipartitionings of a single RB-tree level. While adding expand-receive nets, all nonzeros in $\mathcal{A}_k^\ell$ are visited once (lines 7–12). Again since the parts of the nonzeros of A represented by the leaf nodes of the RB tree are mutually disjoint and exhaustive, each nonzero of A is visited once. Therefore, the complexity of adding expand-send and expand-receive nets in a single level of the RB tree is equal to $O(n_{nz})$. For the addition of the fold-send and fold-receive nets, a dual discussion holds. Then, the overall complexity of adding message nets is $O(n_{nz} \log K)$.

4.2.1 Adaptation for conformality constraint

The conformality of input and output vector partitions is defined as assigning the x_i and y_i to the same processor for each i value. Note that the coefficient matrix should be a square matrix in order the conformality constraint to be imposed. The motivation for the conformality constraint arises in the iterative solvers where y_i of an iteration becomes the x_i of the next iteration. Assigning x_i and y_i to the

same processor prevents communicating y_i to the owner processor of x_i as it will be the new x_i of the next iteration.

In the proposed model, there is no conformality constraint imposed. However, it is possible to adapt the proposed model to obey the conformality constraint with a simple technique used in [33]. In this technique, vertices v_i^x and v_i^y are amalgamated into a new vertex v_i. That is, a single amalgamated vertex v_i represents both x_i and y_i. The weight of v_i is set to be zero as the weights of v_i^x and v_i^y are both zero. In hypergraph $\mathcal{H}_k^\ell$, each (volume or message) net that connects v_i^x or v_i^y then connects the amalgamated vertex v_i. At each bipartitioning, x_i and y_i are both assigned to the processor/processor-group corresponding to the leaf hypergraph which contains v_i.

4.3 Reducing latency in medium-grain model

In this section, we describe a hypergraph partitioning model which is built on the medium-grain model. As in Section 4.2, the proposed model *simultaneously* reduces the bandwidth and latency costs by utilizing volume nets and message nets. The idea is similar to that in Section 4.2, that is, the nets (i.e., the volume nets) of the original medium-grain model capture the bandwidth cost by encoding the volume of communication, whereas the message nets are added to the hypergraphs in the RB process to capture the latency cost by encoding the messages. The context for the message nets, i.e., their types and meanings, is the same as that in Section 4.2. However, the procedure for adding message nets to a medium-grain hypergraph is different than that in Section 4.2.

Consider the medium-grain model described in Section 4.1.3 for obtaining a K-way partition of a given SpMV $y = Ax$. Assume that the RB process is at the state before bipartitioning the kth node of the ℓth level of the RB tree. That is, by using parts $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$, the medium grain hypergraph $\mathcal{H}_k^\ell$ is going to be formed and bipartitioned to obtain $(\mathcal{A}_{2k}^{\ell+1}, \mathcal{A}_{2k}^{\ell+1})$, $(\mathcal{X}_{2k}^{\ell+1}, \mathcal{X}_{2k}^{\ell+1})$ and $(\mathcal{Y}_{2k}^{\ell+1}, \mathcal{Y}_{2k}^{\ell+1})$. As in Section 4.2, the items in $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$ and $\mathcal{Y}_k^\ell$ and their associated tasks are

assigned to processor group $\mathcal{P}_k^\ell$. In each RB step of the proposed model, the following steps are applied:

1. Form the medium-grain hypergraph $\mathcal{H}_k^\ell$ corresponding to $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$, $\mathcal{Y}_k^\ell$.
2. Add message nets to $\mathcal{H}_k^\ell$.
3. Obtain a bipartition $\Pi = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{V}_{2k+1}^{\ell+1}\}$ of $\mathcal{H}_k^\ell$.
4. Derive bipartitions $(\mathcal{A}_{2k}^{\ell+1}, \mathcal{A}_{2k+1}^{\ell+1})$, $(\mathcal{X}_{2k}^{\ell+1}, \mathcal{X}_{2k+1}^{\ell+1})$ and $(\mathcal{Y}_{2k}^{\ell+1}, \mathcal{Y}_{2k+1}^{\ell+1})$ from Π.

Each of these steps are described in detail as follows.

1) Form the medium-grain hypergraph $\mathcal{H}_k^\ell$ using $\mathcal{A}_k^\ell$, $\mathcal{X}_k^\ell$, $\mathcal{Y}_k^\ell$. This process is exactly the same as that takes place for each RB step in the medium-grain model as described in Section 4.1.3. Recall that the nets in the orginal medium-grain hypergraph capture the total volume of communication. Hence, each of these nets are assigned a cost of t_w as in Section 4.2.

2) Add message nets to $\mathcal{H}_k^\ell$. For each processor group $\mathcal{P}_q^r$ different than $\mathcal{P}_k^\ell$, there are four possible message nets that can be added to $\mathcal{H}_k^\ell$ as in Section 4.2. For each of these four net types, the existence condition is the same as that for the corresponding net type in Section 4.2. These four message nets are described as follows:

- **expand-send net** es_q^r: The set of the vertices that are connected by es_q^r is exactly the same as that of the expand-send net described in Section 4.2.
- **expand-receive net** er_q^r: The set of vertices connected by er_q^r is given by

$$\begin{aligned} Pins(er_q^r) = \{v_j^x : a_{t,j} \in \mathcal{A}_k^\ell, map(a_{t,j}) = column_j, x_j \in \mathcal{X}_q^r\} \cup \\ \{v_t^y : a_{t,j} \in \mathcal{A}_k^\ell, map(a_{t,j}) = row_t, x_j \in \mathcal{X}_q^r\}. \end{aligned}$$

- **fold-send net** fs_q^r: The set of vertices connected by fs_q^r is given by

$$Pins(fs_q^r) = \{v_t^x : a_{i,t} \in \mathcal{A}_k^\ell, map(a_{i,t}) = column_t, y_i \in \mathcal{Y}_q^r\} \cup \\ \{v_i^y : a_{i,t} \in \mathcal{A}_k^\ell, map(a_{i,t}) = r_i, y_i \in \mathcal{Y}_q^r\}.$$

- **fold-receive net** fr_q^r: The set of the vertices that are connected by fr_q^r is exactly the same as that of the fold-receive net described in Section 4.2.

As in Section 4.2, each message net is assigned a cost of t_s, as they encapsulate the latency cost.

3) Obtain a bipartition Π. Use a partitioner to bipartition $\mathcal{H}_k^\ell$ and obtain $\Pi = \{\mathcal{V}_{2k}^{\ell+1}, \mathcal{V}_{2k+1}^{\ell+1}\}$.

4) Derive bipartitions $(\mathcal{A}_{2k}^{\ell+1}, \mathcal{A}_{2k+1}^{\ell+1})$, $(\mathcal{X}_{2k}^{\ell+1}, \mathcal{X}_{2k+1}^{\ell+1})$ *and* $(\mathcal{Y}_{2k}^{\ell+1}, \mathcal{Y}_{2k+1}^{\ell+1})$ *from* Π. The same as lines 10–12 of Algorithm 6.

Figure 4.6 displays the medium-grain hypergraph $\mathcal{H}_1^2 = (\mathcal{V}_1^2, \mathcal{N}_1^2)$, which is formed during the RB process for the SpMV instance given in Figure 4.3. Recall that $\mathcal{A}_1^2 = \{a_{1,3}, a_{1,7}, a_{2,3}, a_{2,4}, a_{4,5}, a_{4,7}\}$, $\mathcal{X}_1^2 = \{x_3, x_7\}$, and $\mathcal{Y}_1^2 = \{y_1, y_4\}$. Note that the figure also displays the message nets, hence it is between steps 2 and 3. Also note that $\mathcal{V}_1^2$ contains vertices representing vector entries that are neither in $\mathcal{X}_1^2$ nor in $\mathcal{Y}_1^2$. These vertices are v_4^x, v_5^x, and v_2^y. The assignments of the nonzeros that are computed by the mentioned heuristic are also given in the figure. The assignment information, i.e., $map(\cdot)$ function, is utilized while forming the sets of vertices connected by the nets. For example, net n_3^x connects v_1^y due to $map(a_{1,3}) = row_1$. As seen in the figure, there are four message nets in $\mathcal{H}_1^2$ as in the fine-grain hypergraph $\mathcal{H}_1^2$ given in Figure 4.4. This is because the existency conditions of the message nets in the proposed fine- and medium-grain models are the same. Note that the sets of vertices connected by expand-send net es_2^2 and receive-fold net r_2^2 are the same in Figures 4.4 and 4.6. Expand-receive net er_0^3 connects vertices v_4^x and v_5^y since $\mathcal{P}_1^2$ receives $\{x_4, x_5\}$ from $\mathcal{P}_0^3$ due to nonzeros in $\{a_{2,4}, a_{4,5}\}$, and $map(a_{2,4}) = column_4$ and $map(a_{4,5}) = column_5$. Fold-send net fs_1^3 connects vertices v_4^x and v_2^y since $\mathcal{P}_1^2$ sends the partial result

assignments

$a_{1,3}$	row_1
$a_{1,7}$	row_1
$a_{2,3}$	row_2
$a_{2,4}$	$column_4$
$a_{4,5}$	$column_5$
$a_{4,7}$	row_4

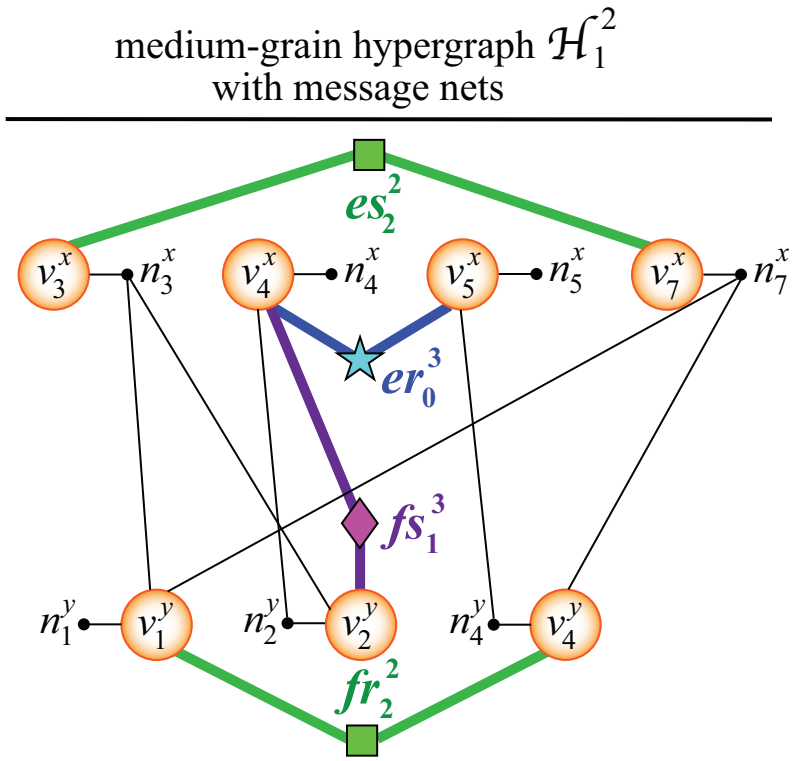


Figure 4.6: The medium-grain hypergraph $\mathcal{H}_1^2$ formed during the RB process for the SpMV instance given in Figure 4.3 and the message nets.

$a_{2,3}x_3 + a_{2,4}x_4$ to $\mathcal{P}_1^3$ due to nonzeros in $\{a_{2,3}, a_{2,4}\}$, and $map(a_{2,3}) = row_2$ and $map(a_{2,4}) = column^4$.

4.3.1 Adaptation for conformality constraint

As in the model proposed in Section 4.2, the model proposed in Section 4.3 does not impose the conformality constraint either. However, the technique for adapting the proposed medium-grain model for obeying the conformality constraint slightly differs from that described in Section 4.2.1. Vertex set $\mathcal{V}_k^\ell$ of the medium-grain hypergraph $\mathcal{H}_k^\ell$ contains the amalgamated vertex v_i if at least one of the following conditions holds:

- $x_i \in \mathcal{X}_k^\ell$, or equivalently, $y_i \in \mathcal{Y}_k^\ell$.

- There exists $a_{t,i} \in \mathcal{A}_k^\ell$ such that $map(a_{t,i}) = column_i$.
- There exists $a_{i,t} \in \mathcal{A}_k^\ell$ such that $map(a_{i,t}) = row_i$.

The weight of v_i is then formulated as

$$w(v_i) = |\{a_{t,i} : a_{t,i} \in \mathcal{A}_k^\ell \text{ and } map(a_{t,i}) = column_i\}| + \\ |\{a_{i,t} : a_{i,t} \in \mathcal{A}_k^\ell \text{ and } map(a_{i,t}) = row_i\}|.$$

As described in Section 4.2.1, each net in $\mathcal{H}_k^\ell$ that connects v_i^x or v_i^y now connects the amalgamated vertex v_i.

4.4 Delayed addition and thresholding for message nets

The preliminary experiments show that adding message nets to the hypergraphs at each RB step increases the total volume of communication drastically. This increase can be attributed to the bipartitionings in the early levels of the RB tree as follows. Note that there are only a few parts in the early levels of the RB tree compared to the latter levels. Each of these early parts represent a large processor group, so, the messages among these large groups of processors are very difficult to refrain from. In hypergraph partitioning terms, message nets in the hypergraphs at the early levels of the RB tree connect larger numbers of vertices and their cost is also much larger than the cost of volume nets since t_s is much larger than t_w in modern architectures. Hence, trying to save these big high-cost nets from the cut is very difficult. While the partitioning tool is trying to save these nets from the cut in these bipartitionings, it may be causing drastically larger numbers of volume nets to be cut. In addition, those cut volume nets will turn into new messages hence message nets in the latter levels of the RB tree. Therefore, adding message nets in the early levels of the RB tree may be adversely affecting the overall partition quality in multiple ways.

The RB approach provides the ability to change/adjust the partitioning decisions/parameters in the individual RB steps for the sake of the overall partition quality. In our model, this flexibility allows us to exploit the trade-off between the bandwidth and latency costs by selectively deciding whether to add message nets or not in each bipartitioning. Although there can be many parameters to be considered while making this decision in each single bipartition, we only use the level information for simplicity. That is, the addition of the message nets is delayed until the ρth RB-tree. In other words, the bipartitionings in level ℓ are carried out only with volume nets for some $1 \leq \ell < \rho \leq \log K$. Thus, message nets enter the picture when their size (in terms of the number of vertices that they connect) is not as large as those in the top-level hypergraphs.

Another solution for not having big high-cost nets is eliminating the message-nets whose size is larger than a threshold. That is, a message net m with $Pins(m) > t$ is not added for some $t > 0$ although message net addition is enabled in the corresponding bipartition. This approach also enables a selective approach for send and receive message nets. In the parallel SpMV algorithm given in Algorithm 5, the receive operations are performed by non-blocking MPI functions (i.e., `MPI_Irecv`), whereas the send operations are performed by blocking MPI functions (i.e., `MPI_Send`). When the maximum number of messages or the maximum volume of communication is considered as causing a bottleneck overhead, blocking send operations may be more limiting compared to non-blocking receive operations. Note that saving message net from the cut tends to assign the respective communication operations less number of processors, hence the maximum number of messages and maximun volume of communication may increase. Then, for the case send operations are more limiting than receive operations, the threshold for the send message nets should be smaller than that of the receive nets. This approach will decrease the increase in the maximum number of messages and the maximum volume of communication metrics.

4.5 Experiments

In this section, we compare five SpMV partitioning models which are listed below:

1. `FG`: the original fine-grain model which addresses only the bandwidth cost (described in Section 4.1.2),
2. `MG`: the original medium-grain model which addresses only the bandwidth cost (described in Section 4.1.3),
3. `FG-LM`: the proposed fine-grain model which simultaneously addresses the bandwidth and latency costs (described in Section 4.2),
4. `MG-LM`: the proposed medium-grain model which simultaneously addresses the bandwidth and latency costs (described in Section 4.3),
5. `1D-LM`: the one-dimensional (1D) coarse-grain model [1] which simultaneously addresses the bandwidth and latency costs.

Among the listed models, `FG-LM` and `MG-LM` are the proposed models whereas the others constitute the baseline algorithms. The suffix `-LM` stands for "latency minimized". In all `-LM` models, each message net is assigned a cost of 50, whereas each volume net is assigned unit cost, i.e., 1. The reasoning for this assignment is given in [1].

Note that the first four of the compared models obtain two-dimensional nonzero-based partitions of the input sparse matrix. In all compared models, the partitioning constraint corresponds to balancing the computational loads of processors, i.e., the number of nonzeros assigned to processors.

For evaluating the performance of the compared models, two groups of performance metrics are utilized: the partition statistics and the runtime results of parallel SpMV. The partition statistics include the total/maximum volume of communication, total/maximum number of messages communicated, the computation load imbalance and the partitioning time. The maximum volume and the

maximum number of messages are computed over the volume of data and the number of messages sent by processors, respectively.

Note that the compared models are all based-on hypergraph partitioning. In all these models, the hypergraphs are bipartitioned using PaToH [19] with the default setting. For the resulting K-way partitions to have at most $\epsilon = 0.10$ imbalance ratio on the part weights, the maximum allowable imbalance ratio in each bipartition is set to

$$(1+\epsilon)^{\frac{1}{\log K}} - 1 = (1.10)^{\frac{1}{\log K}} - 1.$$

The models are tested for five different number of parts/processors, $K \in \{64, 128, 256, 512, 1024\}$.

The parallel SpMV is implemented using the PETSc toolkit [56] and run on a Blue Gene/Q system using the partitions provided by the compared models. A node on Blue Gene/Q system consists of 16 PowerPC A2 processors with 1.6 GHz clock frequency and 16 GB memory. The nodes are connected by a 5D torus chip-to-chip network.

The test matrices are obtained from the SuiteSparse Matrix Collection [47], which is formerly known as the UFL Sparse Matrix Collection. We consider the parallel SpMV with conformal input and output vector partitions as it is more common for the applications in which SpMV is used as a kernel operation. Hence, only the square matrices are considered. Among all square matrices in SuiteSparse Matrix Collection, the matrices that have at least 50 million nonzeros are excluded. Based on the remaining matrices, we formed a different dataset for each K value, i.e., the number of parts/processors. For each K value, we excluded the matrices that have at most $100K$ nonzeros or at most $50K$ rows/columns from the dataset corresponding to that K value. We also excluded the matrices that have at least $100000K$ nonzeros from the corresponding dataset. In the datasets formed for K values of 64, 128, 256, 512 and 1024, there are 833, 730, 616, 475 and 316 matrices, respectively. The union of these datasets contains a total of 978 test matrices.

4.5.1 Parameter tuning for proposed models

In this section, we analyze the effect of the parameters described in Section 4.4 on the partition statistics.

4.5.1.1 Delayed-addition parameter: ρ

Recall that in the delayed message-net addition scheme described in Section 4.4, the message net addition is delayed until the ρth level of the RB tree. That is, the hypergraphs in the ℓth level where $\ell < \rho$ contain only the volume nets.

Table 4.2 presents the average partition statistics of the delayed message-net addition scheme in the fine-grain model for four different ρ values, in comparison against non-delayed `FG-LM` and the original `FG`. The tested ρ values are $\log K - 4$, $\log K - 3$, $\log K - 2$, $\log K - 1$. Note that message nets are utilized in ℓ levels when $\rho = \log K - \ell$. Also note that the non-delayed `FG-LM`, which is described in Section 4.2, corresponds to the delayed `FG-LM` with $\rho \leq 1$. In a dual manner, `FG` corresponds to the delayed `FG-LM` with $\rho \geq \log K$.

Table 4.2 presents the results of the six versions of the fine-grain model for each K value. For a K value, the first and second rows display the results of `FG` and non-delayed `FG-LM`, respectively. The next four rows display the results of the delayed versions of `FG-LM`, each with a different ρ value. The second column displays the model name for `FG` and non-delayed `FG-LM`, and ρ value for the delayed versions of `FG-LM`. The third column displays computational imbalance value in percent, which is computed as

$$\frac{\max_{1 \leq k \leq K} nnz(\mathcal{A}_k)}{\frac{nnz(\mathcal{A})}{K}} - 1.$$

The fourth and fifth columns display the maximum and total value for the volume of communication handled by processors (i.e., bandwidth cost), whereas the sixth and seventh columns display them for the number of messages (i.e., latency cost). the eighth column displays the partitioning time. The columns mentioned so far

present the actual results, while the rest of the columns (from the ninth to the fourteenth) display the results normalized with respect to those of `FG`.

As seen in Table 4.2, `FG-LM` reduces the total number of messages compared to `FG` by 55%, 58%, 62%, 61% and 58% on average for 64, 128, 256, 512 and 1024 processors, respectively. `FG-LM` reduces maximum number of messages compared to `FG-LM`, by 22%, 18% and 7% on average for 64, 128 and 256 processors, while increasing it by 4% and 16% for 512 and 1024 processors, respectively. These reductions can be attributed to the high reductions in total number of messages, whereas these increases can be explained by the clustered messages in some of the processors. As mentioned in Section 4.4, `FG-LM` causes drastic increases in the volume-based metrics compared to `FG`. In total volume, the average increase is 68%, 76%, 85%, 99% and 103% for 64, 128, 256, 512 and 1024 processors, respectively. In maximum volume, it is even worse and 213%, 278%, 376%, 445% and 493% for 64, 128, 256, 512 and 1024 processors, respectively. These increase rates in the volume-based metrics are not tolerable when compared to the reductions achieved in latency-based metrics. As seen in the table, the respective increase rates in the delayed versions are not as large as the ones in the non-delayed `FG-LM`. Nonetheless, the results of $\rho = \log K - 4$ for the latency-based metrics are so close to those of non-delayed `FG-LM`. This implies that having message nets in only last four levels is still good enough for reducing latency-based metrics. As seen in the table, as the number of levels where the message net addition is applied decreases, the increase rate in the volume-based metrics decreases faster than the decrease in the decrease rate in the total number of messages. Therefore, we pick $\rho = \log K - 2$ as the delayed version which gives the most tolerable increase rates in the volume-based metrics while keeping the decrease in the latency-based metrics large enough, i.e., around 50%, for the total number of messages.

Table 4.3 presents the average results for `MG`, non-delayed `MG-LM` and the delayed `MG-LM` with four different ρ values. The formatting of this table is the same as that of Table 4.2. As seen in the table, the reduction rates in the total number of messages achieved by `MG-LM` over `MG` are not as large as those achieved by `FG-LM` over `FG`. `MG-LM` achieves 44%, 46%, 48%, 46% and 43% reductions in total

Table 4.2: Average partition statistics of the delayed message-net addition scheme in the fine-grain model for four different ρ values, in comparison against `FG` and non-delayed `FG-LM`.

		actual values						normalized w.r.t. `FG`					
			volume		message				volume		message		
K	model/ρ	imb	max	total	max	total	time	imb	max	total	max	total	time
64	`FG`	0.91	413	11811	32	968	7.7	-	-	-	-	-	-
	`FG-LM`	0.89	1293	19804	25	436	8.7	0.98	3.13	1.68	0.78	0.45	1.14
	$\lg K-4$	0.89	1232	19733	24	440	8.0	0.98	2.98	1.67	0.75	0.45	1.05
	$\lg K-3$	0.96	1107	19241	24	451	7.9	1.05	2.68	1.63	0.75	0.47	1.03
	$\lg K-2$	0.91	906	18155	24	498	7.7	1.00	2.19	1.54	0.75	0.51	1.01
	$\lg K-1$	0.88	662	15871	26	628	7.4	0.97	1.60	1.34	0.81	0.65	0.96
128	`FG`	1.11	484	24670	45	2332	16.4	-	-	-	-	-	-
	`FG-LM`	1.10	1830	43338	37	972	19.0	0.99	3.78	1.76	0.82	0.42	1.16
	$\lg K-4$	1.20	1654	43058	35	990	17.9	1.08	3.42	1.75	0.78	0.42	1.09
	$\lg K-3$	1.11	1425	41918	34	1035	17.7	1.00	2.94	1.70	0.76	0.44	1.08
	$\lg K-2$	1.08	1112	38972	34	1166	17.1	0.97	2.30	1.58	0.76	0.50	1.04
	$\lg K-1$	1.09	779	33101	36	1512	16.3	0.98	1.61	1.34	0.80	0.65	0.99
256	`FG`	1.36	567	52357	60	5560	40.9	-	-	-	-	-	-
	`FG-LM`	1.33	2700	96802	56	2120	47.9	0.98	4.76	1.85	0.93	0.38	1.17
	$\lg K-4$	1.27	2213	94983	49	2186	44.9	0.93	3.90	1.81	0.82	0.39	1.10
	$\lg K-3$	1.37	1818	90802	46	2317	44.0	1.01	3.21	1.73	0.77	0.42	1.07
	$\lg K-2$	1.33	1346	82651	46	2694	42.6	0.98	2.37	1.58	0.77	0.48	1.04
	$\lg K-1$	1.29	926	69572	49	3574	40.7	0.95	1.63	1.33	0.82	0.64	0.99
512	`FG`	1.67	584	92141	72	11186	77.9	-	-	-	-	-	-
	`FG-LM`	1.67	3183	183332	75	4349	90.9	1.00	5.45	1.99	1.04	0.39	1.17
	$\lg K-4$	1.75	2369	178035	61	4491	85.1	1.05	4.06	1.93	0.85	0.40	1.09
	$\lg K-3$	1.68	1888	168597	57	4796	83.2	1.01	3.23	1.83	0.79	0.43	1.07
	$\lg K-2$	1.58	1384	151451	56	5599	80.7	0.95	2.37	1.64	0.78	0.50	1.04
	$\lg K-1$	1.62	933	124694	60	7358	76.4	0.97	1.60	1.35	0.83	0.66	0.98
1024	`FG`	1.87	530	165923	69	20209	156.2	-	-	-	-	-	-
	`FG-LM`	1.92	3142	337216	80	8414	182.4	1.03	5.93	2.03	1.16	0.42	1.17
	$\lg K-4$	1.90	2261	326383	62	8656	172.3	1.02	4.27	1.97	0.90	0.43	1.10
	$\lg K-3$	1.85	1764	304878	57	9188	167.1	0.99	3.33	1.84	0.83	0.45	1.07
	$\lg K-2$	1.85	1260	269921	55	10576	159.9	0.99	2.38	1.63	0.80	0.52	1.02
	$\lg K-1$	1.81	861	220174	59	13606	155.0	0.97	1.62	1.33	0.86	0.67	0.99

Table 4.3: Average partition statistics of the delayed message-net addition scheme in the medium-grain model for four different ρ values, in comparison against `MG` and non-delayed `MG-LM`.

		actual values						normalized w.r.t. `MG`					
			volume		message				volume		message		
K	model/ρ	imb	max	total	max	total	time	imb	max	total	max	total	time
64	`MG`	0.90	412	11655	31	928	3.9	-	-	-	-	-	-
	`MG-LM`	0.87	835	16985	25	523	4.5	0.97	2.03	1.46	0.81	0.56	1.17
	$\lg K - 4$	0.82	820	16875	25	525	4.3	0.91	1.99	1.45	0.81	0.57	1.12
	$\lg K - 3$	0.85	773	16581	24	534	4.3	0.94	1.88	1.42	0.77	0.58	1.09
	$\lg K - 2$	0.87	694	15765	25	566	4.1	0.97	1.68	1.35	0.81	0.61	1.07
	$\lg K - 1$	0.91	573	14227	26	662	4.0	1.01	1.39	1.22	0.84	0.71	1.02
128	`MG`	1.13	482	24256	44	2217	8.1	-	-	-	-	-	-
	`MG-LM`	1.04	1055	36227	36	1202	9.7	0.92	2.19	1.49	0.82	0.54	1.19
	$\lg K - 4$	1.11	1012	35882	35	1214	9.0	0.98	2.10	1.48	0.80	0.55	1.12
	$\lg K - 3$	1.11	945	34857	35	1242	8.9	0.98	1.96	1.44	0.80	0.56	1.10
	$\lg K - 2$	1.14	821	33061	35	1329	8.6	1.01	1.70	1.36	0.80	0.60	1.07
	$\lg K - 1$	1.14	662	29537	37	1583	8.2	1.01	1.37	1.22	0.84	0.71	1.01
256	`MG`	1.48	558	49867	57	5103	19.1	-	-	-	-	-	-
	`MG-LM`	1.42	1368	77479	50	2674	23.4	0.96	2.45	1.55	0.88	0.52	1.23
	$\lg K - 4$	1.45	1264	77227	48	2735	21.9	0.98	2.27	1.55	0.84	0.54	1.15
	$\lg K - 3$	1.45	1148	74341	47	2809	21.7	0.98	2.06	1.49	0.82	0.55	1.14
	$\lg K - 2$	1.44	969	69159	47	3066	21.0	0.97	1.74	1.39	0.82	0.60	1.10
	$\lg K - 1$	1.35	776	61070	50	3695	19.9	0.91	1.39	1.22	0.88	0.72	1.04
512	`MG`	1.91	588	91856	67	10265	39.7	-	-	-	-	-	-
	`MG-LM`	1.75	1508	147893	64	5537	50.0	0.92	2.56	1.61	0.96	0.54	1.26
	$\lg K - 4$	1.85	1337	144080	59	5610	46.3	0.97	2.27	1.57	0.88	0.55	1.17
	$\lg K - 3$	1.84	1197	137584	57	5785	45.4	0.96	2.04	1.50	0.85	0.56	1.14
	$\lg K - 2$	1.85	1016	126729	57	6327	43.5	0.97	1.73	1.38	0.85	0.62	1.10
	$\lg K - 1$	1.82	809	110850	59	7522	41.5	0.95	1.38	1.21	0.88	0.73	1.04
1024	`MG`	2.05	530	165722	65	18692	82.2	-	-	-	-	-	-
	`MG-LM`	1.97	1394	267677	65	10681	104.4	0.96	2.63	1.62	1.00	0.57	1.27
	$\lg K - 4$	2.03	1209	258857	59	10809	97.7	0.99	2.28	1.56	0.91	0.58	1.19
	$\lg K - 3$	2.00	1094	245893	56	11061	95.9	0.98	2.06	1.48	0.86	0.59	1.17
	$\lg K - 2$	2.11	935	226901	56	11926	92.1	1.03	1.76	1.37	0.86	0.64	1.12
	$\lg K - 1$	2.03	739	197574	57	13924	85.7	0.99	1.39	1.19	0.88	0.74	1.04

number of messages, for 64, 128, 256, 512 and 1024 processors, respectively. It reduces maximum number of messages by 19%, 18%, 12%, 4% and 0% for 64, 128, 256, 512 and 1024 processors, respectively. Similarly, the increase rates in the volume-based metrics caused by `MG-LM` over `MG` are not as large as those caused by `FG-LM` over `FG`. The increase rates in total volume caused by `MG-LM` are 46%, 49%, 55%, 61% and 62% for 64, 128, 256, 512 and 1024 processors, respectively. The increase rates in maximum volume are 103%, 119%, 145%, 156% and 163% for 64, 128, 256, 512 and 1024 processors, respectively. With the same reasoning made for the delayed versions for the variants of the fine-grain model given in the previous table, we pick $\rho = \log K - 2$ as the delayed version with the best results when all metrics are considered.

As seen in Tables 4.2 and 4.3, the variants of the fine-grain model are more extreme in terms of the increase and decrease rates compared to those of the medium-grain model. This can be attributed to the fact that the fine-grain model has a larger solution space hence the metric with a higher importance in a variant of the fine-grain model can be targeted better than the others.

4.5.1.2 Thresholding parameters: τ_s and τ_r

Recall that in the net-thresholding scheme described in Section 4.4, only the message nets with the number of connected vertices smaller than or equal to a threshold $t > 0$ are allowed to be added to the hypergraphs in the RB process. We denote the thresholds for send-message nets and receive-message nets as τ_s and τ_r, respectively. That is, an expand-send/fold-send net s is not added to the respective hypergraph if $Pins(s) > \tau_s$ even though the message net addition is active. Similarly, an expand-receive/fold-receive net r is not added to the respective hypergraph if $Pins(r) > \tau_r$ even though the message net addition is active.

Tables 4.4 and 4.5 present the average partition statistics of the net-thresholding scheme for the variants of the fine-grain model for different combinations of (τ_s, τ_r) values in comparison against the `FG` and `FG-LM` schemes.

Table 4.4: Average partition statistics of the net-thresholding scheme in the fine-grain model for nine different combinations of (τ_s, τ_r) values, in comparison against `FG` and `FG-LM` with all message nets included, for 64, 128 and 256 processors.

		actual values						normalized w.r.t. `FG`					
			volume		message				volume		message		
K	model	imb	max	total	max	total	time	imb	max	total	max	total	time
64	`FG`	0.91	413	11811	32	968	7.7	-	-	-	-	-	-
	`FG-LM`	0.89	1293	19804	25	436	8.7	0.98	3.13	1.68	0.78	0.45	1.14
	$15, 15$	0.87	468	12382	31	842	7.4	0.96	1.13	1.05	0.97	0.87	0.96
	$15, 30$	0.87	509	12825	30	791	7.4	0.96	1.23	1.09	0.94	0.82	0.97
	$15, 50$	0.88	542	13267	29	753	7.4	0.97	1.31	1.12	0.91	0.78	0.97
	$30, 15$	0.86	518	12819	31	791	7.4	0.95	1.25	1.09	0.97	0.82	0.97
	$30, 30$	0.93	538	13109	30	761	7.4	1.02	1.30	1.11	0.94	0.79	0.97
	$30, 50$	0.89	573	13541	29	726	7.4	0.98	1.39	1.15	0.91	0.75	0.97
	$50, 15$	0.87	562	13267	31	751	7.4	0.96	1.36	1.12	0.97	0.78	0.97
	$50, 30$	0.85	580	13526	30	724	7.4	0.93	1.40	1.15	0.94	0.75	0.97
	$50, 50$	0.83	601	13840	29	699	7.4	0.91	1.46	1.17	0.91	0.72	0.96
128	`FG`	1.11	484	24670	45	2332	16.4	-	-	-	-	-	-
	`FG-LM`	1.10	1830	43338	37	972	19.0	0.99	3.78	1.76	0.82	0.42	1.16
	$15, 15$	1.08	568	26186	43	1964	16.3	0.97	1.17	1.06	0.96	0.84	0.99
	$15, 30$	1.04	618	27171	42	1841	16.4	0.94	1.28	1.10	0.93	0.79	1.00
	$15, 50$	1.01	669	28159	40	1751	16.3	0.91	1.38	1.14	0.89	0.75	1.00
	$30, 15$	1.05	637	27176	43	1838	16.3	0.95	1.32	1.10	0.96	0.79	0.99
	$30, 30$	1.01	669	27825	42	1763	16.3	0.91	1.38	1.13	0.93	0.76	1.00
	$30, 50$	1.02	711	28702	41	1678	16.4	0.92	1.47	1.16	0.91	0.72	1.00
	$50, 15$	1.01	707	28156	44	1741	16.4	0.91	1.46	1.14	0.98	0.75	1.00
	$50, 30$	1.00	728	28789	42	1671	16.3	0.90	1.50	1.17	0.93	0.72	0.99
	$50, 50$	1.05	757	29461	41	1615	16.3	0.95	1.56	1.19	0.91	0.69	0.99
256	`FG`	1.36	567	52357	60	5560	40.9	-	-	-	-	-	-
	`FG-LM`	1.33	2700	96802	56	2120	47.9	0.98	4.76	1.85	0.93	0.38	1.17
	$15, 15$	1.26	706	56218	58	4539	40.9	0.93	1.25	1.07	0.97	0.82	1.00
	$15, 30$	1.27	773	58452	56	4258	40.7	0.93	1.36	1.12	0.93	0.77	0.99
	$15, 50$	1.21	835	60864	54	4043	40.8	0.89	1.47	1.16	0.90	0.73	1.00
	$30, 15$	1.22	793	58418	59	4251	41.0	0.90	1.40	1.12	0.98	0.76	1.00
	$30, 30$	1.21	827	60086	57	4087	40.8	0.89	1.46	1.15	0.95	0.74	1.00
	$30, 50$	1.23	900	62393	55	3879	41.1	0.90	1.59	1.19	0.92	0.70	1.00
	$50, 15$	1.20	879	61099	59	4037	40.8	0.88	1.55	1.17	0.98	0.73	1.00
	$50, 30$	1.23	908	62516	58	3877	40.9	0.90	1.60	1.19	0.97	0.70	1.00
	$50, 50$	1.17	952	64041	56	3729	40.5	0.86	1.68	1.22	0.93	0.67	0.99

Table 4.5: Average partition statistics of the net-thresholding scheme in the fine-grain model for nine different combinations of (τ_s, τ_r) values, in comparison against `FG` and `FG-LM` with all message nets included, for 512 and 1024 processors.

		actual values						normalized w.r.t. `FG`					
			volume		message				volume		message		
K	model	imb	max	total	max	total	time	imb	max	total	max	total	time
512	`FG`	1.67	584	92141	72	11186	77.9	-	-	-	-	-	-
	`FG-LM`	1.67	3183	183332	75	4349	90.9	1.00	5.45	1.99	1.04	0.39	1.17
	$15, 15$	1.55	726	99582	71	9144	78.0	0.93	1.24	1.08	0.99	0.82	1.00
	$15, 30$	1.58	799	103937	69	8613	77.2	0.95	1.37	1.13	0.96	0.77	0.99
	$15, 50$	1.61	863	108497	66	8218	77.2	0.96	1.48	1.18	0.92	0.73	0.99
	$30, 15$	1.54	817	103490	72	8627	76.8	0.92	1.40	1.12	1.00	0.77	0.99
	$30, 30$	1.59	857	105937	69	8245	75.5	0.95	1.47	1.15	0.96	0.74	0.97
	$30, 50$	1.58	917	110302	67	7851	76.1	0.95	1.57	1.20	0.93	0.70	0.98
	$50, 15$	1.59	905	107850	73	8190	75.6	0.95	1.55	1.17	1.01	0.73	0.97
	$50, 30$	1.58	934	110637	70	7872	77.0	0.95	1.60	1.20	0.97	0.70	0.99
	$50, 50$	1.56	984	113629	68	7569	76.0	0.93	1.68	1.23	0.94	0.68	0.98
1024	`FG`	1.87	530	165923	69	20209	156.2	-	-	-	-	-	-
	`FG-LM`	1.92	3142	337216	80	8414	182.4	1.03	5.93	2.03	1.16	0.42	1.17
	$15, 15$	1.83	661	179065	69	17125	158.7	0.98	1.25	1.08	1.00	0.85	1.02
	$15, 30$	1.83	747	187975	67	16242	156.8	0.98	1.41	1.13	0.97	0.80	1.00
	$15, 50$	1.81	811	196236	66	15415	159.6	0.97	1.53	1.18	0.96	0.76	1.02
	$30, 15$	1.78	766	187379	71	16188	157.3	0.95	1.45	1.13	1.03	0.80	1.01
	$30, 30$	1.76	818	193762	69	15514	157.1	0.94	1.54	1.17	1.00	0.77	1.01
	$30, 50$	1.79	869	203544	67	14852	158.3	0.96	1.64	1.23	0.97	0.73	1.01
	$50, 15$	1.72	854	196052	73	15443	154.8	0.92	1.61	1.18	1.06	0.76	0.99
	$50, 30$	1.74	897	201763	70	14843	153.1	0.93	1.69	1.22	1.01	0.73	0.98
	$50, 50$	1.78	931	208023	68	14277	153.7	0.95	1.76	1.25	0.99	0.71	0.98

Table 4.4 displays the results for 64, 128 and 256 processors, whereas Table 4.5 displays the results for 512 and 1024 processors. Similar to Tables 4.2 and 4.3, for each K value, the first and second rows present the results of `FG` and `FG-LM` with all message nets, respectively. The next nine rows present the results of the net-thresholding scheme, each for a different combination of $\tau_s \in \{15, 30, 50\}$ and $\tau_r \in \{15, 30, 50\}$ values. In all variants of the net-thresholding scheme, the delayed parameter ρ is set to be $\log K - 2$, as described in Section 4.5.1.1. The formats of Tables 4.4 and 4.5 are the same as those of Tables 4.2 and 4.3.

As seen in Tables 4.4 and 4.5, the improvements obtained by the net-thresholding variants $(\tau_s = \alpha, \tau_r = \beta)$ and $(\tau_s = \beta, \tau_r = \alpha)$ in a metric different than maximum volume and maximum number of messages are generally the same for a specific K value. This is because such a metric is oblivous to the direction of the communication.

However, a variant with $(\tau_s = \alpha_1, \tau_r = \beta)$ results in less maximum volume and less maximum number of messages compared to a variant with $(\tau_s = \alpha_2, \tau_r = \beta)$ for $\alpha_1 < \alpha_2$. This is expected since including more send-message nets clusters more communication items to be sent in some processors. We pick τ_s as the smallest value tried for this parameter, i.e., $\tau = 15$, since it gives the smallest increase in maximum volume and the largest decrease in maximum number of messages compared to `FG`. However, we pick τ_r as the largest value tried for this parameter, i.e., $\tau = 50$, since it does not affect the mentioned metrics much.

Tables 4.6 and 4.7 present the net-thresholding results for the variant of the medium-grain model. Similar to the case of the fine-grain model, we pick $\tau_s = 15$ and $\tau = 50$ as they give the best combination of the increase and decrease rates.

Table 4.6: Average partition statistics of the net-thresholding scheme in the medium-grain model for nine different combinations of (τ_s, τ_r) values, in comparison against `MG` and `MG-LM` with all message nets included, for 64, 128 and 256 processors.

		actual values						normalized w.r.t. `MG`					
			volume		message				volume		message		
K	model	imb	max	total	max	total	time	imb	max	total	max	total	time
64	`MG`	0.90	412	11655	31	928	3.9	-	-	-	-	-	-
	`MG-LM`	0.87	835	16985	25	523	4.5	0.97	2.03	1.46	0.81	0.56	1.17
	$15, 15$	0.90	462	12313	30	814	4.0	1.00	1.12	1.06	0.97	0.88	1.04
	$15, 30$	0.91	495	12800	29	768	4.1	1.01	1.20	1.10	0.94	0.83	1.05
	$15, 50$	0.87	521	13205	28	732	4.1	0.97	1.26	1.13	0.90	0.79	1.06
	$30, 15$	0.90	500	12804	29	764	4.1	1.00	1.21	1.10	0.94	0.82	1.05
	$30, 30$	0.85	521	13034	29	741	4.1	0.94	1.26	1.12	0.94	0.80	1.05
	$30, 50$	0.85	545	13420	28	710	4.1	0.94	1.32	1.15	0.90	0.77	1.05
	$50, 15$	0.82	536	13237	29	729	4.1	0.91	1.30	1.14	0.94	0.79	1.04
	$50, 30$	0.87	549	13431	29	709	4.1	0.97	1.33	1.15	0.94	0.76	1.05
	$50, 50$	0.84	562	13649	28	691	4.1	0.93	1.36	1.17	0.90	0.74	1.04
128	`MG`	1.13	482	24256	44	2217	8.1	-	-	-	-	-	-
	`MG-LM`	1.04	1055	36227	36	1202	9.7	0.92	2.19	1.49	0.82	0.54	1.19
	$15, 15$	1.10	559	25965	42	1887	8.3	0.97	1.16	1.07	0.95	0.85	1.03
	$15, 30$	1.10	600	26994	40	1775	8.4	0.97	1.24	1.11	0.91	0.80	1.03
	$15, 50$	1.08	634	27799	39	1690	8.4	0.96	1.32	1.15	0.89	0.76	1.04
	$30, 15$	1.08	611	27013	41	1775	8.3	0.96	1.27	1.11	0.93	0.80	1.03
	$30, 30$	1.13	635	27585	41	1715	8.4	1.00	1.32	1.14	0.93	0.77	1.04
	$30, 50$	1.10	658	28331	39	1644	8.5	0.97	1.37	1.17	0.89	0.74	1.05
	$50, 15$	1.06	649	27902	41	1692	8.4	0.94	1.35	1.15	0.93	0.76	1.04
	$50, 30$	1.08	669	28354	40	1646	8.5	0.96	1.39	1.17	0.91	0.74	1.05
	$50, 50$	1.12	691	28853	39	1603	8.4	0.99	1.43	1.19	0.89	0.72	1.04
256	`MG`	1.48	558	49867	57	5103	19.1	-	-	-	-	-	-
	`MG-LM`	1.42	1368	77479	50	2674	23.4	0.96	2.45	1.55	0.88	0.52	1.23
	$15, 15$	1.44	681	54772	55	4293	20.3	0.97	1.22	1.10	0.96	0.84	1.06
	$15, 30$	1.40	727	56981	53	4052	20.4	0.95	1.30	1.14	0.93	0.79	1.07
	$15, 50$	1.39	766	58981	52	3876	20.6	0.94	1.37	1.18	0.91	0.76	1.08
	$30, 15$	1.39	746	56998	55	4036	20.3	0.94	1.34	1.14	0.96	0.79	1.06
	$30, 30$	1.41	763	58166	54	3916	20.5	0.95	1.37	1.17	0.95	0.77	1.08
	$30, 50$	1.38	794	59847	52	3758	20.7	0.93	1.42	1.20	0.91	0.74	1.08
	$50, 15$	1.36	783	58933	55	3884	20.4	0.92	1.40	1.18	0.96	0.76	1.07
	$50, 30$	1.40	805	59890	53	3775	20.5	0.95	1.44	1.20	0.93	0.74	1.07
	$50, 50$	1.39	825	60936	52	3681	20.4	0.94	1.48	1.22	0.91	0.72	1.07

Table 4.7: Average partition statistics of the net-thresholding scheme in the medium-grain model for nine different combinations of (τ_s, τ_r) values, in comparison against `MG` and `MG-LM` with all message nets included, for 512 and 1024 processors.

		actual values						normalized w.r.t. `MG`					
			volume		message				volume		message		
K	model	imb	max	total	max	total	time	imb	max	total	max	total	time
512	`MG`	1.91	588	91856	67	10265	39.7	-	-	-	-	-	-
	`MG-LM`	1.75	1508	147893	64	5537	50.0	0.92	2.56	1.61	0.96	0.54	1.26
	$15, 15$	1.75	710	100363	66	8629	43.0	0.92	1.21	1.09	0.99	0.84	1.08
	$15, 30$	1.82	757	103475	64	8177	42.4	0.95	1.29	1.13	0.96	0.80	1.07
	$15, 50$	1.80	785	108128	62	7878	43.7	0.94	1.34	1.18	0.93	0.77	1.10
	$30, 15$	1.76	773	104449	66	8170	42.5	0.92	1.31	1.14	0.99	0.80	1.07
	$30, 30$	1.85	803	106445	65	7956	42.5	0.97	1.37	1.16	0.97	0.78	1.07
	$30, 50$	1.80	838	109963	63	7683	42.4	0.94	1.43	1.20	0.94	0.75	1.07
	$50, 15$	1.75	814	108019	66	7859	42.8	0.92	1.38	1.18	0.99	0.77	1.08
	$50, 30$	1.78	842	109801	64	7658	42.7	0.93	1.43	1.20	0.96	0.75	1.07
	$50, 50$	1.68	855	111668	62	7491	42.5	0.88	1.45	1.22	0.93	0.73	1.07
1024	`MG`	2.05	530	165722	65	18692	82.2	-	-	-	-	-	-
	`MG-LM`	1.97	1394	267677	65	10681	104.4	0.96	2.63	1.62	1.00	0.57	1.27
	$15, 15$	2.04	637	181918	64	16222	85.6	1.00	1.20	1.10	0.98	0.87	1.04
	$15, 30$	1.98	693	189551	63	15468	87.9	0.97	1.31	1.14	0.97	0.83	1.07
	$15, 50$	2.00	724	196443	61	14827	87.5	0.98	1.37	1.19	0.94	0.79	1.06
	$30, 15$	1.91	710	189740	65	15458	86.4	0.93	1.34	1.14	1.00	0.83	1.05
	$30, 30$	1.99	731	193037	63	15002	86.3	0.97	1.38	1.16	0.97	0.80	1.05
	$30, 50$	1.98	757	199297	61	14496	85.9	0.97	1.43	1.20	0.94	0.78	1.05
	$50, 15$	1.98	736	195599	64	14864	86.5	0.97	1.39	1.18	0.98	0.80	1.05
	$50, 30$	1.93	768	199212	62	14449	86.0	0.94	1.45	1.20	0.95	0.77	1.05
	$50, 50$	2.01	793	203012	62	14128	85.3	0.98	1.50	1.23	0.95	0.76	1.04

4.5.2 Comparison against coarse-grain model `1D-LM`

In Section 4.5.1, the parameters of the delayed-addition and thresholding schemes for the message nets in `FG-LM` and `MG-LM` models are set as follows.

$$\rho = \log K - 2, \tau_s = 15, \tau_r = 50.$$

Table 4.8 presents the average partition statistics for these `FG-LM` and `MG-LM` models, in comparison against their own baselines (`FG`, `MG`) and the general baseline algorithm, `1D-LM` [1], for 64, 128, 256, 512 and 1024 processors. The formatting of this table is the same as the other tables in this chapter.

As seen in Table 4.8, `1D-LM` results in the largest average computational imbalance value among the compared models, for each K value. This is because 1D models have less flexibility while maintaining the partitioning constraint in contrast to the 2D models. `FG` and `FG-LM` have similar reduction rates in this imbalance value compared to `1D-LM`, as well as `MG` and `MG-LM`. `FG-LM` obtains 29%, 47%, 52%, 49% and 45% average reductions in imbalance compared to `1D-LM`, for 64, 128, 256, 512 and 1024 processors. Similarly, `MG-LM` obtains 30%, 44%, 44%, 43% and 39% average reductions in imbalance compared to `1D-LM`, for 64, 128, 256, 512 and 1024 processors.

As seen in Table 4.8, all 2D models achieve better average results in volume-based metrics compared to `1D-LM`. This is because 2D models have a larger solution space for optimizing the bandwidth cost, especially `FG` and `MG`. Although `FG-LM` and `MG-LM` cause an increase in volume-based metrics respectively compared to `FG` and `MG`, they still achieve less maximum and total volume compared to `1D-LM`. `FG-LM` achieves 43%, 42%, 40%, 43% and 39% reductions in total volume compared to `1D-LM`, for 64, 128, 256, 512 and 1024 processors. The respective reduction rates in maximum volume are 28%, 21%, 14%, 8% and 2%. Similarly, `MG-LM` achieves 44%, 43%, 42%, 43% and 39% reductions in total volume compared to `1D-LM`, for 64, 128, 256, 512 and 1024 processors. The respective reduction rates in maximum volume are 31%, 25%, 21%, 16% and 13%.

Table 4.8: Average partition statistics of FG-LM and MG-LM in comparison against their baselines (FG and MG) and the general baseline algorithm, 1D-LM [1], for 64, 128, 256, 512 and 1024 processors.

		actual values						normalized w.r.t. 1D-LM					
			volume		message				volume		message		
K	model	imb	max	total	max	total	time	imb	max	total	max	total	time
	1D-LM	1.24	750	23479	16	416	2.4	-	-	-	-	-	-
	FG	0.91	413	11811	32	968	7.7	0.73	0.55	0.50	2.00	2.33	3.16
64	MG	0.90	412	11655	31	928	3.9	0.73	0.55	0.50	1.94	2.23	1.61
	FG-LM	0.88	542	13267	29	753	7.4	0.71	0.72	0.57	1.81	1.81	3.06
	MG-LM	0.87	521	13205	28	732	4.1	0.70	0.69	0.56	1.75	1.76	1.70
	1D-LM	1.92	851	48730	24	1007	5.2	-	-	-	-	-	-
	FG	1.11	484	24670	45	2332	16.4	0.58	0.57	0.51	1.88	2.32	3.17
128	MG	1.13	482	24256	44	2217	8.1	0.59	0.57	0.50	1.83	2.20	1.56
	FG-LM	1.01	669	28159	40	1751	16.3	0.53	0.79	0.58	1.67	1.74	3.15
	MG-LM	1.08	634	27799	39	1690	8.4	0.56	0.75	0.57	1.63	1.68	1.62
	1D-LM	2.50	968	101565	33	2448	13.2	-	-	-	-	-	-
	FG	1.36	567	52357	60	5560	40.9	0.54	0.59	0.52	1.82	2.27	3.11
256	MG	1.48	558	49867	57	5103	19.1	0.59	0.58	0.49	1.73	2.08	1.45
	FG-LM	1.21	835	60864	54	4043	40.8	0.48	0.86	0.60	1.64	1.65	3.10
	MG-LM	1.39	766	58981	52	3876	20.6	0.56	0.79	0.58	1.58	1.58	1.56
	1D-LM	3.18	933	190222	40	5082	27.9	-	-	-	-	-	-
	FG	1.67	584	92141	72	11186	77.9	0.53	0.63	0.48	1.80	2.20	2.79
512	MG	1.91	588	91856	67	10265	39.7	0.60	0.63	0.48	1.68	2.02	1.42
	FG-LM	1.61	863	108497	66	8218	77.2	0.51	0.92	0.57	1.65	1.62	2.77
	MG-LM	1.80	785	108128	62	7878	43.7	0.57	0.84	0.57	1.55	1.55	1.57
	1D-LM	3.28	830	324324	40	9562	53.4	-	-	-	-	-	-
	FG	1.87	530	165923	69	20209	156.2	0.57	0.64	0.51	1.73	2.11	2.92
1024	MG	2.05	530	165722	65	18692	82.2	0.63	0.64	0.51	1.63	1.95	1.54
	FG-LM	1.81	811	196236	66	15415	159.6	0.55	0.98	0.61	1.65	1.61	2.99
	MG-LM	2.00	724	196443	61	14827	87.5	0.61	0.87	0.61	1.53	1.55	1.64

As seen in Table 4.8, all 2D models have worse average results in latency-based metrics compared to `1D-LM`. This is explained by the two distinct communication phases in 2D models while 1D models have only a single one. Despite this, `FG-LM` and `MG-LM` reduces the increase in the latency-based metrics over `1D-LM`, respectively compared to `FG` and `MG`. `FG-LM` reduces the increase in total number of messages of `FG` over `1D-LM`, from 133%, 132%, 127%, 120% and 111%, to 81%, 74%, 65%, 62% and 61%, for 64, 128, 256, 512 and 1024 processors, respectively. Similarly, `MG-LM` reduces the increase in total number of messages of `MG` over `1D-LM`, from 123%, 120%, 108%, 102% and 95%, to 76%, 68%, 58%, 55% and 55%, for 64, 128, 256, 512 and 1024 processors, respectively. Similar reductions are achieved in the increase in maximum number of messages by `FG-LM` and `MG-LM` as well.

When we compare `FG-LM` and `MG-LM` against each other, `MG-LM` outperforms `FG-LM` in terms of maximum volume, maximum number of messages and total number of messages. They are comparable in terms of total volume, however, `FG-LM` outperforms `MG-LM` in terms of computational imbalance. When we also take the partitioning time into account, `MG-LM` becomes a better alternative than `FG-LM`, for the comparison held against `1D-LM`. For this reason, we only present the parallel runtime results of `MG` and `MG-LM` while comparing them against those of `1D-LM` in the next section.

4.5.3 Parallel SpMV runtime results

Figure 4.7 presents the strong scaling results for parallel SpMV for each of the models `1D-LM`, `MG` and `MG-LM` for $K \in \{64, 128, 256, 512\}$ on eight different matrices. These matrices vary in both their sizes and problem kinds. The number of nonzeros in these matrices varies between 330870 and 57708624, whereas the number of rows/columns varies between 45101 and 23947347. The problem kinds include 2D/3D problem, computational fluid dynamics problem, circuit simulation problem and directed/undirected graphs.

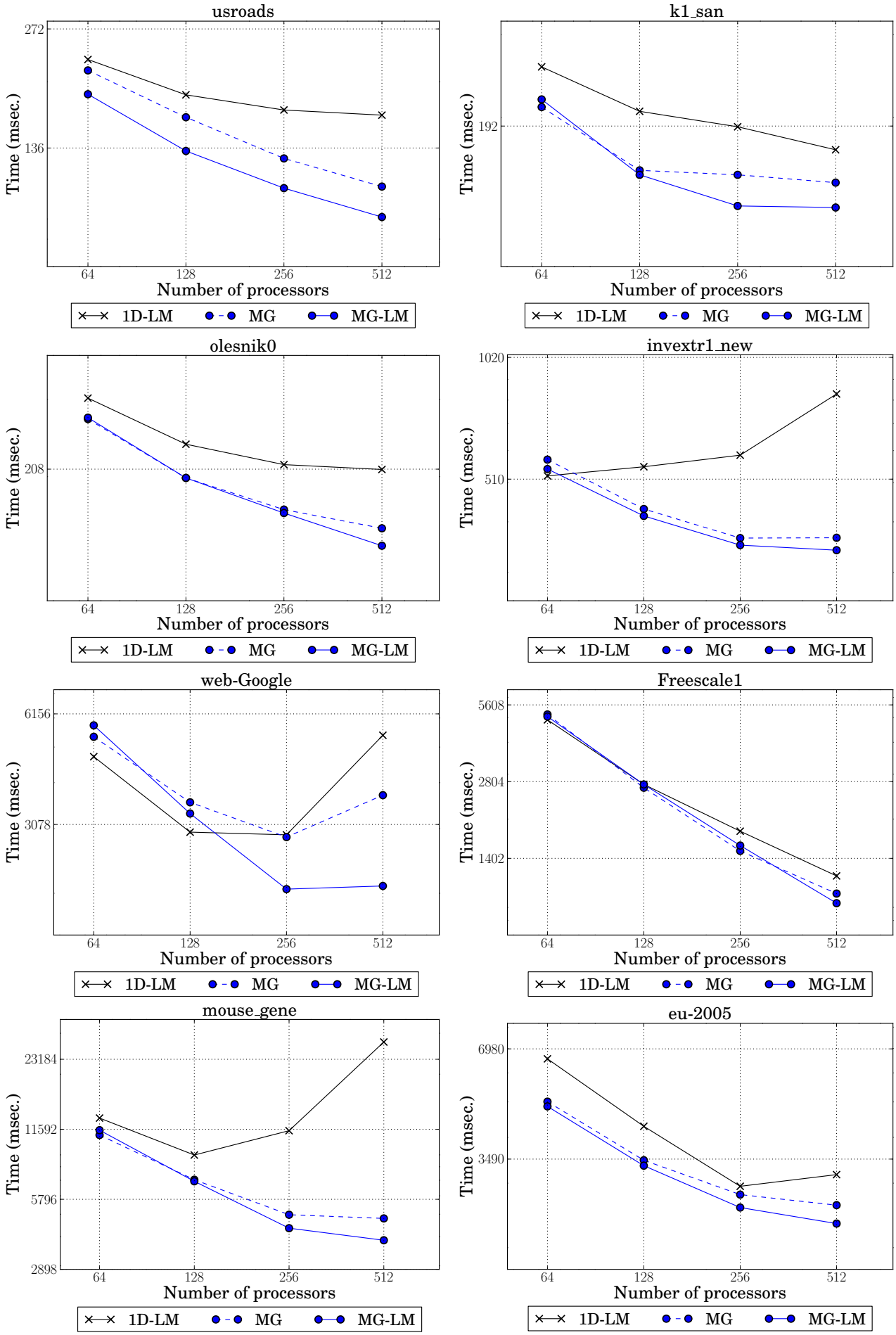


Figure 4.7: Strong scaling analysis of parallel SpMV for `1D-LM`, `MG` and `MG-LM`.

As seen in Figure 4.7, for all eight tested matrices, MG-LM scales better than 1D-LM and MG. Except for Freescale1 and eu-2005, there is a significant/drastic gap between the performances of MG-LM and 1D-LM. For invextr1_new, web-Google, mouse_gene and eu-2005, 1D-LM starts to speed down while MG-LM still speeds up. These strong scaling results indicate that MG-LM finds a better trade-off among the many performance metrics compared to 1D-LM.

4.6 Conclusion

In this work, we proposed two separate hypergraph partitioning models each of which aims at reducing the bandwidth and latency costs of parallel SpMV simultaneously. We utilized recursive bipartitioning paradigm together with message nets to encapsulate this objective. One of the proposed models is based on the fine-grain model, whereas the other one is based on the medium-grain model. The experiments performed in almost a thousand matrices validate the proposed models as they achieved upto 60% of average reduction in total number of messages. To expolit the tradeoff among different performance metrics, we also proposed delayed addition and thresholding for the message nets and tuned the proposed models for the sake of other metrics. The experiments comparing the proposed models against a baseline algorithm [1] showed that the proposed models achieve significant reductions in at least three metrics on average. The runtime results of parallel SpMV verified the validity of the proposed models.

Chapter 5

Improving performance of envelope methods: a top-down profile reduction algorithm

The focus of this work is reducing the profile of a given $m \times m$ sparse matrix $A = (a_{i,j})$ through symmetric row/column permutation. The profile reduction is a crucial factor in the performance of envelope methods widely used in scientific computing community for solving sparse symmetric systems of linear equations. A key parameter that determines the envelope size or profile is the column index $fc(i)$ of the first nonzero entry of row i:

$$fc(i) = min\{j : a_{i,j} \neq 0, 1 \leq j \leq i\}. \tag{5.1}$$

The envelope of A is defined as the set of column indices that lie between the first nonzero entry and the diagonal in each row. That is,

$$envelope(A) = \{(i, j) : fc(i) \leq j < i, 1 \leq i \leq m\}. \tag{5.2}$$

The *profile* of A is the number of elements in the envelope which is equal to the sum of the row widths, i.e.,

$$profile(A) = \sum_{i=1}^{m} row\_width(i) = \sum_{i=1}^{m} (i - fc(i)). \tag{5.3}$$

The envelope methods store $fc(i)$ together with the numeric values (including zeros) between column indices $fc(i)$ and i, for each row i. So, the storage complexity and computational complexity of the envelope methods are respectively proportional to the sum of row widths (profile) and the sum of squares of row widths [15, 16, 17, 18]. Hence, profile reduction is an important combinatorial optimization problem for increasing the performance of envelope methods.

The problem of minimizing the profile of a sparse matrix is known to be NP-hard [57]. So many heuristics are proposed and implemented in the scientific computing community to solve this important combinatorial problem. RCM, which was proposed by George [58], is the earliest commonly-used heuristic and utilizes the level structure of the standard graph representation of a given matrix. Gibbs, Poole and Stockmeyer [59] proposed GPS algorithm, which includes several modifications on RCM in terms of efficiency. Hence, GPS produces comparable results with those of RCM but is many times faster than RCM. Several further improvements of GPS [60, 61, 62, 63] including reductions in both computational and storage complexities have been reported. Sloan [18] proposed a more successful improvement on GPS by utilizing a priority function for nodes during the renumbering step of GPS. Sloan's algorithm was reported to produce smaller profile than previous algorithms and improved further by Duff, Reid and Scott [64].

The spectral algorithm, which was proposed by Barnard, Pothen and Simon [15], adopts a linear algebraic approach utilizing Laplacian matrices rather than a graph-based approach. Kumfert and Pothen [65] proposed a hybrid algorithm in which a global view is obtained by spectral ordering in the first step and fed to Sloan's algorithm for local refinement in the second step. The hybrid algorithm was reported to produce better results than both of the spectral

and Sloan's algorithms alone. Reid and Scott [66] further improved the hybrid algorithm by using binary heap and supervariables in the implementation.

By the motivation of the success achieved by multilevel approaches in graph partitioning [30, 32, 67, 68], two multilevel profile reduction algorithms were proposed by Boman and Hendrickson [16] and Hu and Scott [17]. The former algorithm utilizes the spectral algorithm in reordering the coarsest graph step, whereas the latter one utilizes Sloan in both reordering the coarsest graph and refinement steps.

Hager [69] proposed two heuristics which includes greedy and weighted greedy labeling of rows/columns that try to minimize the number of zeros entering the envelope at each time. In the same work, Hager also proposed two exchange methods for rows/columns in which reduction in profile is calculated for a set of sequence of adjacent row/column exchanges and the one that gives the best reduction is realized each time. Reid and Scott [70] provided an efficient implementation of Hager's exchange methods and reported that applying exchange methods as a post-processing step to multilevel hybrid algorithm [17] produces the best results.

Unlike the traditional profile reduction algorithms, we propose a two-phase approach that adopts a top-down approach in the first phase and a bottom-up approach in the second phase. In the first phase, we recursively bipartition the rows/columns of diagonal blocks until the size of each diagonal block becomes sufficiently small. We utilize the bipartitions on diagonal blocks to induce bipartitions on the respective row/column stripes for obtaining a block structure. In the second phase, we adapt and use Hager's weighted greedy row/column relabeling algorithm [69] to reorder the rows/columns of individual row/column stripes.

For the first phase, we first identify two quality metrics to be maintained during the recursive bipartitioning (RB) process in order to obtain a block structure with a "small" profile. Then, we propose a novel hypergraph model that encapsulates these quality metrics. We present a step-by-step construction of the proposed hypergraph model starting from arow-net hypergraph model.

The construction of the proposed hypergraph model includes fixed vertex, clone-net and external-row net additions, and row-net deletions to/from the primitive row-net hypergraph model of diagonal blocks. In symmetric row/column bipartitioning of a diagonal block, the proposed hypergraph model encodes the first quality metric of minimizing the number of nonzero row-segments in only lower off-diagonal block through vertex fixation and clone net addition schemes, whereas the conventional row-net hypergraph model encodes minimizing the sum of the number of nonzero row-segments in lower and upper off-diagonal blocks. Row-net deletion and external-row net addition schemes are introduced to encode the second quality metric of moving the nonzeros in lower off-diagonal blocks towards diagonal.

The rest of the chapter is organized as follows. Section 5.1 presents the proposed quality metrics and how they are maintained during the RB process. The proposed hypergraph model is discussed in section 5.2. Section 5.3 discusses the method proposed for constructing hypergraphs during the RB process. Experimental results are presented and discussed in section 5.4.

5.1 Maintaining quality metrics during recursive row/column bipartitioning

In the first phase of the proposed approach, we recursively bipartition the rows/columns of matrix A into a block structure form

$$A^{\pi} = PAP^T = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1k} & \cdots & A_{1K} \\ A_{21} & A_{22} & \cdots & A_{2k} & \cdots & A_{2K} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ A_{k1} & A_{k2} & \cdots & A_{kk} & \cdots & A_{kK} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & \cdots & A_{Kk} & \cdots & A_{K,K} \end{pmatrix} = \begin{pmatrix} R_1 \\ R_2 \\ \vdots \\ R_k \\ \vdots \\ R_K \end{pmatrix}$$

$$= \begin{pmatrix} C_1 & C_2 & \cdots & C_k & \cdots & C_K \end{pmatrix} \tag{5.4}$$

until the number of rows/columns in each row/column stripe becomes sufficiently small. This RB process induces a symmetric partial row/column permutation, where P denotes an $m \times m$ permutation matrix to be applied on both rows and columns of A. In symmetrically permuted matrix A^{π}, row i belongs to row stripe R_k if and only if column i belongs to column stripe C_k. The induced row/column permutation is partial since the rows and columns of the row/column stripes can be ordered arbitrarily.

In A^{π}, for $k = 1, \ldots, K$, A_{kk} denotes the kth diagonal block, whereas A_{kh} denotes a lower off-diagonal block for $h < k$ and $A_{k\ell}$ denotes an upper off-diagonal block for $k < \ell$. Let $fcs(i)$ denote the index h of the column stripe C_h in which row i contains its first nonzero $a_{i,j}$. That is,

$$fcs(i) = min\{h : fc(i) \in C_h\}. \tag{5.5}$$

A row i in R_k is said to be a nonzero row if it contains at least one nonzero in lower off-diagonal blocks $\{A_{kh}\}_{h=1}^{k-1}$ of R_k, (i.e., $fcs(i) < k$) or zero row otherwise (i.e., $fcs(i) \geq k$). The nonzeros of a zero-row i are confined to the diagonal block

A_{kk} and upper off-diagonal blocks $\{A_{k\ell}\}_{\ell=k+1}^{K}$ of R_k.

Our algorithm considers the following quality metrics during the RB process to obtain the above-mentioned block structure with a smaller profile:

- keep the number of nonzero rows in lower off-diagonal blocks as small as possible
- keep the nonzeros in lower off-diagonal blocks close to the diagonal as much as possible

In the first level of the RB process, row/column bipartitioning is applied on the original matrix, whereas in the following bipartitioning levels row/column bipartitionings are applied on the diagonal blocks obtained in the previous level in a hierarchical manner. As shown in Figure 5.1, row and column bipartitions of a diagonal block A_{kk} respectively induce a left-right column bipartition $(C_k^{left}, C_k^{right})$ on the columns of C_k and a up-low row bipartition (R_k^{up}, R_k^{low}) on the rows of R_k. Note that the column bipartition of all off-diagonal blocks along C_k conform the column bipartition $(C_k^{left}, C_k^{right})$, whereas the row bipartition of all off-diagonal blocks along R_k conform the row bipartition (R_k^{up}, R_k^{low}). Also note that due to symmetric partitioning, we have $|R_k^{up}| = |C_k^{left}|$ and $|R_k^{low}| = |C_k^{right}|$, where $|\cdot|$ denotes the number of rows/columns in the respective row/column stripe.

As seen in Figure 5.1, the row/column bipartitioning of a diagonal block A_{kk} considers the first quality metric by trying to minimize the number of nonzero rows introduced by the row/column bipartitioning of A_{kk} in the lower off-diagonal block of A_{kk}. In a bipartition of A_{kk}, for a row i that contains its first nonzero in C_k, assigning all columns of C_k that contain a nonzero in row i to C_k^{right} reduces $row\_width(i)$ roughly by $|C_k^{left}|$ on the average. This corresponds to the case where row i, which was a zero row before the bipartitioning, continues to be a zero row after the bipartitioning.

As seen in Figure 5.1, the row/column bipartitioning of an intermediate diagonal block A_{kk} considers the second quality metric in two distinct ways as follows:

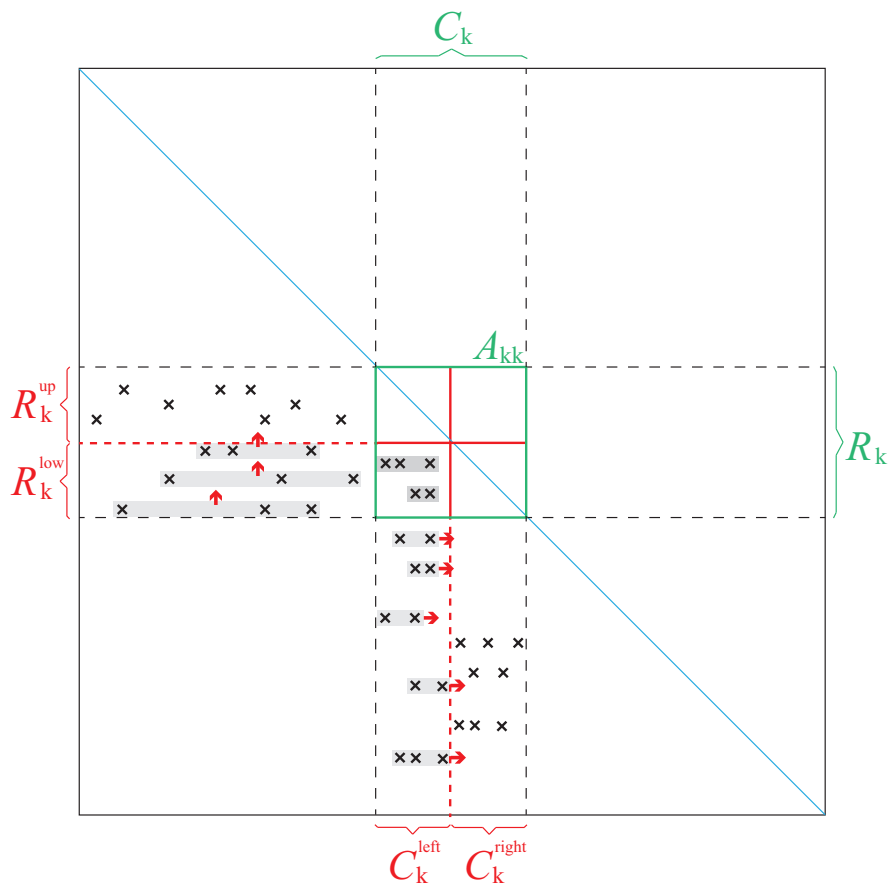


Figure 5.1: A row/column bipartition of A_{kk}

The row bipartitioning tries to move the nonzeros in the lower off-diagonal blocks of R_k upwards through assigning the respective rows to R_k^{up}. In a bipartition of A_{kk}, assigning nonzero-row i to R_k^{up} reduces $row\_width(i)$ roughly by $|R_k^{low}|$ on the average. The column bipartitioning tries to move the nonzeros in the lower off-diagonal blocks of C_k rightwards through assigning the respective columns to C_k^{right}. In a bipartition of A_{kk}, for a nonzero-row i that contains its first nonzero in C_k, assigning all columns of C_k that contain a nonzero in row i to C_k^{right} reduces $row\_width(i)$ roughly by $|C_k^{left}|$ on the average.

All bipartitioning levels–except the first level–of the RB process consider both quality metrics, whereas the first level bipartitioning considers only the first quality metric. For the second quality metric, the bipartitioning of the first diagonal block can only handle moving nonzeros in the lower off-diagonal blocks of the first column slice rightwards, whereas the bipartitioning of the last diagonal block can only handle moving nonzeros in the lower off-diagonal blocks of the last row slice upwards.

5.2 Hypergraph model

Here, we describe our hypergraph model proposed to handle the above-mentioned two quality metrics during the RB process. In an RB step applied for row/column bipartitioning of a diagonal block A_{kk} at a particular level, the proposed hypergraph model is constructed by extending the row-net hypergraph model of A_{kk}.

The row-net hypergraph model [19] of a given matrix $A = (a_{i,j})$ contains one vertex v_i for each column i and one row-net n_j for each row j. Each row-net n_j represents the sparsity pattern of row j by connecting the vertices corresponding to the columns that have a nonzero entry in row j. That is,

$$Pins(n_j) = \{v_i : a_{j,i} \neq 0\} \text{ and } Nets(v_i) = \{n_j : a_{j,i} \neq 0\}. \tag{5.6}$$

In the following two subsections, we describe how we extend the row-net hypergraph model to handle the two quality metrics separately.

5.2.1 Keeping the number of nonzero rows small

Without loss of generality, let A denote a diagonal block subject to row/column bipartitioning at a particular RB step and let $\mathcal{H}(A) = (\mathcal{V}, \mathcal{N})$ denote the row-net hypergraph of A. Consider row/column bipartitioning of A

$$\begin{aligned} A^{\pi} &= PAP^T = \begin{bmatrix} A_{UU} & A_{UL} \\ A_{LU} & A_{LL} \end{bmatrix} = \begin{bmatrix} R^{up} \\ R^{low} \end{bmatrix} \\ &= \begin{bmatrix} C^{left} & C^{right} \end{bmatrix} \end{aligned} \tag{5.7}$$

Here A_{UU} and A_{LL} respectively denote upper and lower diagonal blocks, whereas A_{LU} and A_{UL} respectively denote lower off-diagonal and upper off-diagonal blocks obtained after the bipartitioning. In a similar manner, R^{up} and R^{low} denote upper and lower row stripes, whereas C^{left} and C^{right} denote left and right column stripes.

As described in [19], the cutsize of bipartition of $\mathcal{H}(A)$ encodes the sum of the number of nonzero row segments in lower and upper off-diagonal blocks A_{LU} and A_{UL}. However, the first quality metric refers to minimizing the number of nonzero row segments in only lower off-diagonal block A_{LU}. In order to enable encoding this asymmetric quality metric through symmetric row/column bipartitioning of A, we extend the row-net hypergraph model $\mathcal{H}(A)$ to $\mathcal{H}_e(A) = (\mathcal{V}_e, \mathcal{N}_e)$ by adding two vertices and $|\mathcal{N}|$ nets as follows.

We introduce two new vertices v_U and v_L that are respectively fixed to the upper and lower parts $\mathcal{V}_U$ and $\mathcal{V}_L$ of a bipartition $\Pi = \{\mathcal{V}_U, \mathcal{V}_L\}$. We also introduce

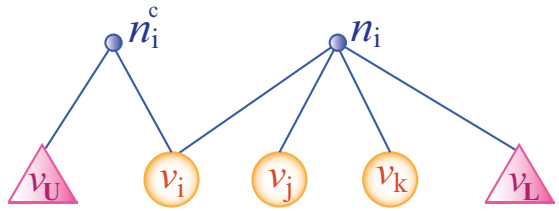


Figure 5.2: Row-net n_i and its clone net n_i^c

a new clone net n_i^c for each row-net n_i. That is,

$$\mathcal{V}_e \leftarrow \mathcal{V} \cup \{v_U, v_L\}, \tag{5.8}$$

$$\mathcal{N}_e \leftarrow \mathcal{N} \cup \{n_i^c : n_i \in \mathcal{N}\}. \tag{5.9}$$

The clone net n_i^c is defined as connecting vertices v_i and v_U, i.e.,

$$Pins(n_i^c) \leftarrow \{v_i, v_U\}. \tag{5.10}$$

We enforce each row-net n_i to connect the fixed vertex v_L, i.e.,

$$Pins(n_i) \leftarrow Pins(n_i) \cup \{v_L\}. \tag{5.11}$$

Figure 5.2 displays a visualization of row-net n_i and its clone net n_i^c for a row i that contains three nonzeros in columns i, j, and k.

For an $m \times m$ matrix A with $nnz(A)$ nonzeros, $\mathcal{H}_e(A)$ contains two fixed vertices, m free vertices, $2 \times m$ nets and $3 \times m + nnz(A)$ pins. Figure 5.4(a) displays the extended row-net hypergraph $\mathcal{H}_e(A_{33})$ of diagonal block A_{33} which contains 7 rows/columns and 15 nonzeros. As seen in Figure 5.4(a), $\mathcal{H}_e(A_{33})$ contains 2 fixed vertices, 7 free vertices, $2 \times 7 = 14$ nets and $3 \times 7 + 15 = 36$ pins.

A vertex bipartition $\Pi = \{\mathcal{V}_U, \mathcal{V}_L\}$, where $v_U \in \mathcal{V}_U$ and $v_L \in \mathcal{V}_L$, is decoded as inducing a row/column bipartition of A as follows. $v_i \in \mathcal{V}_U$ is decoded as assigning row i and column i respectively to row slice R^{up} and column slice C^{left}, whereas $v_i \in \mathcal{V}_L$ is decoded as assigning row i and column i respectively to row slice R^{low} and column slice C^{right}.

Here we will discuss how minimizing the cutsize of a bipartition of $\mathcal{H}_e(A)$ is equivalent to minimizing the number of nonzero rows in the lower off-diagonal block A_{LU}. The two fixed vertices v_U and v_L serve as anchors to the nets as follows for any bipartition $\Pi(\mathcal{H}_e) = (\mathcal{V}_U, \mathcal{V}_L)$. v_U anchors each clone net n_i^c to the upper part $\mathcal{V}_U$, that is $\mathcal{V}_U \in \Lambda(n_i^c)$. v_L anchors each row-net n_i to the lower part $\mathcal{V}_L$, that is $\mathcal{V}_L \in \Lambda(n_i)$. Since $\mathcal{V}_U \in \Lambda(n_i^c)$, $\mathcal{V}_L \in \Lambda(n_i)$ and nets n_i and n_i^c share vertex v_i, either one of them or both of them are in the cut of Π for each row/column i. Below, we investigate these three cut/uncut states for each net pair (n_i, n_i^c) and display their visualizations in the left side of Figure 5.3.

State 1: *n_i is cut and n_i^c is uncut.* This state occurs when $v_i \in \mathcal{V}_U$. Since row i is assigned to R^{up}, it does not incur any nonzero-row segments in A_{LU}.

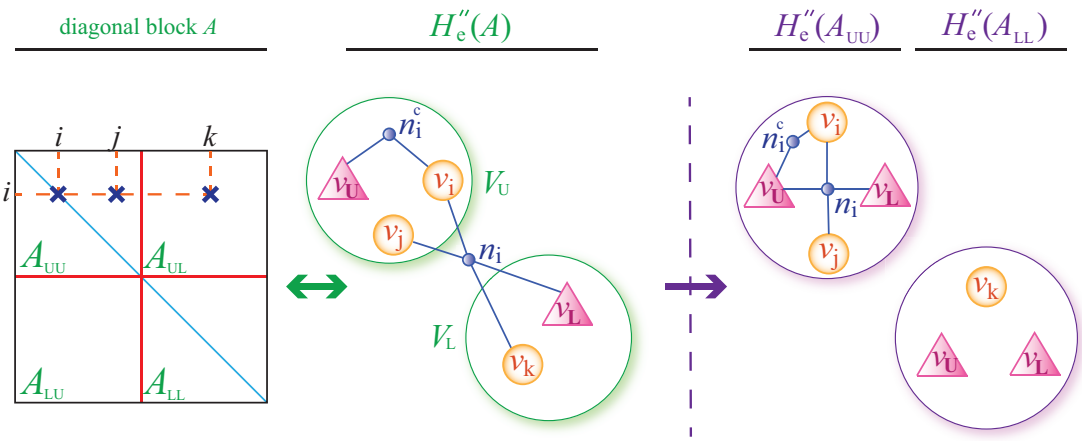
State 2: *n_i is uncut and n_i^c is cut.* This state occurs when all pins of n_i including v_i are assigned to $\mathcal{V}_L$. Although row i is assigned to R^{low}, it does not incur any nonzero-row segments to A_{LU} since all nonzeros of row i are confined to A_{LL}.

State 3: *n_i and n_i^c are both cut.* This state occurs when $v_i \in \mathcal{V}_L$ and n_i has at least one pin in $\mathcal{V}_U$. Since row i is assigned to R^{low} and n_i has at least one pin in $\mathcal{V}_U$, row i has at least one nonzero in A_{LU} and hence it incurs one nonzero-row segment in A_{LU}.

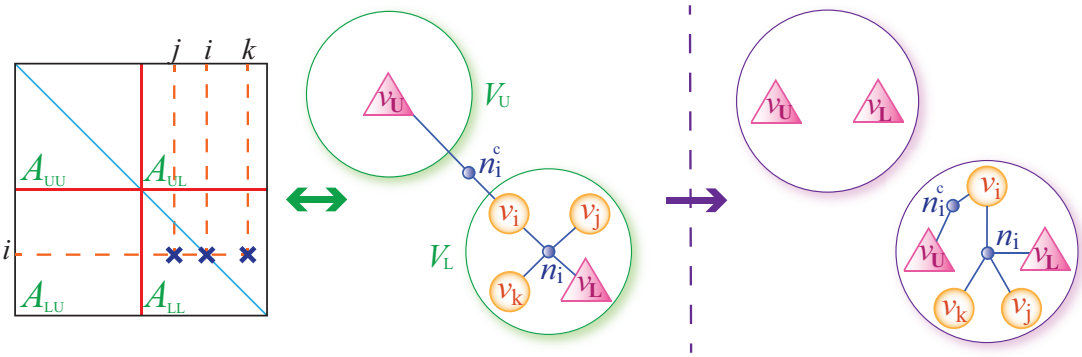
In summary, in the first two states, net pair (n_i, n_i^c) incurs a total cost of 1 (in terms of cutsize) while introducing no nonzero rows to A_{LU}. In third state, net pair (n_i, n_i^c) incurs a total cost of 2 while introducing 1 nonzero row to A_{LU}. So we have

$$\text{number of nonzero rows in } A_{LU} \ = cutsize(\Pi(\mathcal{H}_e)) - |\mathcal{N}|. \tag{5.12}$$

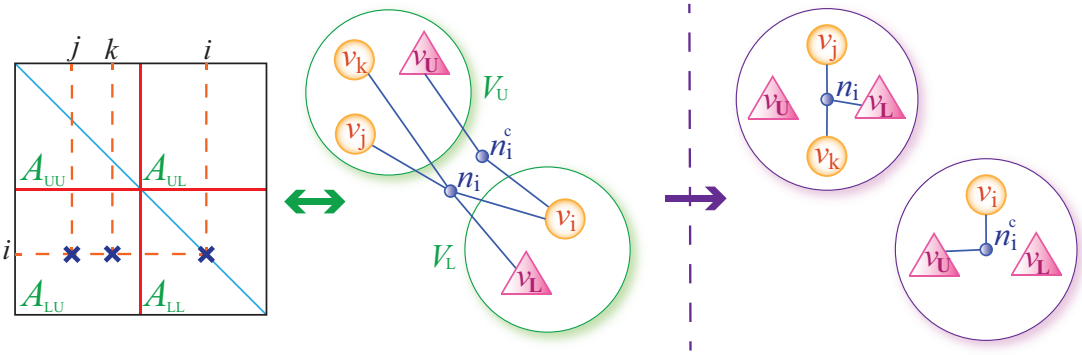
Since $|\mathcal{N}|$ (i.e., number of rows/columns) is constant, the bipartitioning objective of minimizing the number of cut-nets corresponds to minimizing number of nonzero rows in A_{LU}.



(a) State 1: n_i is cut and n_i^c is uncut.

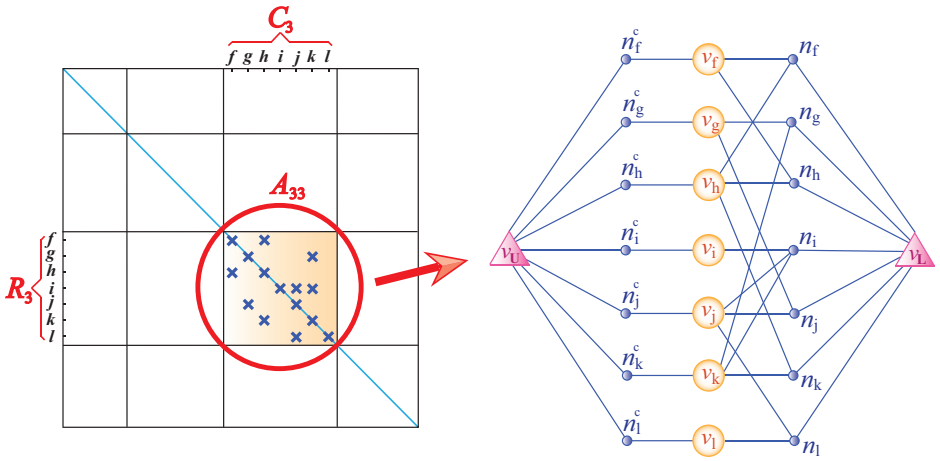


(b) State 2: n_i is uncut and n_i^c is cut.

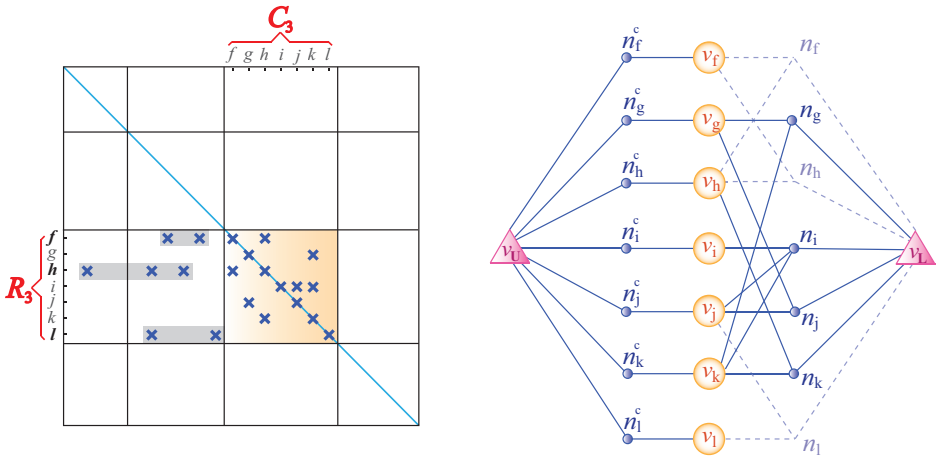


(c) State 3: n_i and n_i^c are both cut.

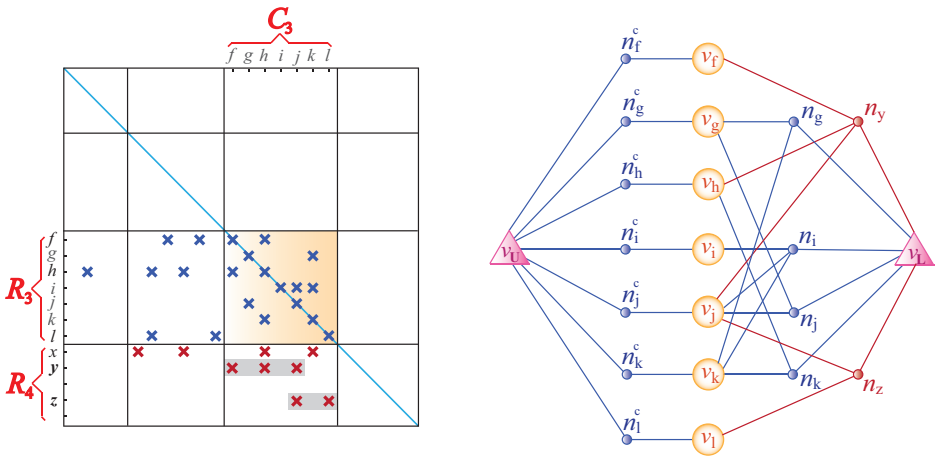
Figure 5.3: Left side: three cut/uncut states for net pair (n_i, n_i^c). Right side: construction of $\mathcal{H}_e''(A_{UU})$ and $\mathcal{H}_e''(A_{LL})$ from $\mathcal{H}_e''(A)$ for the case where $\mathcal{H}_e''(A)$ contains both n_i and n_i^c.



(a) Construction of $\mathcal{H}_e(A_{33})$ from row-net hypergraph model $\mathcal{H}(A_{33})$



(b) Construction of $\mathcal{H}'_e(A_{33})$ from $\mathcal{H}_e(A_{33})$ by row-net deletion



(c) Construction of $\mathcal{H}''_e(A_{33})$ from $\mathcal{H}'_e(A_{33})$ by row-net addition

Figure 5.4: Step-by-step construction of the proposed hypergraph model

5.2.2 Keeping nonzeros close to the diagonal

Without loss of generality, let A_{kk} denote a diagonal block subject to row/column bipartitioning at a particular RB step and let $\mathcal{H}_e(A_{kk}) = (\mathcal{V}_e, \mathcal{N}_e)$ denote the extended row-net hypergraph of A_{kk} which is constructed as described in Section 5.2.1. As mentioned earlier, we utilize the column bipartitioning of A_{kk} to move the nonzeros in lower off-diagonal blocks of C_k rightwards. In a dual manner, we utilize the row bipartitioning of A_{kk} to move the nonzero rows in lower off-diagonal blocks of R_k upwards.

5.2.2.1 Moving nonzeros in lower off-diagonal blocks in R_k up towards R_{k-1}: Constructing $\mathcal{H}'_e(A_{kk})$ from $\mathcal{H}_e(A_{kk})$

Consider a row i in R_k that contains at least one nonzero in the lower off-diagonal blocks (A_{kh} for $1 \leq h < k$) of R_k, i.e., $fcs(i) < k$. So, row i is already a nonzero row of R_k independent of the row/column bipartitioning of A_{kk}. In other words, making row i a zero row of A_{kk} through removing row-net n_i from the cut (making n_i uncut as seen in State 2 of Figure 5.3) does not make row i a zero row of R_k. That is, row-net n_i does not serve the purpose of satisfying the first quality metric for row i. Thus, we delete net n_i from $\mathcal{H}_e(A_{kk})$ while keeping n_i^c in the construction of $\mathcal{H}'_e(A_{kk})$. We keep n_i^c in $\mathcal{H}'_e(A_{kk})$, because making net n_i^c uncut will enforce v_i to be assigned to the upper part $\mathcal{V}_U$ as seen in State 1 of Figure 5.3. Assigning v_i to $\mathcal{V}_U$ corresponds to assigning row i to the upper row slice R_k^{up} of the row/column bipartition of A_{kk}. Assigning row i to R_k^{up} serves the purpose of satisfying the second quality metric of moving the nonzeros of row i in lower off-diagonal blocks in R_k upwards. In other words, making n_i^c uncut reduces $row\_width(i)$ roughly by $|R_k^{low}|$ on the average.

The above-mentioned row-net deletion process can be expressed as follows:

$$\mathcal{N}'_e = \mathcal{N}_e - \{n_i : fcs(i) < k\}. \tag{5.13}$$

Figure 5.4(b) shows the row-net deletion process for constructing $\mathcal{H}'_e(A_{33})$ from the extended hypergraph model $\mathcal{H}_e(A_{33})$ given in Figure 5.4(a). As seen in Figure 5.4(b), there exists 3 nonzero-rows f, h and l in R_3 with $fcs(f) = 2$, $fcs(h) = 1$ and $fcs(l) = 2$. So row-nets n_f, n_h and n_l are deleted from $\mathcal{H}_e(A_{33})$ together with their pins, whereas nets n_f^c, n_h^c and n_l^c remain in $\mathcal{H}'_e(A_{33})$.

5.2.2.2 Moving nonzeros in lower off-diagonal blocks in C_k right towards C_{k+1}: Constructing $\mathcal{H}''_e(A_{kk})$ from $\mathcal{H}'_e(A_{kk})$

Consider a nonzero-row x in R_ℓ with $\ell > k$ and $fcs(x) = k$, just before the current RB step. We encapsulate moving the nonzeros of row x rightwards by adding a new net n_x that represents the sparsity pattern of row x in $A_{\ell k}$ as an external-row net to $\mathcal{H}'_e(A_{kk})$. Row-net n_x connects the vertices corresponding to the columns that have a nonzero entry in row x in $A_{\ell k}$ as well as the fixed vertex v_L. Since row-net n_x is anchored to $\mathcal{V}_L$, n_x can be removed from the cut only if all of its pins are assigned to $\mathcal{V}_L$. This corresponds to moving all nonzeros of row x to the right column slice C_k^{right} induced by the column bipartition of A_{kk}. In other words, making n_x uncut reduces $row\_width(x)$ roughly by $|C_k^{left}|$ on the average. Note that if nonzero-row x contains at least one nonzero in lower off-diagonal blocks that lie left to $A_{\ell k}$ (i.e., $fcs(x) < k$), we do not add an external-row net n_x to $\mathcal{H}'_e(A_{kk})$. This is because moving all nonzeros of row x in C_k to C_k^{right} does not have any potential to reduce $row\_width(x)$, in this case.

The above-mentioned external-row net addition process can be expressed as follows:

$$\begin{aligned} \mathcal{N}''_e &= \mathcal{N}'_e \cup \{n_x : fcs(x) = k\}, \text{ where} \\ Pins(n_x) &= \{v_L\} \cup \{v_j : a_{x,j} \neq 0 \text{ and } j \in C_k\}. \end{aligned} \tag{5.14}$$

Figure 5.4(c) shows the external-row net addition process for constructing $\mathcal{H}''_e(A_{33})$ from $\mathcal{H}'_e(A_{33})$ for the sample matrix A given in Figure 5.4(a). As seen in Figure 5.4 (c), there exists 3 nonzero-rows x, y and z in lower off-diagonal block(s) of C_3 with $fcs(x) = 2$, $fcs(y) = 3$ and $fcs(z) = 3$. Since nonzero-rows y and z contain their first nonzero in C_3, we add row-nets n_y and n_z with

$Pins(n_y) = \{v_f, v_h, v_j\}$ and $Pins(n_z) = \{v_j, v_l\}$ to $\mathcal{H}'_e(A_{33})$ as external nets. Although row x is also a nonzero-row in C_3, row x does not incur the addition of a new row-net to $\mathcal{H}'_e(A_{33})$ as an external-row net since row x contains its first nonzero in C_2. However, row x as well as rows f and l incur the addition of external-row nets to $\mathcal{H}''(A_{22})$, since all of the rows x, f and l have their first nonzero in C_2.

We should mention here the close relation between the row-net deletion and the external-row net addition processes mentioned above. Consider a nonzero-row i in R_ℓ with $fcs(i) = k$. Nonzero-row i incurs the deletion of a row-net n_i with $Pins(n_i) = \{v_L\} \cup \{v_j : a_{i,j} \neq 0 \text{ and } j \in C_\ell\}$ from $\mathcal{H}_e(A_{\ell\ell})$, whereas it incurs the addition of external-row net n'_i with $Pins(n'_i) = \{v_L\} \cup \{v_j : a_{i,j} \neq 0 \text{ and } j \in C_k\}$ to $\mathcal{H}'_e(A_{kk})$. For example, in Figure 5.4, consider nonzero-row h in R_3 with $fcs(h) = 1$. Nonzero-row h incurs the deletion of a row-net from $\mathcal{H}_e(A_{33})$, whereas it incurs the addition of a new net to $\mathcal{H}'_e(A_{11})$ as an external-row net.

5.3 Hypergraph construction during recursive bipartitioning

In Section 5.2, the extended row-net hypergraph model, row-net deletion and external-row net addition operations required to construct $\mathcal{H}''_e(A_{kk})$ for a diagonal block A_{kk} are discussed in terms of the sparsity pattern of A_{kk} as well as the sparsity patterns of lower off-diagonal blocks of row and column stripes R_k and C_k. Although this scheme is utilized for the sake of clarity of the presentation, it does not lead to an efficient implementation scheme since it requires multiple passes over the nonzeros of the original matrix at each RB level. Here, we propose and discuss an efficient hypergraph construction scheme, which requires only one pass over the nonzeros of the original matrix, during each RB level.

In the proposed method, only in the first RB level, the hypergraph model is constructed from scratch by using the extended row-net hypergraph model that

utilizes the sparsity pattern of the original matrix. In the successive RB levels, hypergraph model of each diagonal block is constructed from the hypergraph model of its parent diagonal block. In other words, hypergraphs $\mathcal{H}''_e(A_{UU})$ and $\mathcal{H}''_e(A_{LL})$ are constructed for the upper and lower diagonal blocks A_{UU} and A_{LL} that are induced by the bipartition $\Pi(\mathcal{H}''_e(A)) = \{\mathcal{V}_U, \mathcal{V}_L\}$ of the hypergraph $\mathcal{H}''_e(A)$ of their parent diagonal block A (see (5.7)). Without loss of generality, let A represent the diagonal block A_{kk} at the row and column stripes R_k and C_k, just before the current RB step. For the sake of simplicity, we will use $\mathcal{H}_A$, $\mathcal{H}_U$ and $\mathcal{H}_L$ to refer to $\mathcal{H}''_e(A)$, $\mathcal{H}''_e(A_{UU})$ and $\mathcal{H}''_e(A_{LL})$, respectively.

We identify three cases depending on the existence of either one of n_i and n_i^c or both of them in $\mathcal{H}_A$. In all cases, we will have

$$\mathcal{H}_U = (\mathcal{V}_U \cup \{v_L\}, \mathcal{N}_U) \text{ and } \mathcal{H}_L = (\mathcal{V}_L \cup \{v_U\}, \mathcal{N}_L). \tag{5.15}$$

Below, we discuss the construction of the net sets $\mathcal{N}_U$ and $\mathcal{N}_L$ in an incremental manner from the net set $\mathcal{N}_A$ of $\mathcal{H}_A$ for each different case.

Case 1: $\mathcal{H}_A$ contains both n_i and n_i^c. This case occurs when row i contains at least one off-diagonal nonzero in A and no nonzeros in lower off-diagonal blocks of R_k. The latter requirement refers to the fact that row i is a zero row in the block structure just before the current RB step. In other words, the net deletion process mentioned earlier does not apply to row-net n_i.

There exists a different net construction scheme for each different cut/uncut states for net pair (n_i, n_i^c) as shown in Figure 5.3.

State 1: *n_i is cut and n_i^c is uncut.* Row i is assigned to R^{up}. So, row i incurs a net pair (n'_i, n_i^c) in $\mathcal{N}_U$, where pins of n'_i correspond to the nonzeros of row i that are confined to A_{UU}. That is,

$$\mathcal{N}_U = \mathcal{N}_U \cup \{n_i^c\} \cup \{n'_i : Pins(n'_i) = (Pins(n_i) \cap \mathcal{V}_U) \cup \{v_L\}\}. \tag{5.16}$$

Row i does not incur any nets in $\mathcal{N}_L$.

State 2: *n_i is uncut and n_i^c is cut.* Row i is assigned to R^{low} in such a way that all of its nonzeros are confined to A_{LL}, row i incurs a net pair (n_i', n_i^c) in $\mathcal{N}_L$, where pins of n_i' are exactly equal to those of n_i. That is,

$$\mathcal{N}_L = \mathcal{N}_L \cup \{n_i^c\} \cup \{n_i' : Pins(n_i') = Pins(n_i)\}. \tag{5.17}$$

Row i does not incur any nets in $\mathcal{N}_U$.

State 3: *n_i and n_i^c are both cut.* Row i is assigned to R^{low} in such a way that its nonzeros are scattered between A_{LU} and A_{LL}. So, row i incurs net n_i^c in $\mathcal{N}_L$ and n_i' in $\mathcal{N}_U$, where pins of n_i' correspond to the nonzeros of row i that are confined to A_{UU}. That is,

$$\mathcal{N}_L = \mathcal{N}_L \cup \{n_i^c\}, \tag{5.18}$$

$$\mathcal{N}_U = \mathcal{N}_U \cup \{n_i' : Pins(n_i') = (Pins(n_i) \cap \mathcal{V}_U) \cup \{v_L\}\}. \tag{5.19}$$

In summary, for all three states, net n_i^c with $Pins(n_i^c) = \{v_i, v_U\}$ become a net of the hypergraph that represents the part to which vertex v_i is assigned in Π. Note that row i, which was a zero row before the bipartitioning, continues to be a zero row after the bipartitioning in the first two states, whereas it becomes a nonzero row after the bipartitioning in the third state. So, in the first two states, net n_i' derived from n_i also becomes a net of the hypergraph that represents the part to which vertex v_i is assigned, whereas in the third state n_i' becomes a net of the hypergraph that represents the other part of the bipartition. Hence, having n_i and n_i^c in the same hypergraph in the first two states serves the first quality metric, whereas having n_i and n_i^c in two different hypergraphs in the third state serves the second quality metric. In the third state, instead of first including net pair (n_i, n_i^c) in $\mathcal{H}_L$ then deleting row-net n_i from $\mathcal{H}_L$ and then adding external-row net n_i' in $\mathcal{H}_U$ as described in Section 5.2, we simply include n_i' and n_i^c in $\mathcal{H}_U$ and $\mathcal{H}_L$, respectively.

Case 2: $\mathcal{H}_A$ contains n_i^c but not n_i. This case occurs when row i contains

at least one nonzero in lower off-diagonal blocks of R_k, i.e., row i is a nonzero row in the block structure just before the current RB step. In other words, $\mathcal{H}_A$ does not contain row-net n_i due to the net deletion process mentioned earlier. There are two cut/uncut states for n_i^c as shown in Figure 5.5.

State 1: *n_i^c is uncut.* Row i is assigned to R^{up} . So, row i incurs n_i^c in $\mathcal{N}_U$. That is,

$$\mathcal{N}_U = \mathcal{N}_U \cup \{n_i^c\}. \tag{5.20}$$

State 2: *n_i^c is cut.* Row i is assigned to R^{low} . So, row i incurs n_i^c in $\mathcal{N}_L$. That is,

$$\mathcal{N}_L = \mathcal{N}_L \cup \{n_i^c\}. \tag{5.21}$$

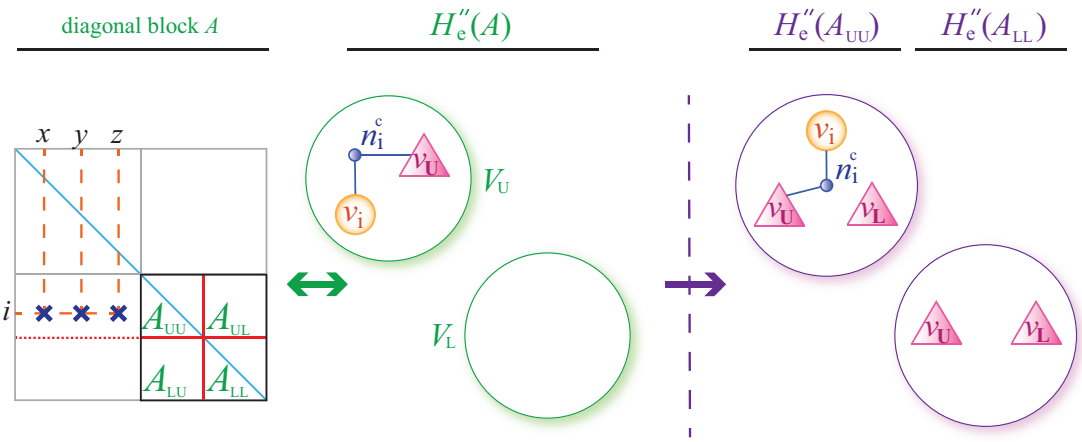
In summary, for both states, net n_i^c with $Pins(n_i^c) = \{v_i, v_U\}$ become a net of the hypergraph that represents the part to which vertex v_i is assigned in Π. Note that in both states of n_i^c, instead of first including net pair (n_i, n_i^c) in the hypergraph that represents the part to which vertex v_i is assigned and then removing row-net n_i from that hypergraph as described in Section 5.2, we simply include only n_i^c, since having n_i^c but not n_i in $\mathcal{H}_A$ infers that row i is already a nonzero row.

Case 3: $\mathcal{H}_A$ contains n_i but not n_i^c. This case occurs when row i contains its first nonzero in one of the lower off-diagonal blocks of C_k, i.e., row i is a nonzero row in the block structure just before the current RB step. In other words, $\mathcal{H}_A$ contains external-row net n_i due to the net addition process mentioned earlier. There are two cut/uncut states for n_i as shown in Figure 5.6.

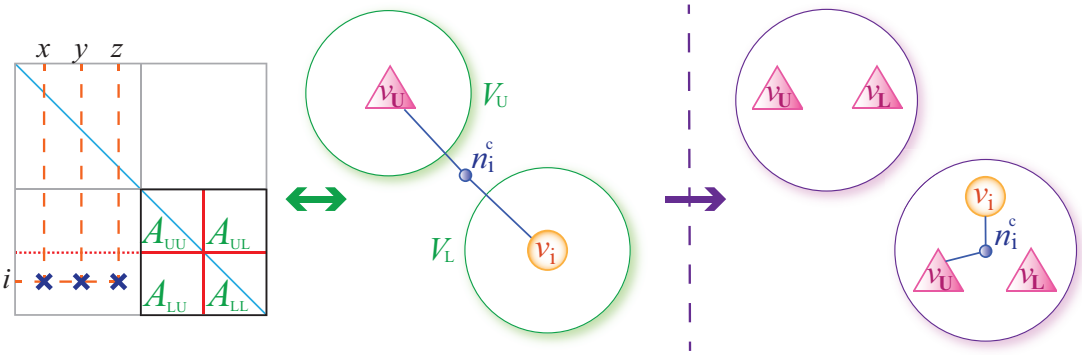
State 1: *n_i is uncut.* All nonzeros of row i are confined to C^{right}. So, row i incurs n_i' in $\mathcal{N}_L$ as

$$\mathcal{N}_L = \mathcal{N}_L \cup \{n_i' : Pins(n_i') = Pins(n_i)\}. \tag{5.22}$$

State 2: *n_i is cut.* Nonzeros of row i are scattered between C^{left} and C^{right}, i.e.,

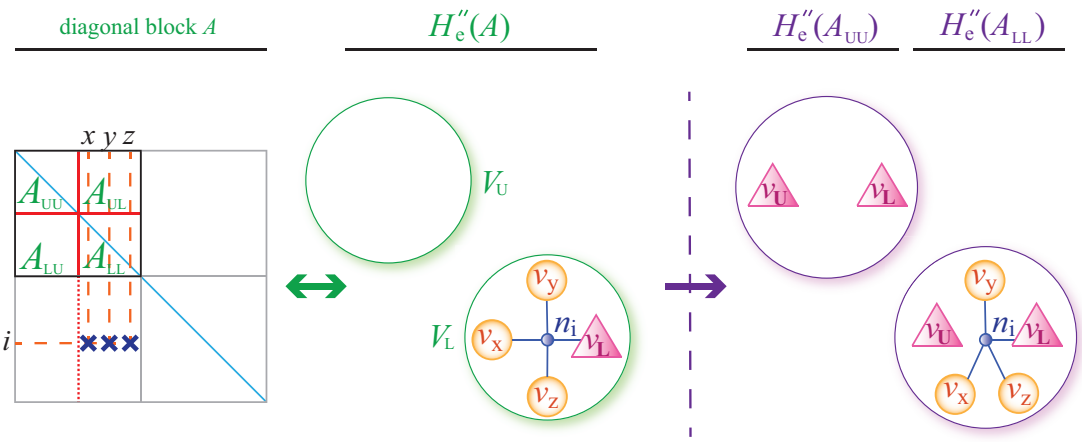


(a) State 1: n_i^c is uncut

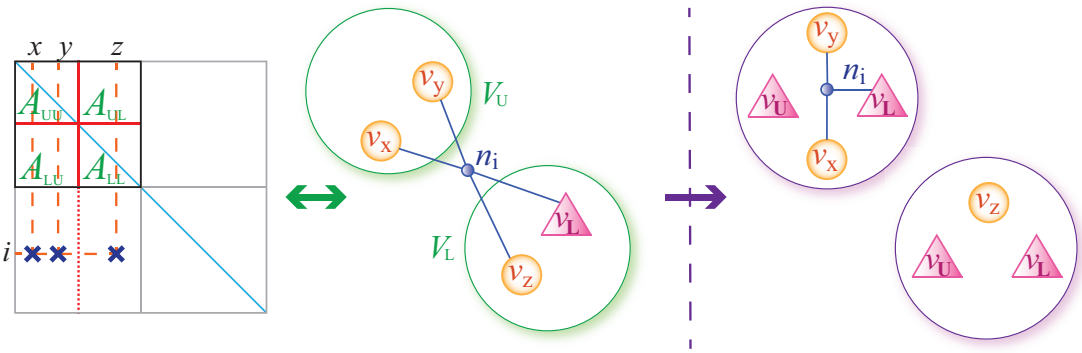


(b) State 2: n_i^c is cut

Figure 5.5: Left side: two cut/uncut states for net n_i^c of nonzero-row i. Right side: construction of $\mathcal{H}_U$ and $\mathcal{H}_L$ from $\mathcal{H}_A$ for the case where $\mathcal{H}_A$ contains n_i^c but not n_i.



(a) State 1: n_i is uncut



(b) State 2: n_i is cut

Figure 5.6: Left side: two cut/uncut states for external-row net n_i of nonzero-row i. Right side: construction of $\mathcal{H}_U$ and $\mathcal{H}_L$ from $\mathcal{H}_A$ for the case where $\mathcal{H}_A$ contains n_i but not n_i^c.

row i contains its first nonzero in C^{left}. So, row i incurs n_i' in $\mathcal{N}_U$ as

$$\mathcal{N}_U = \mathcal{N}_U \cup \{n_i' : Pins(n_i') = (Pins(n_i) \cap \mathcal{V}_U) \cup \{v_L\}\}. \tag{5.23}$$

5.4 Experiments

5.4.1 Datasets

We have tested the performance of the proposed profile reduction algorithm on three datasets, each of which consists of symmetric matrices obtained from the SuiteSparse Matrix Collection [47]. These datasets are derived from the ones that were previously used in the evaluations of the well-known successful profile reduction algorithms.

Dataset 1 is obtained by merging the 18 matrices in Kumfert and Pothen's Collection, which was used in [15, 16, 66, 65, 17, 70] and the 8 matrices in NASA Collection, which was used in [15, 69]. The resulting dataset contains 23 matrices from a variety of application areas: structural analysis, fluid dynamics, stochastic optimization and multicommodity flows.

Dataset 2 is derived from the Netlib Linear Programming Problem Collection, which was used in [69]. We first exclude the matrices with less than 1000 rows/columns or more than 100000 rows/columns. Then, we multiply each of the remaining 44 matrices with its transpose and use the resulting AA^T matrix as a test matrix.

Dataset 3 is obtained from the Harwell-Boeing Collection, which was used in [15, 69, 17]. We exclude nonsymmetric matrices and the ones with less than 1000 rows/columns. The resulting dataset contains 71 matrices, which all arise in structural engineering problems.

Tables 5.1, 5.2 and 5.3 display the properties of the matrices in Datasets 1, 2, and 3, respectively. Columns "name", "n" and "nnz" respectively refer to the name, the number of rows/columns and the number of nonzeros of the respective matrix.

Table 5.1: Performance comparison on Dataset 1

matrix properties			normalized profile				
name	n	nnz	orig.	GK+H	S+H	HS+H	HP
nasa1824	1824	39208	112.7	106.8	102.2	103.5	**81.4**
nasa2146	2146	72250	76.7	75.6	**73.2**	75.0	79.0
nasa2910	2910	174296	180.7	172.9	156.0	154.0	**149.8**
nasa4704	4704	104756	195.1	175.8	171.5	**146.7**	151.0
barth4	6019	40965	358.7	59.6	50.2	46.2	**40.2**
barth	6691	46187	2369.6	72.9	66.7	56.9	**54.1**
commanche_dual	7920	31680	2088.8	42.5	38.3	36.9	**31.9**
shuttle_eddy	10429	103599	1117.7	64.4	58.2	58.4	**53.0**
skirt	12598	196520	290.6	73.6	53.3	60.9	**52.1**
barth5	15606	107362	260.0	109.1	83.9	75.5	**70.5**
pds10	16558	149658	1089.3	900.3	534.0	559.0	**398.5**
copter1	17222	211064	1102.3	337.5	**330.3**	332.5	353.5
tandem_vtx	18454	253350	4389.7	393.3	308.8	268.7	**248.4**
ford1	18728	101576	1879.4	100.5	98.6	80.0	**76.5**
bcsstk30	28924	2043492	542.4	533.2	552.2	294.3	**288.2**
pwt	36519	326107	2416.4	140.8	133.7	132.6	**117.3**
finance256	37376	298496	6458.7	723.6	166.4	175.9	**114.7**
nasasrb	54870	2677324	370.2	346.4	341.8	323.7	**307.5**
copter2	55476	759952	19540.8	803.3	649.9	**557.9**	603.9
finance512	74752	596992	12858.2	563.5	155.9	**136.1**	141.4
onera_dual	85567	419201	8286.8	1003.1	932.7	**509.8**	513.2
tandem_dual	94069	460493	5182.0	757.3	625.2	430.4	**401.9**
ford2	100196	544688	3714.6	358.4	333.3	248.7	**231.2**
average profile				218.8	172.2	152.7	**140.6**
number of smallest profiles				0	2	4	**17**

Table 5.2: Performance comparison on Dataset 2

matrix properties			normalized profile				
name	n	nnz	orig.	GK+H	S+H	HS+H	HP
lp_truss	1000	26120	272.1	**44.3**	44.5	46.5	44.7
lp_sctap2	1090	12100	103.0	56.1	56.7	57.9	**41.8**
lp_woodw	1098	41940	182.5	125.8	122.2	122.9	**80.4**
lp_osa_07	1118	106050	78.4	**54.2**	55.0	61.0	77.0
lp_ship12l	1151	22388	462.1	31.5	17.4	**16.7**	19.5
lp_ship12s	1151	11732	461.4	19.8	13.7	**13.1**	14.1
lp_sierra	1227	5819	309.4	6.6	6.2	**6.0**	6.3
lp_ganges	1309	16621	331.5	32.5	86.9	35.0	**29.9**
lp_pilot	1441	124461	494.5	233.8	221.1	232.0	**192.3**
lp_sctap3	1480	16252	142.1	76.9	77.8	54.4	**41.6**
lp_degen3	1503	101775	670.0	239.7	179.0	183.6	**157.0**
lp_cycle	1903	57318	137.1	70.2	215.9	78.1	**58.2**
lp_pilot87	2030	238588	573.3	363.0	310.7	**279.9**	303.1
lp_stocfor2	2157	27633	328.5	77.5	28.8	29.4	**26.7**
lp_d2q06c	2171	56131	514.0	200.0	240.8	112.1	**104.7**
lp_80bau3b	2262	22410	551.8	109.4	85.0	**68.1**	80.1
lp_bnl2	2324	29224	194.3	103.9	115.8	92.7	**83.1**
lp_osa_14	2337	230023	80.3	**55.9**	56.9	61.9	92.2
lp_greenbea	2392	70061	405.5	190.8	145.9	109.5	**92.3**
lp_greenbeb	2392	70061	405.5	190.8	145.9	109.5	**92.3**
lpi_greenbea	2393	70074	405.9	189.0	146.5	111.1	**95.4**
lp_ken_07	2426	14382	35.1	28.2	158.2	27.8	**27.4**
lpi_gran	2658	197312	287.6	147.0	137.2	157.5	**111.9**
lpi_bgindy	2671	126747	446.6	**164.1**	417.7	338.0	187.6
lp_pds_02	2953	23281	208.7	142.9	138.7	102.2	**64.5**
lpi_cplex1	3005	2265521	1000.5	**626.2**	**626.2**	**626.2**	703.3
lp_cre_c	3068	40776	1316.2	151.6	128.5	118.8	**55.8**
lp_maros_r7	3136	664080	272.8	**271.3**	**271.3**	**271.3**	400.4
lp_qap12	3192	152376	1590.0	1057.7	1052.1	1014.8	**926.9**
lp_cre_a	3516	44866	1549.3	160.2	147.3	89.5	**57.4**
lpi_ceria3d	3576	1963306	1774.3	989.4	**982.9**	984.3	1034.8
lpi_gosh	3792	206008	1506.1	506.8	332.3	235.0	**182.2**
lp_osa_30	4350	436738	81.1	**56.6**	57.7	62.7	71.6
lp_dfl001	6071	81917	2474.7	749.2	619.4	554.0	**534.0**
lp_qap15	6330	378480	3157.5	2099.1	2121.8	1937.6	**1873.4**
lp_cre_d	8926	369786	2587.0	1170.8	431.4	438.8	**266.4**
lp_cre_b	9648	396310	2928.6	1234.5	480.2	394.0	**292.9**
lp_pds_06	9881	88003	677.9	547.1	346.2	340.6	**242.8**
lp_osa_60	10280	1016354	81.6	58.2	**56.1**	57.1	75.8
lp_ken_11	14694	82454	82.6	62.2	814.4	60.2	**58.2**
lp_pds_10	16558	149658	1089.3	900.3	534.0	559.0	**398.5**
lp_stocfor3	16675	223395	2664.4	546.0	49.4	51.1	**48.5**
lp_ken_13	28632	161804	114.4	85.8	1912.9	82.6	**80.6**
lp_pds_20	33874	320120	2180.3	1269.5	1051.7	932.7	**869.2**
average profile				166.0	168.8	124.5	**110.7**
number of smallest profiles				7	4	7	**30**

Table 5.3: Performance comparison on Dataset 3

matrix properties			normalized profile				
name	n	nnz	orig.	GK+H	S+H	HS+H	HP
saylr3	1000	3750	34.7	21.8	21.6	23.0	**18.1**
sherman1	1000	3750	34.7	21.8	21.6	23.0	**18.1**
dwt_1005	1005	8621	120.5	33.1	32.4	29.4	**28.7**
dwt_1007	1007	8575	25.6	20.9	21.0	**18.9**	**18.9**
jagmesh2	1009	6865	53.8	23.2	23.2	31.9	**23.0**
lshp1009	1009	6865	53.8	23.2	23.2	31.9	**23.0**
can_1054	1054	12196	254.3	32.9	32.1	29.3	**28.6**
can_1072	1072	12444	258.6	41.5	40.4	30.8	**30.1**
bcsstk08	1074	12960	223.6	77.6	52.5	**51.4**	53.3
lock1074	1074	51588	96.2	70.1	71.2	67.0	**66.7**
bcsstk09	1083	18437	57.8	56.1	56.1	56.1	**53.5**
bcsstk10	1086	22070	34.1	**18.1**	**18.1**	**18.1**	23.1
bcsstm10	1086	22092	34.1	**18.1**	**18.1**	**18.1**	19.9
jagmesh3	1089	7361	65.1	**22.5**	**22.5**	**22.5**	22.9
1138_bus	1138	4054	80.5	13.6	8.9	**8.3**	9.1
jagmesh7	1138	7450	36.9	17.1	17.1	**16.1**	16.2
jagmesh8	1141	7465	30.2	22.0	20.6	23.8	**19.5**
eris1176	1176	18552	76.3	17.8	19.5	**17.0**	17.8
jagmesh5	1180	7750	29.5	17.6	16.0	**14.1**	**14.1**
bcsstk27	1224	56126	40.3	40.3	**40.1**	**40.1**	46.5
bcsstm27	1224	56126	40.3	40.3	**40.1**	**40.1**	46.5
dwt_1242	1242	10426	88.7	32.1	27.7	28.1	**26.9**
lshp1270	1270	8668	61.0	26.0	26.1	**25.7**	25.8
jagmesh9	1349	9101	51.2	22.9	22.9	19.8	**19.2**
jagmesh6	1377	8993	20.9	**11.7**	12.7	11.9	11.8
jagmesh4	1440	9504	30.9	**18.3**	**18.3**	**18.3**	19.8
bcspwr06	1454	5300	52.0	13.5	11.2	9.4	**8.1**
bcsstk11	1473	34241	90.8	44.6	45.0	**44.4**	46.8
bcsstk12	1473	34241	90.8	44.6	45.0	**44.4**	46.8
bcsstm12	1473	19659	90.2	25.3	**24.6**	27.1	24.7
lshp1561	1561	10681	68.3	28.8	28.8	40.7	**28.5**
bcspwr07	1612	5824	55.2	14.3	11.0	10.3	**8.2**
bcspwr08	1624	6050	58.4	15.1	12.4	9.6	**9.5**
bcspwr09	1723	6511	274.2	15.5	12.7	**10.0**	10.6
bcsstk14	1806	63454	108.4	97.9	86.5	90.2	**83.6**
lshp1882	1882	12904	75.6	31.7	31.7	**31.1**	31.2
plat1919	1919	32399	645.5	48.2	**39.8**	41.5	40.2
bcsstk26	1922	30336	98.9	89.2	61.2	63.2	**50.6**
bcsstk13	2003	83883	217.1	209.1	246.8	211.5	**169.7**
bcsstm13	2003	21943	52.7	28.7	32.5	27.9	**24.8**
blckhole	2132	14872	88.4	76.0	72.6	66.6	**49.9**
lock2232	2232	80352	54.4	46.6	**43.6**	46.2	44.7
lshp2233	2233	15337	82.9	34.5	34.5	53.3	**33.9**
lshp2614	2614	17980	90.3	37.3	37.3	36.7	**36.6**
dwt_2680	2680	25026	219.4	32.6	31.7	31.8	**31.2**
cegb2802	2802	277362	117.2	85.5	**83.8**	86.1	121.6
zenios	2873	27191	368.3	**4.4**	**4.4**	**4.4**	7.9
cegb2919	2919	321543	336.9	198.4	186.9	**174.4**	177.2

Table 5.3 – *Continued from previous page*

matrix properties			normalized profile				
name	n	nnz	orig.	GK+H	S+H	HS+H	HP
cegb3024	3024	79848	157.4	83.1	72.2	**45.6**	58.9
lshp3025	3025	20833	97.6	40.1	40.1	62.8	**39.6**
bcsstk23	3134	45178	332.5	216.7	216.4	214.2	**204.8**
cegb3306	3306	74916	108.0	104.0	**55.7**	56.5	68.5
sstmodel	3345	22749	32.3	24.9	21.0	21.3	**20.5**
lshp3466	3466	23896	105.0	42.9	42.9	**42.1**	42.5
lock3491	3491	160444	172.7	153.1	**119.1**	135.2	131.4
bcsstk24	3562	159910	569.4	152.1	**118.3**	118.6	144.3
saylr4	3564	22316	188.5	78.8	78.2	73.8	**71.8**
bcsstk21	3600	26600	117.2	**47.9**	**47.9**	52.2	49.0
bcsstk15	3948	117816	251.8	240.0	185.7	178.2	**166.7**
bcsstk28	4410	219024	180.8	173.2	181.9	128.2	**114.6**
bcsstk16	4884	290378	125.0	**120.8**	121.0	121.5	121.6
bcspwr10	5300	21842	1155.1	39.0	30.4	**22.0**	22.6
man_5976	5976	225046	211.7	206.6	191.3	**168.0**	205.5
bcsstk33	8738	591904	408.7	394.9	**319.9**	358.8	366.3
bcsstk17	10974	428650	258.6	234.6	200.3	**186.7**	201.3
bcsstk18	11948	149090	427.6	253.2	240.0	168.6	**139.5**
bcsstk29	13992	619488	531.8	504.1	403.8	**183.9**	189.3
bcsstk25	15439	252241	190.7	161.0	165.0	152.2	**151.6**
bcsstk30	28924	2043492	542.4	533.2	552.2	294.3	**288.2**
bcsstk31	35588	1181416	650.9	622.0	915.1	513.5	**449.6**
bcsstk32	44609	2014701	2477.7	1053.7	514.4	451.3	**308.8**
average profile				51.9	48.1	45.7	**44.2**
number of smallest profiles				8	16	25	**37**

5.4.2 Baseline algorithms

We use the following well-known profile reduction algorithms as the baseline algorithms for evaluating the performance of the proposed algorithm.

- *Gibbs-King:* The efficient implementation of Gibbs-King algorithm [60] provided by Lewis [61] in ACM Algorithm `582`.
- *Sloan:* The enhanced Sloan algorithm [18] provided by Reid and Scott [66] in HSL code `MC60`.
- *Hu-Scott:* The multilevel hybrid algorithm proposed and implemented by Hu and Scott [17] in HSL code `MC73`. This hybrid algorithm refines the multilevel spectral reordering using `MC60`.

- *Hager:* The efficient implementation of Hager's exchange algorithm [69] provided by Reid and Scott [70] in HSL code `MC67`. The algorithm applies the down exchange algorithm followed by the up exchange algorithm five times.

In [70], Reid and Scott report that applying Hager's exchange algorithm as a post-processing step to certain profile reduction algorithms yields smaller profiles than using them separately. Hence, we compare the performance of our algorithm against three baseline algorithms, all containing two phases, where the input matrix is reordered with one of *Gibbs-King*, *Sloan* or *Hu-Scott* in the first phase and *Hager* is applied on the reordered matrix in the second phase. Hereinafter, we use "GK+H", "S+H" and "HS+H" to respectively denote utilizing *Gibbs-King*, *Sloan* and *Hu-Scott* in the first phase and Hager in the second phase. We use the double-precision implementations of the baseline algorithms if available and compiled them with `gfortran` version `4.7.2` with the `O2` optimization flag.

5.4.3 Implementation details

We use PaToH [71] for bipartitioning hypergraphs, with the Absorption Matching as the coarsening algorithm, the Sweep as the vertex visit order during coarsening and the Kernihgan-Lin as the refinement algorithm (see PaToH manual [71]). We set the maximum imbalance ratio of each bipartition to 50% since the proposed algorithm does not require tightly-balanced bipartitions to reduce the profile.

The proposed algorithm continues recursive bipartitioning until the number of vertices in a part reduces below 25. This process results in a K-way BS form in which each row/column stripe has less than 25 rows/columns. Then, in order to determine the internal ordering of each row/column stripe, we utilize Algorithm 2 (weighted greedy) which was also proposed by Hager [69]. We slightly modify Algorithm 2 in such a way that the partial ordering induced by the K-way BS form is respected. In other words, we call modified Algorithm 2 once for each stripe to let the scoring function in each call consider only the rows

in the corresponding stripe. The original Algorithm 2 labels/orders the rows in the given matrix starting from the last row and continues towards the first row in order to score them more accurately. In a similar manner, we apply Algorithm 2 starting from the last row/column stripe R_K/C_K towards the first row/column stripe R_1/C_1. The reordering of row stripe R_k is then also used as the reordering of column stripe C_k, for each k. Our initial experiments show that using Algorithm 2 on a BS form with sufficiently small stripes leads to smaller profiles than continuing recursive bipartitioning until each stripe has one row/column. This is probably because multilevel hypergraph partitioning tools do not perform well in very small graphs.

The proposed algorithm was implemented in `C` and compiled with `gcc` version `4.7.2` with the `O2` optimization flag. All experiments were carried out on a Linux workstation equipped with six 2100-MHz quad-core CPUs and 132 GB of memory.

5.4.4 Performance comparison

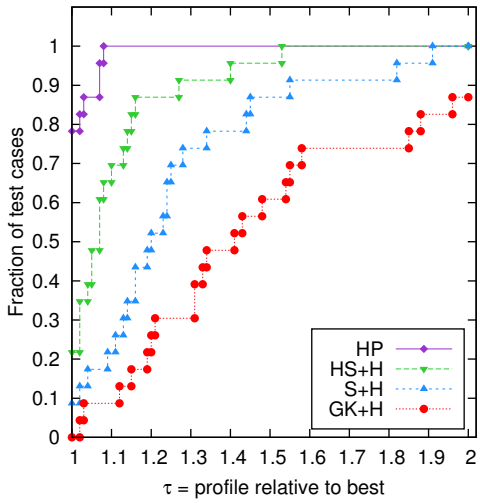
In Tables 5.1, 5.2 and 5.3, we compare the performance of the proposed algorithm against the baseline algorithms in terms of normalized profile, which is defined for a matrix as the ratio of its profile to its number of rows/columns. Hereinafter, we use "HP", which stands for "Hypergraph Partitioning", to refer to the proposed algorithm. For each matrix, we display the normalized profile of the original ordering under column "orig." and give the smallest profile attained by the compared algorithms in bold. The "average profile" and "number of smallest profiles" given at the end of each of these tables respectively refer to the geometric averages of the normalized profiles and the number of smallest profiles attained by the corresponding algorithm in the respective dataset. Note that "average profile" can be dominated by exceptionally high or low results whereas "number of smallest profiles" does not reflect the amount of improvement achieved. To overcome these problems, we also present the performance profiles of the compared algorithms in terms of normalized profiles in Figure 5.7.

Table 5.1 displays the performance comparison of the algorithms on Dataset 1. As seen in the table, in terms of both the average profile and the number of smallest profiles, the proposed HP algorithm performs significantly better than the baseline algorithms on this dataset. Among the results of the baseline algorithms, the best values for average profile and number of smallest profiles are respectively 152.7 and 4, which are both obtained by HS+H. The average profile obtained by HP is 140.6, which is 8% smaller than that of HS+H. The number of smallest profiles obtained by HP is 17, which is 3.3× more than that of HS+H.

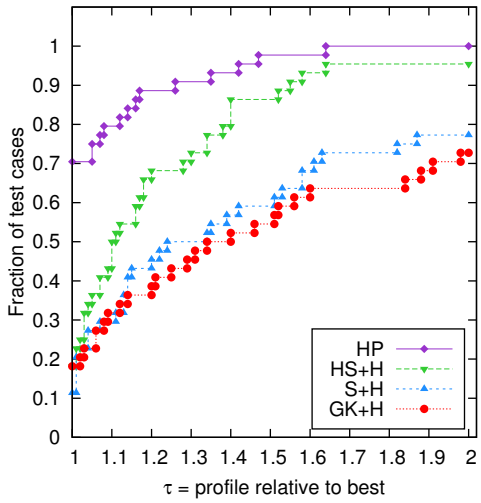
Table 5.2 displays the performance comparison of the algorithms on Dataset 2. As seen in the table, in terms of both the average profile and the number of smallest profiles, the proposed HP algorithm again performs significantly better than the baseline algorithms on this dataset. Among the baseline algorithms, the best values for average profile and number of smallest profiles are respectively 124.5 (obtained by HS+H) and 7 (obtained by both HS+H and GK+H). The average profile obtained by HP is 110.7, which is 11% smaller than that of HS+H. The number of smallest profiles obtained by HP is 30, which is again 3.3× more than that of HS+H.

Table 5.3 displays the performance comparison of the algorithms on Dataset 3. As seen in the table, in terms of both the average profile and the number of smallest profiles, the proposed HP algorithm performs slightly better than the baseline algorithms on this dataset. Among the baseline algorithms, the best values for average profile and number of smallest profiles are respectively 45.7 and 25 which are both obtained by HS+H, again. The average profile obtained by HP is 44.2, which is 3% smaller than that of HS+H. The number of smallest profiles obtained by HP is 37, which is 0.5× more than that of HS+H.

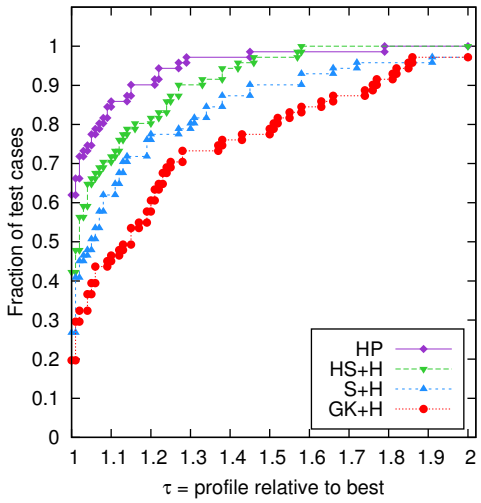
Figure 5.7 displays the performance profile plots for the compared algorithms in terms of the normalized profile on Datasets 1, 2 and 3, respectively. In the performance profile, the value Y displayed for method M in $\tau = X$ denotes that the ratio of the number of instances for which method M produces a result within a factor X of the best result to the overall dataset size is Y [72]. As seen in



(a) Dataset 1



(b) Dataset 2



(c) Dataset 3

Figure 5.7: The performance profile plots comparing the profile achieved by GK+H, S+H, HS+H and HP on Datasets 1, 2 and 3.

Table 5.4: Performance of the Harwell frontal solver MA42 applied on the matrices reordered with HS+H and HP.

	normalized profile		# of nnzs in factor matrices		# of FLOPs		factorization time (s)		time
name	HS+H	HP	HS+H	HP	HS+H	HP	HS+H	HP	ratio
nasa1824	103.5	81.4	638K	556K	108M	84M	0.164	0.140	0.85
barth4	46.2	40.2	885K	784K	62M	48M	0.128	0.128	1.00
comm._dual	36.9	31.9	999K	872K	72M	49M	0.120	0.108	0.90
pds10	559.0	398.5	106M	73M	410G	191G	217.718	122.284	0.56
copter2	557.9	603.9	138M	160M	177G	252G	236.539	299.759	1.27
lp_sctap3	54.4	41.6	1087K	772K	475M	224M	0.228	0.192	0.84
lp_degen3	183.6	157.0	2057K	1903K	1821M	1526M	0.880	0.668	0.76
lp_pds_06	340.6	242.8	35M	29M	74G	51G	21.361	15.541	0.73
sherman1	23.0	18.1	93K	68K	7179K	3369K	0.052	0.052	1.00
jagmesh8	23.8	19.5	88K	71K	3259K	2045K	0.056	0.044	0.79
bcsstk13	211.5	169.7	1798K	1530K	950M	718M	0.672	0.628	0.93
lshp3025	62.8	39.6	586K	365K	58M	20M	0.100	0.072	0.72

the figures, for any factor τ smaller than 1.6, HP produces the normalized profiles within that factor of the best result for significantly more matrices than the baseline algorithms, in all datasets. For the remaining factors, HP still performs better than the baseline algorithms in Datasets 1 and 2 whereas HS+H and HP perform comparable in Dataset 3. For all matrices in Dataset 1, HP produces the normalized profiles within a factor $\tau = 1.1$ of the best result, whereas HS+H achieves the same for 70% of the matrices. In Dataset 2, the matrices for which the normalized profiles produced by HP and HS+H are within a factor $\tau = 1.1$, respectively constitute 80% and 55% of the dataset. For Dataset 3, the improvement attained by HP compared to HS+H is smaller than the other datasets, since this dataset contains much smaller matrices than the other datasets, as seen in Table 5.3. This is probably because the proposed algorithm is top-down and thus performs better on larger matrices compared to smaller matrices.

Table 5.4 displays the performance of the Harwell frontal solver MA42 [73] applied on 12 matrices that were reordered with HS+H and HP. Since the results in terms of average profiles and number of smallest profiles displayed in Tables 5.1, 5.2 and 5.3 as well as the performance profiles displayed in Figure 5.7 consistently show that among the baseline algorithms, HS+H performs better than the others,

Table 5.5: Average running times of the profile reduction algorithms (in seconds)

	GK+H	S+H	HS+H	HP
Dataset 1	0.385	0.325	0.395	2.279
Dataset 2	0.155	0.113	0.187	0.457
Dataset 3	0.049	0.038	0.060	0.210

we only compare HP against HS+H in this experiment. The matrices used in this experiment constitute a representative set for all matrices in Datasets 1, 2 and 3. Matrices `nasa1824`, `barth4`, `commanche_dual`, `pds10` and `copter2` are from Dataset 1, matrices `lp_sctap3`, `lp_degen3` and `lp_pds 06` are from Dataset 2 and matrices `sherman1`, `jagmesh8`, `bcsstk13` and `lshp3025` are from Dataset 3. The columns under "normalized profile" display the normalized profiles produced by HS+H and HP for the respective matrix. The columns under "# of nnzs in factor matrices" display the sum of the nonzeros in factors PL and UQ obtained by HS+H and HP for the respective matrix A, where P and Q are permutation matrices, L is a lower triangular matrix and U is an upper triangular matrix in the factorization $A = PLUQ$. The columns under "# of FLOPs" and "factorization time" respectively display the number of floating-point operations in the innermost loops and the factorization times in seconds obtained by HS+H and HP for the respective matrix. The column labeled with "time ratio" denotes the ratio of the factorization time obtained by HP to the one obtained by HS+H, for the respective matrix. As seen in the table, for most of these matrices, a smaller profile produced by HP results in a smaller factorization time. The correlation between a small profile and a small factorization time can be observed better for matrices `pds10` and `lshp3025`. For matrices `barth4` and `sherman1`, although the factorization times obtained by HS+H and HP are equal, the smaller profile obtained by HP leads to a less number of FLOPs compared to HS+H.

Table 5.5 displays the running times of the proposed algorithm together with the ones of the baseline algorithms in seconds as the geometric averages over the matrices in each dataset. As seen in the table, running times of the baseline algorithms are almost comparable with each other where HS+H is the slowest one for each dataset. HP runs 5.8×, 2.4× and 3.5× slower than HS+H, respectively

for Dataset 1, 2 and 3. The higher running times of HP can be justified for applications that involve multiple iterations with a factorization in each iteration such as interior point methods used for solving linear programming problems.

5.5 Conclusion

We proposed and implemented a novel two-phase approach for reducing the profile of sparse matrices through symmetric row/column permutation. In the first phase, we use a top-down approach that makes use of the global view of the sparsity pattern of the given matrix to obtain a block structure with a smaller profile through recursive row/column bipartitioning. Two quality metrics identified for small profile of a block structure are successfully encapsulated by the proposed novel hypergraph model during recursive bipartitionings. The multilevel paradigm utilized in the state-of-the-art hypergraph partitioning tool used for bipartitioning hypergraphs successfully capture the global view of the sparsity of the matrix during the recursive bipartitioning process. In the second phase, we adopt an existing constructive algorithm that makes use of the local view of row stripes of the block structure obtained in the first phase. We have tested our proposed approach on three commonly used data sets against state-of-the-art profile reduction codes and showed its validity by producing significantly better solutions.

Chapter 6

Conclusion

In this thesis, we presented graph and hypergraph partitioning models for performance improvement in different sparse matrix computations. These computations include sparse matrix dense matrix multiplication (SpMM), sparse matrix vector multiplication (SpMV) and the envelope methods used in solvers for sparse linear systems of equations. For SpMM and SpMV, we particularly considered their distributed-memory parallel implementations. For parallel SpMM, we proposed graph and hypergraph partitioning models each of which minimizes multiple volume-related communication cost metrics simultaneously. For parallel SpMV, we proposed two hypergraph partitioning models each of which simultaneously minimizes total volume and total number of messages. For the factorization in the envelope methods, we proposed a hypergraph-partitioned-based reordering model reducing the matrix profile. In each of these models, the RB paradigm allows targeting a complicated partitioning objective tailored for the respective computation. The proposed models were compared against the state-of-the-art algorithms on large sets of test matrices and significant performance improvents are observed over these algorithms.

Bibliography

[1] O. Selvitopi and C. Aykanat, "Reducing latency cost in 2d sparse matrix partitioning models," *Parallel Computing*, vol. 57, pp. 1 – 24, 2016.

[2] D. P. O'Leary, "The block conjugate gradient algorithm and related methods," *Linear Algebra and its Applications*, vol. 29, no. 0, pp. 293 – 322, 1980. Special Volume Dedicated to Alson S. Householder.

[3] D. P. O'Leary, "Parallel implementation of the block conjugate gradient algorithm," *Parallel Computing*, vol. 5, no. 12, pp. 127 – 139, 1987. Proceedings of the International Conference on Vector and Parallel Computing-Issues in Applied Research and Development.

[4] Y. Feng, D. Owen, and D. Peri, "A block conjugate gradient method applied to linear systems with multiple right-hand sides," *Computer Methods in Applied Mechanics and Engineering*, vol. 127, no. 14, pp. 203 – 215, 1995.

[5] A. Murli, L. D'Amore, G. Laccetti, F. Gregoretti, and G. Oliva, "A multi-grained distributed implementation of the parallel block conjugate gradient algorithm," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 15, pp. 2053–2072, 2010.

[6] R. G. Grimes, J. G. Lewis, and H. D. Simon, "A shifted block lanczos algorithm for solving sparse symmetric generalized eigenproblems," *SIAM J. Matrix Anal. Appl.*, vol. 15, pp. 228–272, Jan. 1994.

[7] M. Sadkane, "A block Arnoldi-Chebyshev method for computing the leading eigenpairs of large sparse unsymmetric matrices," *Numerische Mathematik*, vol. 64, no. 1, pp. 181–193, 1993.

[8] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system implementation and observations," in *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, (Washington, DC, USA), pp. 229–238, IEEE Computer Society, 2009.

[9] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader, "Scalable graph exploration on multicore processors," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA), pp. 1–11, IEEE Computer Society, 2010.

[10] Z. Shi and B. Zhang, "Fast network centrality analysis using GPUs," *BMC Bioinformatics*, vol. 12, no. 1, 2011.

[11] A. Buluç and J. R. Gilbert, "The Combinatorial BLAS: Design, implementation, and applications," *Int. J. High Perform. Comput. Appl.*, vol. 25, pp. 496–509, Nov. 2011.

[12] A. Buluç and K. Madduri, "Parallel breadth-first search on distributed memory systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, (New York, NY, USA), pp. 65:1–65:12, ACM, 2011.

[13] A. E. Sarıyüce, E. Saule, K. Kaya, and U. V. Çatalyürek, "Regularizing graph centrality computations," *Journal of Parallel and Distributed Computing*, vol. 76, no. 0, pp. 106 – 119, 2015. Special Issue on Architecture and Algorithms for Irregular Applications.

[14] S. Wasserman, *Social network analysis: Methods and applications*, vol. 8. Cambridge University Press, 1994.

[15] S. T. Barnard, A. Pothen, and H. D. Simon, "A spectral algorithm for envelope reduction of sparse matrices," *Numerical Linear Algebra with Applications*, vol. 2, no. 4, pp. 317–334, 1995.

[16] E. G. Boman and B. Hendrickson, "A Multilevel Algorithm for Reducing the Envelope of Sparse Matrices," Tech. Rep. SCCM-96-14, Stanford University, Stanford, CA, 1996.

[17] Y. Hu and J. Scott, "A multilevel algorithm for wavefront reduction," *SIAM Journal on Scientific Computing*, vol. 23, no. 4, pp. 1352–1375, 2001.

[18] S. W. Sloan, "An algorithm for profile and wavefront reduction of sparse matrices," *International Journal for Numerical Methods in Engineering*, vol. 23, no. 2, pp. 239–251, 1986.

[19] U. V. Çatalyürek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 10, pp. 673–693, July 1999.

[20] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified NP-complete problems," in *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, (New York, NY, USA), pp. 47–63, ACM, 1974.

[21] T. Lengauer, *Combinatorial algorithms for integrated circuit layout.* New York, NY, USA: John Wiley & Sons, Inc., 1990.

[22] U. V. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, SC '01, (New York, NY, USA), pp. 28–28, ACM, 2001.

[23] G. Karypis and V. Kumar, "Multilevel algorithms for multi-constraint hypergraph partitioning," Tech. Rep. 99-034, University of Minnesota, Dept. Computer Science/Army HPC Research Center, Minneapolis, MN 55455, 1998.

[24] Reprinted from *Parallel Computing*, Vol. 59, S. Acer, O. Selvitopi and C. Aykanat, "Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems", pp. 71–96, Copyright (2016), with permission from Elsevier.

[25] E.-J. Im and K. Yelick, "Optimization of sparse matrix kernels for data mining," in *submitted to First SIAM Conf. on Data Mining*, Citeseer, 2000.

[26] A. Buluç and J. R. Gilbert, "Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C170–C191, 2012.

[27] L. A. N. Amaral, A. Scala, M. Barthélémy, and H. E. Stanley, "Classes of small-world networks," *Proceedings of the National Academy of Sciences*, vol. 97, no. 21, pp. 11149–11152, 2000.

[28] "Intel Math Kernel Library," 2015.

[29] "Nvidia cuSPARSE Library," 2015.

[30] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," in *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '95, (New York, NY, USA), ACM, 1995.

[31] B. Hendrickson and T. G. Kolda, "Partitioning rectangular and structurally unsymmetric sparse matrices for parallel processing," *SIAM J. Sci. Comput.*, vol. 21, pp. 2048–2072, Dec. 1999.

[32] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, pp. 359–392, Dec. 1998.

[33] B. Uçar and C. Aykanat, "Revisiting hypergraph models for sparse matrix partitioning," *SIAM Rev.*, vol. 49, pp. 595–603, November 2007.

[34] B. Uçar and C. Aykanat, "Partitioning sparse matrices for parallel preconditioned iterative methods," *SIAM J. Sci. Comput.*, vol. 29, pp. 1683–1709, June 2007.

[35] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *High-Performance Computing and Networking* (H. Liddell, A. Colbrook, B. Hertzberger, and P. Sloot, eds.), vol. 1067 of *Lecture Notes in Computer Science*, pp. 493–498, Springer Berlin Heidelberg, 1996.

[36] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: applications in vlsi domain," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 7, pp. 69–79, March 1999.

[37] K. Devine, E. Boman, R. Heapby, B. Hendrickson, and C. Vaughan, "Zoltan data management service for parallel dynamic applications," *Computing in Science and Engg.*, vol. 4, pp. 90–97, March 2002.

[38] B. Uçar and C. Aykanat, "Minimizing communication cost in fine-grain partitioning of sparse matrices," in *Computer and Information Sciences - ISCIS 2003* (A. Yazıcı and C. Şener, eds.), vol. 2869 of *Lecture Notes in Computer Science*, pp. 926–933, Springer Berlin Heidelberg, 2003.

[39] B. Uçar and C. Aykanat, "Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies," *SIAM J. Sci. Comput.*, vol. 25, no. 6, pp. 1837–1859, 2004.

[40] R. H. Bisseling and W. Meesen, "Communication balancing in parallel sparse matrix-vector multiply," *Electronic Transactions on Numerical Analysis*, vol. 21, pp. 47–65, 2005.

[41] U. V. Çatalyürek, M. Deveci, K. Kaya, and B. Uçar, "UMPa: A Multi-objective, multi-level partitioner for communication minimization," in *Graph Partitioning and Graph Clustering 2012* (D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, eds.), vol. 588 of *Contemporary Mathematics*, pp. 53–66, AMS, 2013.

[42] M. Deveci, K. Kaya, B. Uçar, and U. V. Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *Journal of Parallel and Distributed Computing*, vol. 77, no. 0, pp. 69 – 83, 2015.

[43] O. Selvitopi, S. Acer, and C. Aykanat, "A recursive hypergraph bipartitioning framework for reducing bandwidth and latency costs simultaneously," *IEEE Transactions on Parallel and Distributed Systems, to appear*, 2016.

[44] B. Hendrickson, "Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract)," in *Proceedings of the 5th International*

Symposium on Solving Irregularly Structured Problems in Parallel, IRREGULAR '98, (London, UK, UK), pp. 218–225, Springer-Verlag, 1998.

[45] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, pp. 1519–1534, Nov. 2000.

[46] V. Kumar, *Introduction to Parallel Computing*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2002.

[47] T. A. Davis, "University of Florida Sparse Matrix Collection," *NA Digest*, vol. 92, 1994.

[48] "10th DIMACS Implementation Challenge: Graph Partitioning and Graph Clustering," 2011.

[49] B. Uçar and C. Aykanat, "A library for parallel sparse matrix-vector multiplies," Tech. Rep. BU-CE-0506, Bilkent University, Computer Engineering Department, June 2005. Also available at `http://www.cs.bilkent.edu.tr/tech-reports/2005`.

[50] K. Schloegel, G. Karypis, and V. Kumar, "Parallel multilevel algorithms for multi-constraint graph partitioning," in *Euro-Par 2000 Parallel Processing* (A. Bode, T. Ludwig, W. Karl, and R. Wismller, eds.), vol. 1900 of *Lecture Notes in Computer Science*, pp. 296–310, Springer Berlin Heidelberg, 2000.

[51] B. Vastenhouw and R. H. Bisseling, "A two-dimensional data distribution method for parallel sparse matrix-vector multiplication," *SIAM Rev.*, vol. 47, pp. 67–95, January 2005.

[52] D. Pelt and R. Bisseling, "A medium-grain method for fast 2d bipartitioning of sparse matrices," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 529–539, May 2014.

[53] E. Kayaaslan, B. Ucar, and C. Aykanat, "Semi-two-dimensional partitioning for parallel sparse matrix-vector multiplication," in *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pp. 1125–1134, May 2015.

[54] U. V. Çatalyürek, C. Aykanat, and B. Uçar, "On two-dimensional sparse matrix partitioning: Models, methods, and a recipe," *SIAM J. Sci. Comput.*, vol. 32, pp. 656–683, Feb. 2010.

[55] U. Catalyurek and C. Aykanat, "A fine-grain hypergraph model for 2d decomposition of sparse matrices," in *Proceedings of the 15th International Parallel &Amp; Distributed Processing Symposium*, IPDPS '01, (Washington, DC, USA), pp. 118–, IEEE Computer Society, 2001.

[56] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang, "PETSc users manual," Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.

[57] Y. Lin and J. Yuan, "Profile minimization problem for matrices and graphs," *Acta Mathematicae Applicatae Sinica*, vol. 10, no. 1, pp. 107–112, 1994.

[58] A. George, *Computer implementation of the finite element method.* PhD thesis, Stanford University, Stanford, CA, 1971.

[59] N. E. Gibbs, W. G. Poole, and P. K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix," *SIAM Journal on Numerical Analysis*, vol. 13, no. 2, pp. 236–250, 1976.

[60] N. E. Gibbs, "Algorithm 509: A hybrid profile reduction algorithm [F1]," *ACM Trans. Math. Softw.*, vol. 2, pp. 378–387, Dec. 1976.

[61] J. G. Lewis, "Implementation of the Gibbs-Poole-Stockmeyer and Gibbs-King algorithms," *ACM Trans. Math. Softw.*, vol. 8, pp. 180–189, June 1982.

[62] Q. Wang, Y.-C. Guo, and X.-W. Shi, "A generalized GPS algorithm for reducing the bandwidth and profile of a sparse matrix," *Progress In Electromagnetics Research*, vol. 90, pp. 121–136, 2009.

[63] Q. Wang, X. Shi, C. Guo, and Y. Guo, "An improved GPS method with a new pseudo-peripheral nodes finder in finite element analysis," *Finite Elements in Analysis and Design*, vol. 48, no. 1, pp. 1409 – 1415, 2012.

[64] I. S. Duff, J. K. Reid, and J. A. Scott, "The use of profile reduction algorithms with a frontal code," *International Journal for Numerical Methods in Engineering*, vol. 28, no. 11, pp. 2555–2568, 1989.

[65] G. Kumfert and A. Pothen, "Two improved algorithms for envelope and wavefront reduction," *BIT Numerical Mathematics*, vol. 37, no. 3, pp. 1–32, 1997.

[66] J. K. Reid and J. A. Scott, "Ordering symmetric sparse matrices for small profile and wavefront," *International Journal for Numerical Methods in Engineering*, vol. 45, no. 12, pp. 1737–1755, 1999.

[67] B. Hendrickson and R. Leland, "An improved spectral graph partitioning algorithm for mapping parallel computations," *SIAM Journal on Scientific Computing*, vol. 16, no. 2, pp. 452–469, 1995.

[68] G. Karypis and V. Kumar, "Parallel multilevel series k-way partitioning scheme for irregular graphs," *SIAM Review*, vol. 41, no. 2, pp. 278–300, 1999.

[69] W. W. Hager, "Minimizing the profile of a symmetric matrix," *SIAM Journal on Scientific Computing*, vol. 23, no. 5, pp. 1799–1816, 2002.

[70] J. K. Reid and J. A. Scott, "Implementing Hager's exchange methods for matrix profile reduction," *ACM Trans. Math. Softw.*, vol. 28, pp. 377–391, Dec. 2002.

[71] U. V. Çatalyürek and C. Aykanat, "Patoh: partitioning tool for hypergraphs," tech. rep., Department of Computer Engineering, Bilkent University, 1999.

[72] E. D. Dolan and J. J. Mor, "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2002.

[73] *HSL. A collection of Fortran codes for large scale scientific computation.* http://www.hsl.rl.ac.uk/.