HYPERGRAPH PARTITIONING AND REORDERING FOR PARALLEL SPARSE TRIANGULAR SOLVES AND TENSOR DECOMPOSITION

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

Tuğba Torun

July 2021

Hypergraph Partitioning and Reordering for Parallel Sparse Triangular Solves and Tensor Decomposition
By Tuğba Torun
July 2021

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)

Murat Manguoğlu (Co-Advisor)

Özcan Öztürk

Özgür Ulusoy

Engin Demir

Emrullah Fatih Yetkin

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

Copyright Information

ABSTRACT

HYPERGRAPH PARTITIONING AND REORDERING FOR PARALLEL SPARSE TRIANGULAR SOLVES AND TENSOR DECOMPOSITION

Tuğba Torun
Ph.D. in Computer Engineering
Advisor: Cevdet Aykanat
Co-Advisor: Murat Manguoğlu
July 2021

Several scientific and real-world problems require computations with sparse matrices, or more generally, sparse tensors which are multi-dimensional arrays. For sparse matrix computations, parallelization of sparse triangular systems introduces significant challenges because of the sequential nature of the computations involved. One approach to parallelize sparse triangular systems is to use sparse triangular SPIKE (stSPIKE) algorithm, which was originally proposed for shared memory architectures. stSPIKE decouples the problem into independent smaller systems and requires the solution of a much smaller reduced sparse triangular system. We extend and implement stSPIKE for distributed-memory architectures. Then we propose distributed-memory parallel Gauss-Seidel (dmpGS) and ILU (dmpILU) algorithms by means of stSPIKE. Furthermore, we propose novel hypergraph partitioning models and in-block reordering methods for minimizing the size and nonzero count of the reduced systems that arise in dmpGS and dmpILU. For sparse tensor computations, tensor decomposition is widely used in the analysis of multi-dimensional data. The canonical polyadic decomposition (CPD) is one of the most popular tensor decomposition methods, which is commonly computed by the CPD-ALS algorithm. Due to high computational and memory demands of CPD-ALS, it is inevitable to use a distributed-memory-parallel algorithm for efficiency. The medium-grain CPD-ALS algorithm, which adopts multi-dimensional cartesian tensor partitioning, is one of the most successful distributed CPD-ALS algorithms for sparse tensors. We propose a novel hypergraph partitioning model, CartHP, whose partitioning objective correctly encapsulates the minimization of total communication volume of multi-dimensional cartesian tensor partitioning. Extensive experiments on real-world sparse matrices and tensors validate the parallel scalability of the proposed algorithms as well as the effectiveness of the proposed hypergraph partitioning and reordering models.

Keywords: Hypergraph partitioning, distributed-memory architectures, sparse matrix, sparse tensor, sparse linear system solution, parallel sparse triangular solve, SPIKE algorithm, parallel Gauss-Seidel, incomplete LU factorization, ILU(0), tensor decomposition, canonical polyadic decomposition (CPD), cartesian partitioning, communication volume.

ÖZET

PARALEL SEYREK ÜÇGENSEL SİSTEMLER VE TENSÖR AYRIŞTIRMA İÇİN HİPERÇİZGE BÖLÜMLEME VE YENİDEN SIRALAMA YÖNTEMLERİ

Tuğba Torun
Bilgisayar Mühendisliği, Doktora
Tez Danışmanı: Cevdet Aykanat
İkinci Tez Danışmanı: Murat Manguoğlu
Temmuz 2021

Bir çok bilimsel ve gerçek hayatta karşılaşılan problem, seyrek matris veya daha genel haliyle çok boyutlu seyrek tensör hesaplamalarını gerektirmektedir. Seyrek matris hesaplamaları için, içerdiği işlemlerin doğal seri yapısı sebebiyle, seyrek üçgensel sistemlerin paralelleştirilmesi önemli zorluklar ortaya çıkarmaktadır. Seyrek üçgensel sistemleri paralelleştirmek için bir yaklaşım, seyrek üçgensel SPIKE (stSPIKE) algoritmasını kullanmaktır. İlk olarak paylaşımlı bellekler için önerilmiş olan stSPIKE, problemi daha küçük bağımsız sistemlere ayrıştırır ve çok daha küçük bir indirgenmiş seyrek üçgensel sistemin çözümünü gerektirir. Biz bu çalışmada, stSPIKE algoritmasını dağıtık bellekli sistemler için genişleterek yazılımını gerçekleştirdik. Daha sonra, stSPIKE algoritmasını kullanarak dağıtık bellekli paralel Gauss-Seidel (dmpGS) ve ILU (dmpILU) algoritmalarını önerdik. Ayrıca, dmpGS ve dmpILU çözümünde ortaya çıkan indirgenmiş sistemlerin boyutunu ve sıfırdışı eleman sayısını en aza indirmek amacıyla özgün hiperçizge bölümleme modelleri ve blok-içi yeniden sıralama yöntemleri önerdik. Diğer yandan seyrek tensör hesaplamaları konusunda, tensör ayrıştırma, çok boyutlu verilerin analizi için oldukça yaygın kullanılmaktadır. Kanonik çok öğeli ayrıştırma (CPD), en sık kullanılan tensör ayrıştırma yöntemlerinden biridir ve yaygın olarak CPD-ALS algoritması ile çözülür. CPD-ALS algoritmasının yüksek hesaplama ve hafıza talepleri sebebiyle, dağıtık bellekli paralel bir algoritma kullanmak verimlilik için kaçınılmazdır. Çok boyutlu kartezyen tensör bölümleme yöntemini benimseyen orta ölçekli CPD-ALS algoritması, seyrek tensör ayrıştırması için önerilmiş en başarılı dağıtık bellekli CPD-ALS algoritmalarından biridir. Biz, çok boyutlu kartezyen tensör bölümlemesinin iletişim hacmini en aza indirgemeyi, bölümleme hedefiyle doğru bir şekilde karşılayan özgün bir hiperçizge bölümleme modeli (CartHP) öneriyoruz. Gerçek hayat problemlerinden elde edilmiş seyrek matris ve tensörler üzerindeki geniş kapsamlı deneyler, önerilen algortimaların paralel ölçeklenebilirliğini ve önerilen hiperçizge bölümleme ve yeniden sıralama modellerinin etkinliğini doğrular niteliktedir.

Anahtar sözcükler: Hiperçizge bölümleme, dağıtık bellekli sistemler, seyrek matris, seyrek tensör, seyrek doğrusal sistem çözümü, paralel seyrek üçgensel sistemler, SPIKE algoritması, paralel Gauss-Seidel, eksik LU faktörizasyonu, ILU(0), tensör ayrıştırması, kanonik çok öğeli ayrıştırma (CPD), kartezyen bölümleme, iletişim hacmi.

Acknowledgement

I would first like to thank my advisor Prof. Dr. Cevdet Aykanat for his insightful and enlightening guidance throughout my Ph.D. studies. I would also like to express my sincere gratitude to my co-advisor Prof. Dr. Murat Manguoğlu for his generous assistance and kindness. It was a privilege for me to have the chance to study with such leading professors.

I would like to thank the dissertation jury members Prof. Dr. Özcan Öztürk, Prof. Dr. Özgür Ulusoy, Asst. Prof. Dr. Engin Demir and Asst. Prof. Dr. Emrullah Fatih Yetkin for their valuable comments on the thesis. I would like to acknowledge Dr. Seher Acer for her substantial contributions to the progress of our joint work. I would also like to thank my colleagues from the Parallel Computing Group, especially Dr. Ozan Karsavuran, for their cooperation. I am grateful to my friends, mostly Fatma and Gamze, for their sincerity and encouragement.

I owe thanks to my family for their continuous support and motivation. I am indebted to my mom for her unbounded affection and devotion. I should thank my grandparents Necla and Metin, and my aunt Esra, for believing in me and making me who I am. I am deeply grateful to Hafize and Ali Torun for being a second mother and father to me and providing me all kinds of support. I feel so lucky to have such cheerful sisters and brothers, especially Gülşah who always volunteers to play with Büşra so that I can study relievedly. I appreciate the times that we spend with Gülten Torun, modeling us with her modesty and vigor.

I would like to say a special thank you to my pretty daughter Büşra. She has spent her entire 4-year life being so understanding for letting her mom doing research. I love her for being my precious treasure bringing joy to my life.

The person who deserves the most thanks is F. Şükrü Torun, for being a wonderful husband, a diligent collaborator, and an excellent father to our daughter. He beautifies and facilitates my life with his endless support and cherishment.

Computing resources used in this work were provided by the National Center for High Performance Computing of Turkey (UHeM) under grant number 4007772020. We acknowledge PRACE for awarding us access to resource Hazel Hen (Cray XC40) based in Germany at HLRS.

To my lovely daughter, Büşra...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A wide range of applications in science and engineering require the solution of a sparse linear system of equations

$$Ax = f, \tag{1.1}$$

where $A \in \mathbb{R}^{m\times m}$ is a general large sparse nonsingular matrix; and x and $f \in \mathbb{R}^m$ are the unknown and right hand side vectors, respectively. Depending on the numerical and structural properties of the coefficient matrix, various solvers have been proposed.

Direct solvers require a sequence of operations: reordering and partitioning, symbolic factorization, numerical factorization, and finally obtaining the solution, typically via forward and backward substitution. The reordering and partitioning schemes are used both to reduce the amount of fill-in and to enhance the parallel scalability. Symbolic factorization is used to determine the sparsity pattern of the factors, and finally the numerical factorization (such as sparse LU [1], QR [2], SVD [3] and WZ [4]) is computed. Direct solvers are robust and, in general, are known to be very scalable during the factorization phase [5, 6], but not so much during the triangular solution phase [7].

Iterative solvers, on the other hand, are known to be more scalable but not as robust as direct solvers. Nevertheless, they are still preferred for large sparse systems due to their lower memory requirements. Starting with an initial guess for the solution vector, these methods improve the solution at each iteration. There are two main types of iterative solvers: stationary and non-stationary methods.

Stationary methods have the general form $x^{(k+1)} = \phi(x^{(k)})$ where $x^{(k)}$ is the solution vector at the k^{th} iteration and $\phi(\cdot)$ is a function which does not change during the iterations. For example, $\phi(x) = Bx + g$ where B is a matrix and g is a vector, define a stationary iterative method. Jacobi, Gauss-Seidel, Successive Over Relaxation (SOR) and Symmetric SOR (SSOR) are some examples of stationary iterative solvers [2, 8]. Non-stationary methods have the form $x^{(k+1)} = \phi^{(k)}(x^{(k)})$ in which the function $\phi^{(k)}(\cdot)$ changes at each iteration; for example $\phi^{(k)}(x) = x + \alpha^{(k)} y^{(k)}$ where $\alpha^{(k)}$ is a scalar and $y^{(k)}$ is a vector at k^{th} iteration, respectively. Projection methods, Krylov subspace methods and Chebyshev iterations are some examples of non-stationary iterative methods [9, 8].

In practice, linear systems are preconditioned to reduce the required number of iterations of the iterative solvers and to improve their robustness. There could be a variety of choices of preconditioners, some are problem specific and others are more general. General classical preconditioners include, incomplete factorization based preconditoners (such as incomplete LU (ILU) [10, 8]), sparse approximate inverse [11], algebraic multigrid (AMG) [12, 13], and others. We refer the reader to [14] for a detailed survey of preconditioners. Among these preconditioners, AMG has been widely used recently in many applications [15, 16, 17] which is a generalization of Geometric Multigrid (GMG) [18]. GMG requires some knowledge of the physical problem and/or its geometry, while there is no such requirement for AMG. AMG can be also used as a direct solver [19, 20]. Furthermore, AMG typically uses another iterative method as a "smoother" which is required to reduce the error at each level and the smoother itself can also be preconditioned. More recently a preferred smoother for AMG is Gauss-Seidel [21, 22, 23], as in BoomerAMG [19] and Trilinos-ML [24].

Sparse triangular systems constitute an important kernel operation in several applications such as LU, ILU, Cholesky, Gauss-Seidel, Jacobi, SOR, SSOR, and approximate inverse preconditioners [1, 8]. However, solving triangular systems often constitutes a sequential bottleneck due to the dependencies between unknowns in forward or backward substitution operations. This increases the importance of finding efficient parallel solutions to sparse triangular systems.

In [25], a parallel banded triangular solver is proposed. This algorithm is extended for solving banded linear systems [26, 27] and further improved by implementing various alternatives in each step of the factorization including the solution of the reduced system in [28, 29, 30]. At this point, the algorithm is called SPIKE algorithm. For sparse linear systems, SPIKE is also proposed as a solver for a banded preconditioner that is sparse within the band [31, 32], and it is generalized for sparse linear systems [33, 34, 35]. In [36], a SPIKE-based parallel solver for general tridiagonal systems is implemented for GPU architectures.

A recent study [37] proposes a multi-threaded parallel solver for sparse triangular systems by extending the SPIKE algorithm [25]. This sparse triangular SPIKE (stSPIKE) algorithm decouples the triangular system into smaller systems which can be solved concurrently and requires the solution of a much smaller reduced sparse triangular system. We propose and implement a parallel sparse triangular solver by extending the stSPIKE algorithm for distributed-memory architectures.

Gauss-Seidel (GS) is a well-known stationary iterative method which solves the linear system (1.1) by splitting the coefficient matrix into its lower and strictly upper triangular parts, $A = L + U$. Then the solution is obtained iteratively by

$$x^{(k+1)} = L^{-1}(f - Ux^{(k)}).$$

In this formulation of GS, both a lower triangular system is required to be solved and an upper triangular SpMV (sparse matrix-vector multiplication) is performed at each iteration. It is known to be effective and preferred as a smoother for a wide variety of problems [21, 38]. However, a true distributed-memory parallelization of GS is considered to be a challenging task [21]. The main difficulty in parallelizing GS inherits from the sequential nature of triangular solve included in GS [38].

We propose a distributed-memory parallel GS (dmpGS) by using stSPIKE for solving the sparse lower triangular system in GS. The reduced sparse triangular system in stSPIKE constitutes a sequential bottleneck. In order to alleviate this bottleneck and to reduce the communication overhead of dmpGS, we propose a partitioning and reordering model consisting of two phases. The first phase is a novel hypergraph partitioning model whose partitioning objective simultaneously encodes minimizing the reduced system size and the communication volume. The second phase is an in-block row reordering method for decreasing the nonzero count of the reduced system. Extensive experiments on a dataset consisting of 359 sparse linear systems verify the effectiveness of the proposed partitioning and reordering model in terms of reducing the communication and the sequential computational overheads. Parallel experiments on 12 large systems using up to 320 cores demonstrate that the proposed model significantly improves the scalability of dmpGS.

Incomplete factorization techniques are known to be successful preconditioning strategies for Krylov subspace methods. At each iteration of a Krylov subspace method, incomplete LU (ILU) requires both a lower triangular and an upper triangular system to be solved. There are various ways to form these factors depending on the degree of fill-in allowed. Among them, incomlplete LU with zero fill-in, namely ILU(0), is commonly used as a preconditioner for iterative Krylov subspace-based methods in several studies [39, 40, 41]. This is because its computation and storage demands are low, and it is known to be highly effective for some important problem classes such as M-matrices or diagonally dominant matrices [42]. It is shown in [43] that ILU(0) yields slower convergence but better speedup than the other ILU preconditioners. We choose ILU(0) to study since it does not allow fill-in, and hence the reordering on the coefficient matrix directly determines the nonzero structure in the factorized triangular systems.

We propose a distributed-memory parallel algorithm, namely dmpILU, for solving the triangular systems in ILU(0) by using stSPIKE. The reduced systems in both lower and upper stSPIKE constitutes the sequential bottleneck. We propose a two-phase partitioning and reordering model for reducing the size and the nonzero count of these reduced systems simultaneously. The first phase is a novel hypergraph partitioning model whose partitioning objective encodes the minimization of the total reduced system size in lower and upper stSPIKE. The second phase is an in-block row reordering method for decreasing the total nonzero count of these reduced systems. Extensive experiments verify the effectiveness of the proposed partitioning and reordering model.

Tensors are multi-dimensional arrays which can consist of three or more dimensions (modes). The applications that make use of sparse tensors often benefit from tensor decomposition to discover the latent features of the modes. The most popular tensor decomposition method achieving this feat is the canonical polyadic decomposition (CPD) [44, 45, 46]. CPD is an extension of singular value decomposition for tensors and approximates a given tensor as a sum of rank-one tensors. One common method for computing CPD is the CPD-ALS algorithm, which exploits the alternating least squares method [47]. CPD-ALS includes a bottleneck operation called Matricized Tensor Times Khatri-Rao Product (MTTKRP), which requires significantly large amounts of computation and memory. This necessitates an efficient distributed-memory implementation for the CPD-ALS algorithm.

Recently, Smith and Karypis [48] have proposed a successful distributed-memory implementation of CPD-ALS algorithm. Their algorithm adopts a medium-grain model, in which a cartesian partition of the input tensor is utilized. Cartesian partitioning has the nice property of confining the communications to the layers of a virtual multi-dimensional processor mesh, thus providing upper bounds on communication overheads. Hence, this algorithm outperforms the earlier CPD-ALS implementations by achieving smaller parallel runtimes and better scalability.

In order to obtain a cartesian partition of the tensor, the medium-grain algorithm applies block partitioning on each mode, which is randomly permuted beforehand to maintain balance on the number of tensor nonzeros assigned to processors, hence their computational loads. However, this algorithm does not utilize the sparsity pattern of the tensor to minimize the total communication volume. The objective of this work is to fill this literature gap by proposing an intelligent partitioning algorithm that utilizes the sparsity pattern for minimizing the total communication volume of the medium-grain model. For this purpose, we exploit the conceptual similarity between MTTKRP and sparse matrix vector multiplication (SpMV), for which many partitioning models and methods with different granularities are well-studied [49, 50, 51, 52]. The 2D cartesian partitioning for parallel SpMV, which is known as checkerboard partitioning, was first introduced by Hendrickson et al. [53] and its total communication volume is minimized by a hypergraph partitioning (HP) model, CBHP, proposed by Çatalyürek and Aykanat [54, 50]. Relying on the similarity between MTTKRP and SpMV, extending CBHP for cartesian partitioning of tensors with more than two dimensions seems promising for minimizing the total communication volume of the medium-grain CPD-ALS.

CBHP is a two-phase HP model, where row and column partitions are respectively obtained in the first and second phases. The row partition obtained in the first phase implies a division information in each column. However, this column division information is not utilized in the topology of the hypergraph formed in the second phase. On the contrary, in the case of more than two dimensions, a slice's division information obtained in a phase needs to be utilized in each of the subsequent phases which further divide that slice. Note that this need does not arise for the two-dimensional case since each row/column is divided in exactly one phase. Since the direct extension of the CBHP model for tensor partitioning does not keep division history, it fails to correctly encapsulate the objective of minimizing the total communication volume.

In order to overcome the above-mentioned problem on extending the CBHP model for more than two dimensions, we propose a new hypergraph partitioning model in which hypergraph topologies contain the priori division information of slices. The partitioning objective of our model encapsulates the minimization of the total communication volume of the medium-grain CPD-ALS. To validate the proposed model, we conduct parallel experiments on 12 real-world tensors for up to 1024 processors. Compared to the baseline medium-grain model [48], the proposed model achieves average reductions of 52%, 43% and 24% in total communication volume, communication time and overall runtime of CPD-ALS, respectively.

This thesis is organized as follows. Sections 2 and 3 provide the background information and the related work, respectively. We introduce the proposed dmpGS and dmpILU algorithms along with the partitioning and reordering models for their efficiency in Sections 4 and 5, respectively. The proposed hypergraph partitioning model for sparse tensor decomposition is explained in Section 6. Finally, Section 7 concludes the thesis.

Chapter 2

Background

In this chapter, we provide the background information related to hypergraphs, linear system solution methods, sparse triangular SPIKE (stSPIKE) algorithm and sparse tensors. Throughout the thesis, we use a semicolon as in MATLAB notation, e.g., $\mathbf{A}(i,:)$, to refer to a varying index.

2.1 Hypergraphs

Hypergraphs can be defined as generalization of graphs in which a more general form of edges called *nets* can connect any number of vertices. A hypergraph $\mathcal{H}=(\mathcal{V},\mathcal{N})$ consists of a set of vertices $\mathcal{V}=\{v_i\}_{1\leq i\leq n}$ and a set of nets $\mathcal{N}=\{n_j\}_{1\leq j\leq m}$. Each net $n_j \in \mathcal{N}$ connects a subset of vertices in $\mathcal{V}$, which is referred to as the *pins* of n_j, and denoted by $Pins(n_j)$ or $Pins(n_j,\mathcal{H})$, depending on the necessity. Each vertex v_i is assigned a weight of $w(v_i)$ whereas each net n_j is assigned a cost of $c(n_j)$. $\Pi=\{\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_k\}$ is a K-way partition of $\mathcal{H}$, if parts are mutually disjoint and exhaustive. The weight of a part is the sum of the weights of vertices in that part. For a given partition, if a net connects at least one vertex in a part, it is said to *connect* that part. *Connectivity* $\lambda(n_j)$ of net n_j is the number of parts connected by n_j. If a net n_j connects multiple parts (i.e. $\lambda(n_j)>1$), it is called *cut*; and *internal*, otherwise (i.e. $\lambda(n_j)=1$). The set of cut nets is denoted by $\mathcal{N}_{cut}$. The *cutsize* of Π is defined in various ways. Two most commonly used cutsize definitions are the *cut-net* and the *connectivity* metrics [55], which are respectively defined as

$$cs_{cutn}(\Pi) = \sum_{n \in \mathcal{N}_{cut}} c(n), \text{ and} \tag{2.1}$$

$$cs_{conn}(\Pi) = \sum_{n \in \mathcal{N}_{cut}} (\lambda(n)-1)c(n). \tag{2.2}$$

2.1.1 Hypergraph Partitioning (HP)

The *Hypergraph partitioning (HP)* problem is defined as finding a K-way partition Π of a given hypergraph $\mathcal{H}$ with the objective of minimizing the cutsize and the constraint of maintaining balance on the weights of the parts. The balance criterion is formulated as

$$W_{max} \leq W_{avg}(1+\epsilon), \tag{2.3}$$

where ϵ denotes the given maximum allowable imbalance ratio; and W_{max} and W_{avg} respectively denote the maximum and average part weights.

In the case of multi-constraint hypergraph partitioning with C constraints, the c^{th} constraint for $c = 1, 2, \ldots, C$ is formulated as

$$W_c(\mathcal{V}_k) \leq W_c^{tot}(1+\epsilon)/K. \tag{2.4}$$

Here, $W_c(\mathcal{V}_k)$ and W_c^{tot} denote the sums of the c^{th} weights of the vertices in $\mathcal{V}_k$ and $\mathcal{V}$, respectively.

HP with fixed vertices ensures to assign some preassigned vertices which are called *fixed vertices* to the respective parts. The rest of the vertices, namely *free vertices*, are free to be assigned to any part.

The *recursive bipartitioning (RB)* is a widely used paradigm to obtain a K-way HP. It first partitions the hypergraph into two and then each part is further bipartitioned recursively until reaching the desired number of parts K. In order to encode the cut-net and connectivity metrics, cut-net removal and cut-net splitting methods are utilized in the RB-based HP, respectively [55].

2.1.2 Sparse Matrix Partitioning with HP

Several HP models and methods have been proposed and successfully utilized for obtaining matrix partitioning [56, 57, 58, 59, 60, 61, 62, 63, 64]. Among these, the most relevant one is the *column-net* model [55] that represents a given sparse matrix A as a hypergraph $\mathcal{H}_{CN}(A)$ in which nets and vertices respectively represent columns and rows. In this model, vertex v_i is added to the pin list of net n_j for each nonzero $A(i,j)$ in A. Throughout the thesis, row r_i and column c_j respectively denote both the vector and the index of row i and column j interchangeably, depending on the context.

A K-way ordered partition $\Pi = \langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle$ of the column-net model $\mathcal{H}_{CN}(A)$ is decoded as a partial reordering of the rows of A in such a way that the rows corresponding to vertices in $\mathcal{V}_k$ are ordered before the rows corresponding to the vertices in $\mathcal{V}_\ell$ for $k<\ell$. This is a partial reordering since the rows corresponding to the vertices in the same part can be ordered arbitrarily. Let $\mathcal{B}^r_k$ denote the k^{th} row block which contains the rows corresponding to the vertices in $\mathcal{V}_k$. We consider a symmetric row-column reordering that yields a 2D grid structure of A. The submatrix consisting of the rows of $\mathcal{B}^r_k$ and columns of ℓ^{th} column block $\mathcal{B}^c_\ell$ is referred as block-(k,ℓ) of A. A column is said to *link* a row block $\mathcal{B}^r_k$ if it contains at least one nonzero in $\mathcal{B}^r_k$. A column is called a *linking column* if it links more than one row block. For a matrix with nonzero diagonal entries, each column links a diagonal block and becomes a linking column if it links at least one off-diagonal block.

In the column-net model with unit net cost, the partitioning objective using the connectivity (2.2) and cut-net metrics (2.1) respectively encode the minimization of the number of nonzero column segments in off-diagonal blocks and the number of linking columns. The former partitioning objective is successfully utilized in encoding the minimization of the row parallel SpMV operations [55].

2.2 Linear System Solution Methods

There are numerous methods proposed to solve sparse linear systems. Here, we present the ones that are relevant to this thesis work involving sparse triangular system solution.

2.2.1 Gauss-Seidel(GS)

Gauss Seidel is an iterative method to solve the linear system (1.1) by decomposing the coefficient matrix A into its lower triangular component L and its strictly upper triangular component U such that $A = L + U$. The equations can be rewritten as

$$\begin{aligned} Ax &= f \\ (L+U)x &= f \\ Lx &= f - Ux \\ x &= L^{-1}(f - Ux). \end{aligned} \tag{2.5}$$

The Gauss-Seidel algorithm iteratively solves

$$x^{(k+1)} = L^{-1}(f - Ux^{(k)}), \tag{2.6}$$

where $x^{(k)}$ is the approximate value of x at k^{th} iteration. Note that at each iteration, a sparse matrix-vector product is done for Ux^k and a triangular solve is needed for L^{-1}. GS is guaranteed to converge if A is strictly or irreducibly diagonal dominant [65] or symmetric positive definite [2]. In practice, the iterations of GS are continued until reaching a sufficient state of convergence, which can be checked in different ways. One way is to check the residual ($\|f - Ax^{(k)}\|$) or the relative residual ($\|f - Ax^{(k)}\|/\|f\|$) drops less than a pre-determined threshold. However, it requires to compute $Ax^{(k)}$ at each iteration, which could be costly. Another alternative is to check whether $\|x^{(k+1)} - x^{(k)}\|$ drops less than a certain threshold, which we adapt in this work.

2.2.2 LU decomposition

LU decomposition is a direct method for solving the linear system (1.1) by factorizing the coefficient matrix as the product $A = LU$, where L and U are lower and upper triangular matrices, respectively. It requires solving two triangular systems such that

$$\begin{aligned} y &= L^{-1} f \\ x &= U^{-1} y. \end{aligned} \tag{2.7}$$

This triangular systems are usually solved with forward and backward substitution schemes which are naturally hard to parallelize due to dependencies between the unknowns. For sparse matrices, L and U factors in the LU decomposition may have much more nonzeros than the original matrix, which are called *fill-in*.

2.2.3 Incomplete LU (ILU) decomposition

The memory requirements of using a direct solver due to high fill-in may constitute a bottleneck when solving sparse linear systems. Incomplete LU (ILU) decomposition seeks triangular matrices L and U such that $A \approx LU$ rather than $A = LU$. Due to small (or none) fill-in rates, the system $LUx = f$ can be solved faster but does not yield the exact solution for $Ax = f$. Therefore, the matrix $M = LU$ is instead used as a preconditioner in iterative methods such as the conjugate gradient method or the generalized minimal residual (GMRES) method.

A common method is to choose L and U with the sparsity pattern same as the sparsity pattern of the coefficient matrix, which is called ILU(0) since there is no allowed fill-in. The complexity of ILU(0) is low with respect to the other ILU methods due to its sparsity pattern. The L and U factors are computed as in the Gaussian Elimination, but the computation is done only for the nonzero pattern of A. This factorization is computed only once and the upper and lower triangular systems with L and U as in (2.7) are solved at each iteration of the

Algorithm 1 Incomplete LU (ILU) Factorization

for $i \leftarrow 2$ **to** m **do**
  for $k \leftarrow 1$ **to** $i-1$ **where** $(i,k) \in \mathcal{Z}$ **do**
    $a_{ik} \leftarrow a_{ik}/a_{kk}$
    for $j \leftarrow k+1$ **to** m **where** $(i,j) \in \mathcal{Z}$ **do**
      $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$

iterative Krylov subspace methods.

The pseudocode of the ILU factorization is given in Algorithm 1. The algorithm overwrites on A matrix so that U is stored in the upper triangular part including the diagonal and L is stored in the lower triangular part excluding the diagonal since diagonal of L is assumed to consist of ones. Here, $\mathcal{Z}$ denotes the allowable nonzero pattern of the resulting factors. In the case of ILU(0), $\mathcal{Z}$ is same as the nonzero pattern of the A matrix.

2.3 Sparse Triangular SPIKE (stSPIKE) Algorithm

We describe stSPIKE for lower triangular systems since the algorithm for the upper triangular case is similar. Given a lower triangular linear system of equations

$$Ly = b, \tag{2.8}$$

a DS factorization of sparse lower triangular matrix L is computed as $L = DS$, where D is the lower block diagonal of L and S is the Spike matrix. These blocks are assumed to be obtained by matrix partitioning. Multiplying both sides of (2.8) from the left by D^{-1}, we obtain a modified system

$$Sy = g, \tag{2.9}$$

where $g = D^{-1}b$ and $S = D^{-1}L$. By splitting $L = D + R$, we obtain $S = I + G$ where $G = D^{-1}R$, and R is the block off-diagonal part of L. The sparse triangular system $DG = R$ with multiple right hand side vectors can be solved for the block rows of G independently with perfect parallelism.

The nonzero column segments of R constitute dense column segments (called *spikes*) in the off-diagonal blocks of S. The block diagonal of S is identity. Additional nonzeros (fill-in) are introduced within the off-diagonal blocks of S only in the locations below the top nonzero (having the smallest row index) for each nonzero column segment of R. The submatrix consisting of rows and columns $\mathcal{C}$ of S, namely $\widehat{S}=S(\mathcal{C},\mathcal{C})$, constitutes an independent reduced system where $\mathcal{C}$ is the set of nonzero columns of R, i.e., linking columns of L. Then the reduced system is of the form

$$\widehat{S}\widehat{y} = \widehat{g}, \tag{2.10}$$

where $\widehat{g} = g(\mathcal{C})$ and $\widehat{y} = y(\mathcal{C})$, which can be solved independent from the rest of the unknowns in y. After solving the reduced system, the only remaining computation for retrieving the solution of the original system is

$$y = g - D^{-1}(\widehat{R}\widehat{y}), \tag{2.11}$$

which can be obtained in perfect parallelism where $\widehat{R}=R(:,\mathcal{C})$. We only partially compute S just to form $\widehat{S}$, since forming S explicitly is expensive and requires a large amount of memory.

The pseudocode of the parallel stSPIKE algorithm for lower triangular case is given in Algorithm 2. Partial computation of S constitutes the factorization phase (lines 2-5), whereas computation of $\widehat{g}$, solving (2.10) and (2.11) constitutes the solution phase (lines 6-13) of stSPIKE.

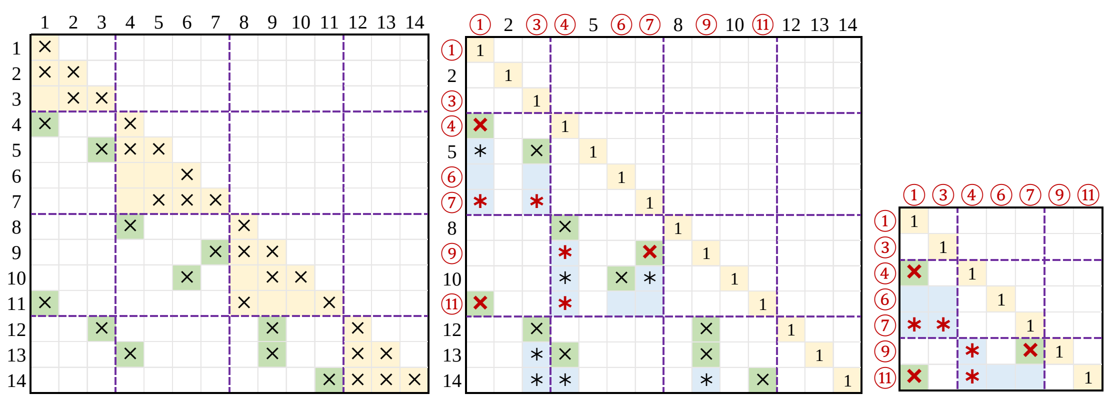
An example L matrix and the corresponding S and $\widehat{S}$ matrices are shown in Figure 2.1. The reduced system indices $\mathcal{C} = \{1, 3, 4, 6, 7, 9, 11\}$ are colored in red and circled. The nonzeros that constitute the reduced system are bold and colored in red. The background colors of the original nonzeros and possible fill-in are green and blue, respectively. Depending on the sparsity pattern of the corresponding column and block diagonal, spikes may not fill the entire column segment. For example, nonzero $L(4, 1)$ in block-(2,1) of L leads to the spike consisting of three nonzeros in the first column of block-(2,1) of S.

In the lower triangular stSPIKE algorithm, spikes occur in the lower part of the highest nonzero for each column segment in the blocks. Conversely for the

Algorithm 2 Sparse Triangular SPIKE (stSPIKE)

Require: Matrix $L = R + D$ and right hand side vector b

```
Choose an initial guess y
do in parallel :
    G ← D^{-1} R
    Ŝ ← Ĝ + I
Gather Ŝ matrix at processor P_1
do in parallel :
    g ← D^{-1} b
Gather g vector at processor P_1
ŷ ← Ŝ^{-1} ĝ                                ▷ solve reduced system
Scatter ŷ to processors
do in parallel :
    z ← R̂ ŷ
    y ← g − D^{-1} z
return x
```



(a) Matrix L (b) Matrix S (c) Matrix $\widehat{S}$

Figure 2.1: Sparsity structure of L and resulting S and $\widehat{S}$ matrices derived from stSPIKE.

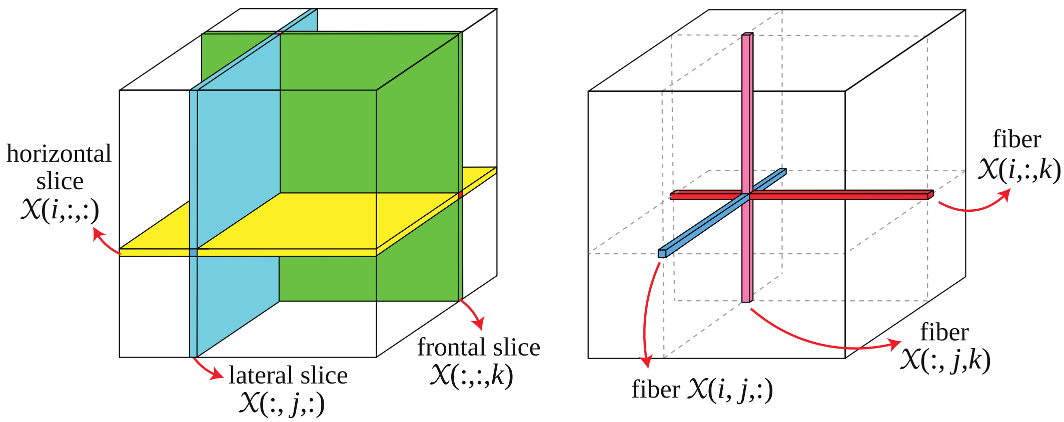


Figure 2.2: Sample slices and fibers of a tensor.

upper triangular stSPIKE, spikes occur in the upper part of the lowest nonzero for each column segment in the blocks.

2.4 Tensors

In the rest of this chapter, we denote tensors, matrices and vectors respectively by calligraphic ($\mathcal{X}$), bold capital ($\mathbf{A}$) and bold lowercase ($\mathbf{a}$) letters. To denote indices, we use lowercase letters ranging from 1 to their capital version, e.g., $q = 1, \ldots, Q$.

A tensor with M dimensions is called an M-mode tensor and mode m refers to the mth dimension. Unless specified, $\mathcal{X}$ is assumed to be a three-mode tensor of size $I \times J \times K$. The tensor element with indices i, j, k is denoted by $\mathcal{X}(i,j,k)$. Slices and fibers are defined as the subtensors obtained by holding one and two indices constant, respectively. $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ respectively denote the ith horizontal (mode-1), jth lateral (mode-2) and kth frontal (mode-3) slices. The intersection of two slices along different modes (e.g., $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,j,:)$) constitutes a fiber (e.g., $\mathcal{X}(i,j,:)$). Figure 2.2 illustrates slices $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ and fibers $\mathcal{X}(i,j,:)$, $\mathcal{X}(:,j,k)$ and $\mathcal{X}(i,:,k)$.

An M-mode tensor is called rank-one if it can be written as an outer product of M vectors. For instance, $(\mathbf{a} \circ \mathbf{b} \circ \mathbf{c})$ is a rank-one tensor. The matrix consisting of the mode-m fibers of a tensor $\mathcal{X}$ as its columns (in the increasing order of the other modes) is called the matricization of $\mathcal{X}$ in mode m, and is denoted by $\mathbf{X}_{(m)}$.

2.4.1 Matrix Kronecker, Khatri-Rao and Hadamard Products

Given an $I \times J$ matrix $\mathbf{A} = (a_{ij})$ and a $K \times L$ matrix $\mathbf{B} = (b_{kl})$, the Kronecker product of $\mathbf{A}$ and $\mathbf{B}$, which is of size $IK \times JL$, is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}.$$

Given an $I \times K$ matrix $\mathbf{A}$ and a $J \times K$ matrix $\mathbf{B}$, the Khatri-Rao product of $\mathbf{A}$ and $\mathbf{B}$, which is of size $IJ \times K$, is defined as

$$\mathbf{A} \odot \mathbf{B} = \begin{bmatrix} \mathbf{a}_1 \otimes \mathbf{b}_1 & \mathbf{a}_2 \otimes \mathbf{b}_2 & \cdots & \mathbf{a}_K \otimes \mathbf{b}_K \end{bmatrix},$$

where $\mathbf{a}_i$ and $\mathbf{b}_i$ respectively denote the ith columns of $\mathbf{A}$ and $\mathbf{B}$. Given two $I \times J$ matrices $\mathbf{A}$ and $\mathbf{B}$, the Hadamard product of $\mathbf{A}$ and $\mathbf{B}$, which is of size $I \times J$, is defined as

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix}.$$

2.4.2 Canonical Polyadic Decomposition (CPD)

CPD with F components factorizes a given tensor $\mathcal{X}$ as a sum of F rank-one tensors:

$$\mathcal{X} \approx \sum_{f=1}^{F} (\bar{\mathbf{a}}_f \circ \bar{\mathbf{b}}_f \circ \bar{\mathbf{c}}_f), \tag{2.12}$$

where $\bar{\mathbf{a}}_f$, $\bar{\mathbf{b}}_f$ and $\bar{\mathbf{c}}_f$ are column vectors of size I, J and K, respectively. Here F is very small with respect to the largest of the values I, J and K. Then, the factor matrices are defined as $\bar{\mathbf{A}} = [\bar{\mathbf{a}}_1 \ldots \bar{\mathbf{a}}_F]$, $\bar{\mathbf{B}} = [\bar{\mathbf{b}}_1 \ldots \bar{\mathbf{b}}_F]$ and $\bar{\mathbf{C}} = [\bar{\mathbf{c}}_1 \ldots \bar{\mathbf{c}}_F]$.

The columns of the factor matrices are stored as normalized to length one. That is, the norms $\lambda_f^A = \|\bar{\mathbf{a}}_f\|$, $\lambda_f^B = \|\bar{\mathbf{b}}_f\|$, $\lambda_f^C = \|\bar{\mathbf{c}}_f\|$ are computed and the columns are updated as $\mathbf{a}_f = \bar{\mathbf{a}}_f/\lambda_f^A$, $\mathbf{b}_f = \bar{\mathbf{b}}_f/\lambda_f^B$, $\mathbf{c}_f = \bar{\mathbf{c}}_f/\lambda_f^C$ for $f = 1, 2, ..., F$. Then, CPD of $\mathcal{X}$ is written as

$$\mathcal{X} \approx [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!] = \sum_{f=1}^{F} \lambda_f (\mathbf{a}_f \circ \mathbf{b}_f \circ \mathbf{c}_f), \tag{2.13}$$

where $\lambda = [\lambda_1 \ldots \lambda_F]$ and $\lambda_f = \lambda_f^A \lambda_f^B \lambda_f^C$.

CPD-ALS is an iterative algorithm whose pseudocode is given in Algorithm 3. At each iteration, it solves a linear least squares problem to find a factor matrix, by fixing the other two factor matrices. For example in order to find $\mathbf{A}$, CPD-ALS solves $min_{\mathbf{A}} ||\mathbf{X}_{(1)} - \mathbf{A}(\mathbf{C} \odot \mathbf{B})^T||_F^2$ for fixed $\mathbf{B}$ and $\mathbf{C}$ by computing

$$\mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}, \tag{2.14}$$

where $\odot$ and $*$ denote Khatri-Rao and Hadamard products, respectively. Here, $M = \mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B}$ is a small $F \times F$ dense matrix. In theory, finding the Moore-Penrose pseudoinverse [66] of M is sufficient [46]. The pseudoiverse is equal to the conventional matrix inverse when M is invertible, and it exists even when M is not invertible. Yet in practice, M is almost always symmetric positive-definite, and the Cholesky factorization can be applied to find its inverse [48]. At the beginning of the CPD-ALS algorithm, the $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ matrices are initialized randomly as recommended by several studies [46, 48, 67, 68].

In Algorithm 3, $\hat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$, $\hat{\mathbf{B}} = \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})$ and $\hat{\mathbf{C}} = \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})$ are the Matricized Tensor Times Khatri-Rao Product (MTTKRP) operations, which constitute the bottleneck operations of CPD-ALS due to large sizes of matrices involved. In MTTKRP operation $\hat{\mathbf{A}} = \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})$, each row $\hat{\mathbf{A}}(i,:)$ can be computed as

$$\hat{\mathbf{A}}(i,:) = \sum_{\mathcal{X}(i,j,k) \neq 0} \mathcal{X}(i,j,k)(\mathbf{B}(j,:) * \mathbf{C}(k,:)). \tag{2.15}$$

Algorithm 3 CPD-ALS($\mathcal{X}$)

1: Initialize matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ randomly
2: **while** not converged **do**
3: $\mathbf{A} \leftarrow \mathbf{X}_{(1)}(\mathbf{C} \odot \mathbf{B})(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$
4: Normalize columns of $\mathbf{A}$ into λ
5: $\mathbf{B} \leftarrow \mathbf{X}_{(2)}(\mathbf{C} \odot \mathbf{A})(\mathbf{C}^T\mathbf{C} * \mathbf{A}^T\mathbf{A})^{-1}$
6: Normalize columns of $\mathbf{B}$ into λ
7: $\mathbf{C} \leftarrow \mathbf{X}_{(3)}(\mathbf{B} \odot \mathbf{A})(\mathbf{B}^T\mathbf{B} * \mathbf{A}^T\mathbf{A})^{-1}$
8: Normalize columns of $\mathbf{C}$ into λ
9: **return** $[\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$

The computation of $\hat{\mathbf{A}}(i,:)$ only involves the nonzeros in slice $\mathcal{X}(i,:,:)$ and for each nonzero $\mathcal{X}(i,j,k)$ in that slice it requires rows $\mathbf{B}(j,:)$ and $\mathbf{C}(k,:)$.

2.4.3 Medium-Grain CPD-ALS Algorithm

The medium-grain CPD-ALS algorithm [48] is based on a 3D cartesian partition of a given tensor $\mathcal{X}$ for a virtual 3D mesh of $P = Q \times R \times S$ processors. In this partition, horizontal, lateral and frontal slices of $\mathcal{X}$ are partitioned among Q, R and S parts, respectively. These partitions are used for reordering the slices into Q horizontal, R lateral and S frontal chunks in such a way that the slices belonging to the same part are ordered consecutively (in any order) to form a chunk. The qth horizontal, rth lateral and sth frontal chunks are respectively denoted by $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$. The intersection of $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$ forms subtensor $\mathcal{X}_{q,r,s}$. Similarly, the qth horizontal, rth lateral and sth frontal layers of the virtual processor mesh are respectively denoted by $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$. Chunks $\mathcal{X}_{q,:,:}$, $\mathcal{X}_{:,r,:}$ and $\mathcal{X}_{:,:,s}$ are respectively distributed among the processors of layers $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$ in such a way that subtensor $\mathcal{X}_{q,r,s}$ is assigned to $p_{q,r,s}$.

A cartesian tensor partition induces a conformal partition of the rows of each factor matrix into chunks, e.g., $\mathbf{A}_1, \ldots, \mathbf{A}_Q$. The rows in the chunks $\mathbf{A}_q$, $\mathbf{B}_r$ and $\mathbf{C}_s$ are exclusively needed and updated by the processors in layers $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$, respectively. The factor-matrix rows owned by processor $p_{q,r,s}$ are assumed to be contiguous and denoted by $\mathbf{A}_{q,r,s}$, $\mathbf{B}_{q,r,s}$ and $\mathbf{C}_{q,r,s}$.

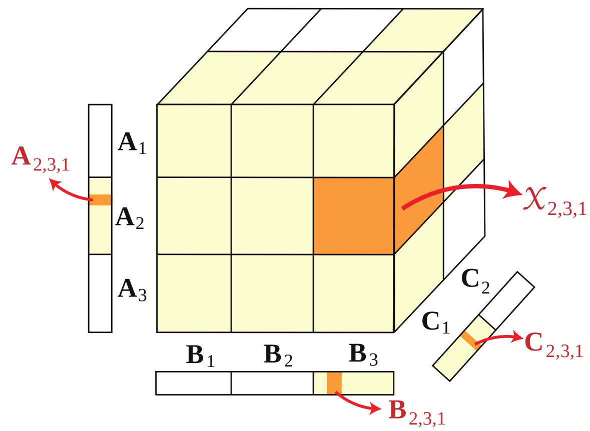


Figure 2.3: A medium-grain partition for a $3 \times 3 \times 2$ virtual mesh of processors.

Fig. 2.3 displays an example medium-grain partition with 3 horizontal, 3 lateral and 2 frontal chunks. Subtensor $\mathcal{X}_{2,3,1}$ as well as factor-matrix rows in $\mathbf{A}_{2,3,1}$, $\mathbf{B}_{2,3,1}$ and $\mathbf{C}_{2,3,1}$, which are all assigned to processor $p_{2,3,1}$, are highlighted with a darker shade. Note that $p_{2,3,1}$ may need to use the rest of the rows in $\mathbf{A}_2$, $\mathbf{B}_3$ and $\mathbf{C}_1$ during the MTTKRP operations.

The parallel medium-grain CPD-ALS algorithm consists of three phases at each iteration. The mth phase involves the computations and communications performed for computing the factor matrix along mode m. We only summarize the first phase since the other phases are similar. First, the MTTKRP operation is performed in a distributed fashion where each processor multiplies its nonzeros with the corresponding $\mathbf{B}$- and $\mathbf{C}$-matrix rows and produces partial results for the corresponding $\hat{\mathbf{A}}$-matrix rows as given in equation (2.15). Here, $\hat{\mathbf{A}}$ and $\mathbf{A}$ have conformal partitions.

After performing the local MTTKRP operation, each processor $p_{q,r,s}$ sends its partial results for non-local $\hat{\mathbf{A}}$-matrix rows to their owner processors, which reside in layer $p_{q,:,:}$. In a dual manner, $p_{q,r,s}$ receives the partial results for its local $\hat{\mathbf{A}}$-matrix rows ($\hat{\mathbf{A}}_{q,r,s}$) from the processors in the same layer and sums them to finalize $\hat{\mathbf{A}}_{q,r,s}$. We refer to this communication step as the *fold step*. Then, $p_{q,r,s}$ multiplies $\hat{\mathbf{A}}_{q,r,s}$ with $(\mathbf{C}^T\mathbf{C} * \mathbf{B}^T\mathbf{B})^{-1}$ and obtains $\mathbf{A}_{q,r,s}$. $\mathbf{A}$ is finalized by normalizing its columns using an all-to-all reduction on local norms. Then, $\mathbf{A}^T\mathbf{A}$ is obtained by another all-to-all reduction on locally computed $\mathbf{A}^T\mathbf{A}$ matrices.

Finally, each processor $p_{q,r,s}$ sends the updated rows in $\mathbf{A}_{q,r,s}$ to the processors that need these rows in the following two phases where $\mathbf{B}$ and $\mathbf{C}$ are computed. These processors are the ones that $p_{q,r,s}$ receives partial results from in the fold step. In a dual manner, $p_{q,r,s}$ receives the updated $\mathbf{A}$-matrix rows that it needs in the following two phases from their owner processors. These processors are the ones that $p_{q,r,s}$ sends partial results to in the fold step. We refer to this communication step as the *expand step*.

The communications in the fold and expand steps are confined to the processor layers. In the first, second and third phases, $p_{q,r,s}$ communicates with at most $R \times S - 1$, $Q \times S - 1$ and $Q \times R - 1$ processors residing in layers $p_{q,:,:}$, $p_{:,r,:}$ and $p_{:,:,s}$, respectively.

For testing the convergence, at the end of each iteration the residual is computed as

$$\sqrt{\langle \mathcal{X}, \mathcal{X} \rangle + \langle \mathcal{Y}, \mathcal{Y} \rangle - \langle \mathcal{X}, \mathcal{Y} \rangle}, \tag{2.16}$$

where $\mathcal{X}$ is the original tensor and $\mathcal{Y} = [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is its CPD approximation. Here, $\langle \mathcal{X}, \mathcal{X} \rangle$ is the sum of squares of the nonzero elements of $\mathcal{X}$, which can be computed once before the iterations. The norm of $\mathcal{Y} = [\![\lambda; \mathbf{A}, \mathbf{B}, \mathbf{C}]\!]$ is

$$\langle \mathcal{Y}, \mathcal{Y} \rangle = \lambda^T (\mathbf{A}^T \mathbf{A} * \mathbf{B}^T \mathbf{B} * \mathbf{C}^T \mathbf{C}) \lambda, \tag{2.17}$$

which is not costly since $\mathbf{A}^T\mathbf{A}$, $\mathbf{B}^T\mathbf{B}$, and $\mathbf{C}^T\mathbf{C}$ are already computed within the CPD-ALS iterations. Finally, the inner product $\langle \mathcal{X}, \mathcal{Y} \rangle$ is

$$\sum_{f=1}^{F} \lambda_f \left(\sum_{\mathcal{X}(i,j,k) \neq 0} \mathcal{X}(i,j,k) \mathbf{A}(i,f) \mathbf{B}(i,f) \mathbf{C}(i,f) \right), \tag{2.18}$$

which can be equivalently computed in practice as $\sum_{i=1}^{P} \mathbf{1}^T (\hat{A}_{p_i} * A_{p_i}) \lambda$, where A_{p_i} denotes the chunk of $\mathbf{A}$ owned by processor p_i and $\mathbf{1}$ is the vector consisting of ones.

Chapter 3

Related Work

3.1 Parallelization of Gauss-Seidel

In the literature, parallel GS implementations are proposed either to solve the original problem (1.1) [69, 70, 71] or to use it as a smoother in multigrid schemes [72, 73, 74]. A commonly-used method to parallelize GS by finding independent sub-tasks is the red-black coloring strategy [75, 76, 77], which has been extended to multi-coloring [78, 79, 80] to attain more parallelism for complicated regular problems. However, multi-colored GS is not feasible for some cases such as unstructured finite element applications since the number of colors becomes too large [70]. Another approach is to use a processor-localized GS in which each processor performs GS as a subdomain solver, but its convergence rate is low and may diverge for a large number of processors [21]. A distributed-memory parallel GS is proposed in [71] which partitions the coefficient matrix into row blocks, however its parallel scalability is poor due to load imbalance among processors. The main factor affecting the performance of parallel GS is the communication delay in the distributed-memory architectures [81].

3.2 Parallelization of ILU(0)

A common method to parallelize ILU(0) is to use the Red-Black colouring strategy whose degree of parallelism is $m/2$ where m is the number of unknowns [82]. However, the convergence of the ILU(0) preconditioner obtained with Red-Black coloring is observed to be poor [83]. The available parallelism is generally bounded by the level of dependent tasks and often very low for the original ordering. Especially for unsymmetric problems, matrix reordering techniques significantly improve the performance of the ILU-preconditioned Krylov subspace solvers [14]. [41] shows that reordering on unstructured grids improves the performance of GMRES using ILU(0) to solve the compressible Navier–Stokes equations.

ILU(0) is also useful as approximate subdomain solvers for domain decomposition-based preconditioners, e.g., Additive Schwarz Method (ASM) [84]. Studies indicate that the performance of ILU(0) may change depending on the problem types. The experimental results in [40] reveals that ILU(0) subdomain solver yields good scalability of ASM for Poisson's equation, but poor scalability for convection dominated problems.

In [85], ILU(0) is used as a preconditioner for the parallel GMRES algorithm and shown to achieve faster convergence than the baseline algorithms. It is demonstrated that using ILU(0) yields three times smaller number of GMRES iterations. Although the runtime of the proposed algorithm is shorter than the parallel program using PETSc, it does not scale for larger than 8 processors due to increasing communication overheads.

3.3 Parallelization of Tensor Decomposition

For sparse tensor computations, CPD is successfully utilized in a large variety of applications from different domains, such as chemometrics [86], telecommunications [87], medical imaging [88, 89], image compression and analysis [90], text mining [91, 92], knowledge bases [93] and recommendation systems [94]. Kolda and Bader [46] provide an extensive survey on tensor decomposition methods and their applications.

There are several distributed-memory CPD-ALS parallelization approaches for sparse tensors, varying on how they define and distribute atomic tasks. DFacTo [95] obtains a coarse-grain partition of the tensor by performing an independent one-dimensional block partitioning along each mode and is reported to be significantly faster than two earlier alternatives, Tensor Toolbox [96] and GigaTensor [97], when compared in a sequential setting. However, DFacTo is not memory scalable since it needs to store the matricized tensor along each mode as well as all factor matrices at each processor.

Kaya and Uçar [98] propose HP models that exploit the sparsity pattern of the tensor to minimize the total communication volumes of coarse- and fine-grain tensor partitionings. The coarse-grain HP model does not lead to a significant reduction in the total communication volume compared to block partitioning. This is due to the inherent limitation of coarse-grain partitioning, where each processor may need all factor-matrix rows in the non-partitioned modes. The fine-grain HP model overcomes this problem by distributing the tensor nonzeros individually, obtaining a multi-dimensional partition. The major drawback of the fine-grain model is the overhead of partitioning a large hypergraph containing vertices at least as many as the number of tensor nonzeros. The fine-grain HP model also suffers from inducing high number of messages, which is a consequence of disturbing the slice coherences.

To overcome these performance bottlenecks of coarse- and fine-grain models, Smith and Karypis [48] propose a successful medium-grain model which is based on multi-dimensional cartesian tensor partitioning. This cartesian tensor partitioning is also used by Austin et al. [99] for parallel Tucker decomposition.

Chapter 4

Partitioning and Reordering for Parallel Gauss-Seidel

We propose a distributed-memory parallel GS (dmpGS) by implementing and using a distributed-memory version of the stSPIKE algorithm. stSPIKE enables obtaining the solution of the system by solving independent sparse triangular subsystems and a smaller reduced triangular system. Solving this reduced system constitutes a sequential computational bottleneck in dmpGS. The size of this reduced system is equal to the number of nonzero columns in the lower off-diagonal blocks of the coefficient matrix. The computational cost of solving the reduced system is proportional to its nonzero count. The communication volume of dmpGS is equal to the number of nonzero column segments in the off-diagonal blocks plus the reduced system size. Both of these communication and computational overheads highly depend on the sparsity pattern of the coefficient matrix.

One way to alleviate the cost of solving the reduced system is to further parallelize the solution of the reduced system which has been done iteratively [28] or recursively [33, 29] in the context of the general banded and sparse Spike algorithms. Instead, we propose to minimize the size and the nonzero count of the reduced system, together with the communication volume, and show that the resulting reduced system is so small that further parallelization of the solution of the reduced system is often no longer needed. For minimizing the size and the nonzero count of the reduced system and the communication volume of dmpGS, we propose a partitioning and reordering model that exploits the sparsity of the coefficient matrix. The proposed model consists of two phases. The first phase is a row-wise partitioning of the coefficient matrix, whereas the second phase is a row reordering within the row blocks induced by the partition obtained in the first phase.

For the first phase, we propose a novel hypergraph model that extends and enhances the conventional column-net model for simultaneously decreasing the reduced system size and the communication volume. We introduce vertex fixing, net anchoring and net splitting schemes within the recursive bipartitioning framework to encode the minimization of the number of nonzero column segments in the lower triangular part of the resulting partition.

For the second phase, we propose an intelligent in-block row reordering method with the aim of decreasing the computational costs of both forming the coefficient matrix of the reduced system once and solving the reduced system at each iteration.

The rest of the chapter is organized as follows. In Section 4.1, we discuss the dmpGS algorithm along with its communication and computational costs. The proposed partitioning and reordering model for dmpGS is introduced in Section 4.2. We provide the experimental results in Section 4.3 and summarize in Section 4.4.

4.1 Distributed-Memory Parallel Gauss-Seidel (dmpGS)

The pseudo-code of dmpGS is given in Algorithm 4 for processor P_k in a K-processor system. Matrix A is assumed to be partitioned into K row blocks, where m_k denotes the number of rows in the k^{th} row block. In the algorithm, R_k,

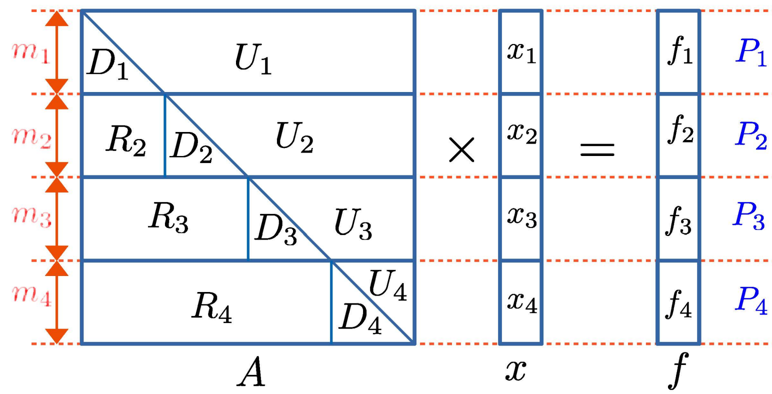


Figure 4.1: Four-way row-wise partition of matrix A and vectors x and f for dmpGS

D_k and U_k respectively denote the k^{th} row block of the strictly block lower triangular, lower triangular part of the block diagonal, and strictly upper triangular parts of A as shown in Figure 4.1. The number of columns in R_k, D_k and U_k are respectively $\sum_{i=1}^{k-1} m_i$, m_k and $\sum_{i=k}^{K} m_i$. f_k, g_k, x_k, h_k, w_k and z_k denote the local subvectors of size m_k that are computed by P_k. These subvectors are partitioned conformably with row-wise partitioning of A as shown in Figure 4.1. $\widehat{S}$, $\widehat{x}$ and $\widehat{g}$ respectively denote the $|\mathcal{C}| \times |\mathcal{C}|$ coefficient matrix, $|\mathcal{C}| \times 1$ unknown and $|\mathcal{C}| \times 1$ right hand side vectors of the reduced system in stSPIKE.

In Algorithm 4, lines 2-7 denote the factorization phase of stSPIKE which computes $\widehat{S}$. This phase is done only once after which we proceed with the GS iterations in lines 8-24. Each dmpGS iteration involves two SpMVs at lines 11 and 22, two vector subtraction operations at lines 12 and 24, an independent sparse triangular solve at line 13, and a reduced system solution at line 18, which enables independent sparse triangular solves at line 23. The upper and lower triangular SpMV operations are incurred by the GS and stSPIKE algorithms, respectively. These two SpMV operations incur communication of x-vector entries depending on the sparsity patterns of the upper triangular U and lower triangular L matrices, respectively. Conformable partitioning of the vectors avoids communication during vector subtraction operations.

At lines 9–10, communication operations are performed for local SpMV (line 11). After P_k receives all necessary non-local x-vector entries, it forms

Algorithm 4 Distributed-Memory Parallel Gauss Seidel (dmpGS) for processor P_k

Input: Submatrices R_k, D_k, U_k, and right-hand side subvector f_k

Output: Subvector x_k

Choose an initial guess for x_k
if $2 \leq k \leq K-1$ **then**
    $G_k \leftarrow D_k^{-1} R_k$ ▷ local partial sparse triangular solve with multiple RHS
    Form and send $\widehat{G}_k$ to processor P_1
if $k = 1$ **then**
    Receive $\widehat{G}_\ell$ from P_ℓ for $2 \leq \ell \leq K-1$ to form $\widehat{G}$
    $\widehat{S} \leftarrow \widehat{G} + I$
while not converged **do**
    Send required local x_k entries to respective processors in $\{P_1, \ldots, P_{k-1}\}$
    Receive non-local x_ℓ entries from processors in $\{P_{k+1}, \ldots, P_K\}$ to form $\breve{x}_k$
    $h_k \leftarrow U_k \breve{x}_k$ ▷ local SpMV
    $h_k \leftarrow f_k - h_k$
    $g_k \leftarrow D_k^{-1} h_k$ ▷ local sparse triangular solve
    if $2 \leq k \leq K-1$ **then**
        Send $\{g_k(i)\}_{i \in \mathcal{C}}$ to processor P_1
    if $k = 1$ **then**
        Receive $\{g_\ell(i)\}_{i \in \mathcal{C}}$ from P_ℓ for $2 \leq \ell \leq K-1$ to form $\widehat{g}$
        $\widehat{x} \leftarrow \widehat{S}^{-1} \widehat{g}$ ▷ solve reduced system
        Send $\widehat{x}$ entries to requiring processors
    if $k \neq 1$ **then**
        Receive required $\widehat{x}$-entries to form $\bar{x}_k$
    $z_k \leftarrow R_k \bar{x}_k$ ▷ local SpMV
    $w_k \leftarrow D_k^{-1} z_k$ ▷ local sparse triangular solve
    $x_k \leftarrow g_k - w_k$

its augmented vector $\breve{x}$. Each processor sends the selected entries of its g_k vector to P_1 (line 15) to form the right hand side vector $\widehat{g}$ (line 17) for the sequential solution of the reduced system to obtain $\widehat{x}$ (line 18). Here $\widehat{x}$ corresponds to those unknowns in x which are at the interface of the partitioning of L and obtaining them decouples the global lower triangular system into independent much smaller systems. P_1 sends only those x-vector entries that are required by other processors (line 19) so that each processor P_k forms its $\bar{x}$ vector (line 21) to perform local SpMV (line 22).

The communication overhead in each iteration of dmpGS is as follows. The communication volume incurred by $h = U\breve{x}$ (line 11) and $z = R\bar{x}$ (line 22) are equal to the number of nonzero column segments in the off-diagonal blocks of U and L, respectively. Thus the communication volume required by these two SpMVs is equal to the total number of off-diagonal nonzero column segments in A, which we refer to as $\texttt{offD\_nzCol\_seg}(A)$. The volume of communication incurred at line 17 is equal to the size of the reduced system, $|\mathcal{C}|$. Therefore, the total communication volume of dmpGS is

$$\texttt{commVol} = \texttt{offD\_nzCol\_seg}(A) + |\mathcal{C}|. \tag{4.1}$$

Note that the different row blocks (R_k) seem to vary in the number of columns because of the triangular structure of the problem. On the other hand, this disadvantage is alleviated by the proposed partitioning model which also gathers most of the nonzeros to the diagonal blocks.

4.2 The Proposed Partitioning and Reordering Model

We propose a two-phase model for reducing the communication overhead of dmpGS while maintaining computational balance as well as reducing the sequential computational overhead incurred by solving the reduced system at each iteration. This computational overhead is proportional to the number of nonzeros in the off-diagonals of $\widehat{S}$. In Section 4.2.1, we propose a novel HP model as the first phase which simultaneously encodes the minimization of the reduced system size $|\mathcal{C}|$ and the communication volume. Decreasing $|\mathcal{C}|$ is important not only because it directly contributes to reducing the communication volume, but it also relates to decreasing the computational overhead. In Section 4.2.2, we propose an in-block reordering method as the second phase which refines the improvement further by decreasing the number of nonzeros in $\widehat{S}$. We provide the illustrations showing the effect of the proposed partitioning model and the reordering method on a few real sparse matrices in Section 4.2.3.

4.2.1 Hypergraph Partitioning Model

The partitioning objective in this phase is minimizing the sum of communication volume overhead (4.1) and sequential overhead costs with proper scaling:

$$\begin{aligned} PartObj &= \texttt{commVol} + (\alpha - 1)|\mathcal{C}| \\ &= (\texttt{offD\_nzCol\_segs}(A) + |\mathcal{C}|) + (\alpha - 1)|\mathcal{C}| \\ &= \texttt{offD\_nzCol\_segs}(A) + \alpha|\mathcal{C}| \end{aligned} \tag{4.2}$$

Here α denotes the scaling factor between the effect of the reduced system size and the number of off-diagonal nonzero column segments on the overall overhead.

4.2.1.1 Definitions and Layout

We define a column as *L-linking* if it links at least one off-diagonal block in the lower triangular part. That is, a column c_i in k^{th} column block $\mathcal{B}_k^c$ is L-linking if it links a row block $\mathcal{B}_\ell^r$ with $\ell > k$. Since L-linking columns of A are the nonzero columns of R, the number of L-linking columns ($\texttt{L-link\_cols}$(A)) is equal to the reduced system size, $|\mathcal{C}|$. Therefore, the partitioning objective (4.2) can be rewritten as

$$PartObj = \texttt{offD\_nzCol\_segs}(A) + \alpha(\texttt{L-link\_cols}(A)). \tag{4.3}$$

Let $\mathcal{H}_{CN}(A) = (\mathcal{V}, \mathcal{N})$ be the column-net hypergraph of an $m \times m$ sparse matrix A with nonzero diagonal entries. An ordered partition $\Pi_K = \langle \mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K \rangle$ of $\mathcal{H}_{CN}(A)$ is decoded as a partial symmetric row and column reordering of A as explained in Section 2.1.2. Each net n_i of $\mathcal{H}_{CN}(A)$ connects vertex v_i since $A(i,i) \neq 0$ for each $1 \leq i \leq m$. A net n_i with $v_i \in \mathcal{V}_k$ is called *L-cut* if it connects at least one vertex part $\mathcal{V}_\ell$ such that $\ell > k$. The set of L-cut nets is denoted as $\mathcal{N}_{Lcut}$. We define a new type of cutsize, which we call the *L-cut-net metric*, as

$$cs_{Lcut}(\Pi_K) = \sum_{n \in \mathcal{N}_{Lcut}} c(n). \tag{4.4}$$

Finally, the cost of partition Π_K is defined as the sum of connectivity metric with unit net cost and L-cut-net metric with net cost α, i.e.,

$$cost_{conn+Lcut}(\Pi_K) = \sum_{n \in \mathcal{N}_{cut}} (\lambda(n) - 1) + \alpha|\mathcal{N}_{Lcut}|. \tag{4.5}$$

Here, each cut net n incurs $\lambda(n) - 1$, and each L-cut net incurs α to the cutsize.

Lemma 4.2.1. *A column c_i of A is L-linking if and only if net n_i of $\mathcal{H}_{CN}(A)$ is L-cut.*

Proof. Due to symmetric row-column ordering, c_i is in $\mathcal{B}_k^c$ if and only if r_i is in $\mathcal{B}_k^r$, which corresponds to $v_i \in \mathcal{V}_k$. Furthermore, c_i links $\mathcal{B}_\ell^r$ if and only if n_i connects $\mathcal{V}_\ell$. Therefore, c_i in $\mathcal{B}_k^c$ links $\mathcal{B}_\ell^r$ if and only if n_i with $v_i \in \mathcal{V}_k$ connects $\mathcal{V}_\ell$, where $\ell > k$. □

Proposition 4.2.2. *Minimizing $cost_{conn+Lcut}(\Pi_K)$ for a K-way partition Π_K of $\mathcal{H}_{CN}(A)$ corresponds to minimizing the partitioning objective (4.3).*

Proof. By Lemma 4.2.1, the number of L-cut nets in $\mathcal{H}_{CN}(A)$ is equal to the number of L-linking columns in A. Thus $\alpha|\mathcal{N}_{Lcut}| = \alpha(\texttt{L-link\_cols}(A))$. Furthermore, it is known by [55] that $\sum_{n \in \mathcal{N}_{cut}} (\lambda(n) - 1) = \texttt{offD\_nzCol\_segs}(A)$. □

Each vertex is associated with a weight equal to the number of nonzeros in the respective row of the matrix, i.e., $w(v_i) = nnz(A(i,:))$. Thus, the partitioning constraint of maintaining balance on part weights approximately encodes the computational load balance during aggregate two triangular SpMVs (lines 11 and 22) and two triangular solves (lines 13 and 23).

The cut-net splitting technique has been successfully used within the RB framework to encode the minimization of the connectivity metric [55]. However to the best of our knowledge, there exists no tool or model for encoding the minimization of the L-cut-net metric in the literature. We propose to use the RB framework with novel net anchoring and splitting schemes to encode the minimization of the L-cut-net metric.

4.2.1.2 Recursive Bipartitioning Model for dmpGS

At each RB step, an ordered bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ of $\mathcal{V}$ is decoded as ordering the vertices of $\mathcal{V}_L$ after those of $\mathcal{V}_U$. Here $\mathcal{V}_U$ and $\mathcal{V}_L$ denote the upper and lower vertex parts, respectively. In RB, the concept of L-cut net takes a special form. In a bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$, a net n_i is L-cut if v_i is assigned to $\mathcal{V}_U$ and n_i connects at least one vertex v_j such that $v_j \in \mathcal{V}_L$. The partitioning objective at each RB step is to minimize

$$cost_{RB}(\Pi_2) = |\mathcal{N}_{cut}| + \alpha|\mathcal{N}_{Lcut}|. \tag{4.6}$$

For encapsulating the connectivity and L-cut net metrics simultaneously, each net n_i in $\mathcal{H}_{CN}(A)$ is replicated as two different kinds of nets, namely *conn-net* n_i^c and *lcn-net* n_i^ℓ. Here, conn-nets encapsulate the connectivity metric whereas lcn-nets encapsulate the L-cut-net metric. The motivation for net replication is the requirement of different net splitting and net removal procedures for encoding the connectivity and L-cut-net metrics at each RB step. In order to encapsulate the RB objective (4.6), we assign unit cost to the conn-nets and cost α to the lcn-nets. We refer to the hypergraph formed by these replicated nets as $\mathcal{H}$.

We extend $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ into a hypergraph $\mathcal{H}' = (\mathcal{V}', \mathcal{N}')$ so that minimizing the number of conventional cut nets in $\mathcal{H}'$ encodes minimizing (4.6). We introduce new fixed vertices $v_U \in \mathcal{V}_U$ and $v_L \in \mathcal{V}_L$ to form the extended vertex set $\mathcal{V}' =$

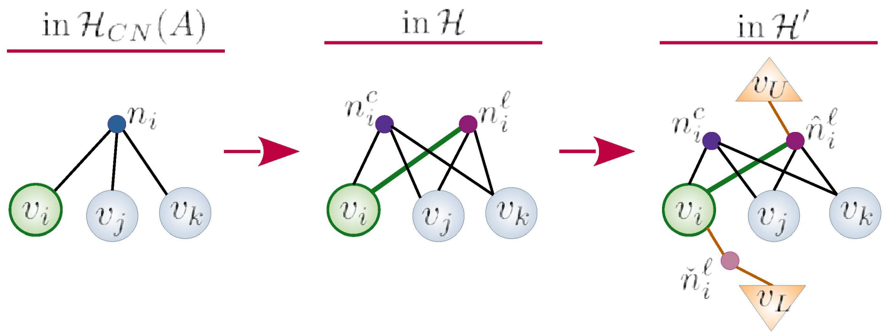


Figure 4.2: Net n_i in $\mathcal{H}_{CN}(A)$ is replicated as conn-net n_i^c and lcn-net n_i^ℓ to form $\mathcal{H}$. Net n_i^ℓ in $\mathcal{H}$ is represented by a pair of nets $\hat{n}_i^\ell$ and $\check{n}_i^\ell$ in $\mathcal{H}'$.

$\mathcal{V} \cup \{v_U, v_L\}$. We represent each lcn-net n_i^ℓ in $\mathcal{H}$ as a pair of nets $\hat{n}_i^\ell$ and $\check{n}_i^\ell$ in $\mathcal{H}'$. $\hat{n}_i^\ell$ is same as n_i^ℓ except it is U-anchored (connects v_U). $\check{n}_i^\ell$ is a 2-pin L-anchored net which connects v_L and v_i. That is, for each net n_i in $\mathcal{H}_{CN}(A)$, $\mathcal{H}'$ contains nets n_i^c, $\hat{n}_i^\ell$ and $\check{n}_i^\ell$, where

$$Pins(n_i^c, \mathcal{H}') = Pins(n_i, \mathcal{H}_{CN}(A)),$$
$$Pins(\hat{n}_i^\ell, \mathcal{H}') = Pins(n_i, \mathcal{H}_{CN}(A)) \cup \{v_U\} \text{ and}$$
$$Pins(\check{n}_i^\ell, \mathcal{H}') = \{v_i, v_L\}.$$

The nets in the extended hypergraph for a sample 3-pin net are shown in Figure 4.2.

We form $\mathcal{H}'$ at the beginning and apply RB steps until reaching the desired part count, K. The resulting K-way partition Π'_K of $\mathcal{H}'$ induces a K-way partition Π_K of $\mathcal{H}_{CN}(A)$. $\mathcal{H}$ is an in-between hypergraph introduced for the sake of clarity of presentation and is not constructed during implementation. We explain the proposed net splitting and removal methods on $\mathcal{H}$, and show the correspondence on $\mathcal{H}'$. We consider that each bipartition $\Pi'_2 = \langle \mathcal{V}'_U, \mathcal{V}'_L \rangle$ of $\mathcal{H}'$ induces a bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ of $\mathcal{H}$. Here $\mathcal{H}$ and $\mathcal{H}'$ refer to the respective hypergraphs just before the current RB step. New hypergraphs $\mathcal{H}_U$ and $\mathcal{H}_L$ are constructed according to $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ as follows. For both conn- and lcn-nets, each internal net in $\mathcal{V}_L$ and $\mathcal{V}_U$ is respectively included in $\mathcal{N}_L$ and $\mathcal{N}_U$ as is. In the net splittings, a new conn- or lcn-net is added to the net list of $\mathcal{H}_U$ or $\mathcal{H}_L$ only if it has more than one pin. The single-pin nets are discarded since they cannot contribute to the cutsize in the following RB steps.

For cut conn-nets, we apply the conventional cut-net splitting procedure [55] to encapsulate the connectivity metric. If a conn-net n_i^c is cut, then n_i^c is split into two pin-wise disjoint nets in $\mathcal{H}_U$ and $\mathcal{H}_L$ such that

$$Pins(n_i^c, \mathcal{H}_U) = Pins(n_i^c, \mathcal{H}) \cap \mathcal{V}_U, \tag{4.7}$$

$$Pins(n_i^c, \mathcal{H}_L) = Pins(n_i^c, \mathcal{H}) \cap \mathcal{V}_L. \tag{4.8}$$

For lcn-nets, we introduce a hybrid cut-net splitting/removal method in order to correctly encapsulate the L-cut-net metric. At each RB step, for each net pair $(\hat{n}_i^\ell, \check{n}_i^\ell)$ in a bipartition Π', we consider the state of n_i^ℓ in Π where $Pins(n_i^\ell, \mathcal{H}) = Pins(\hat{n}_i^\ell, \mathcal{H}') - \{v_U\}$ for ease of understanding. If an lcn-net n_i^ℓ is not internal, then it can be L-cut or "cut but not L-cut".

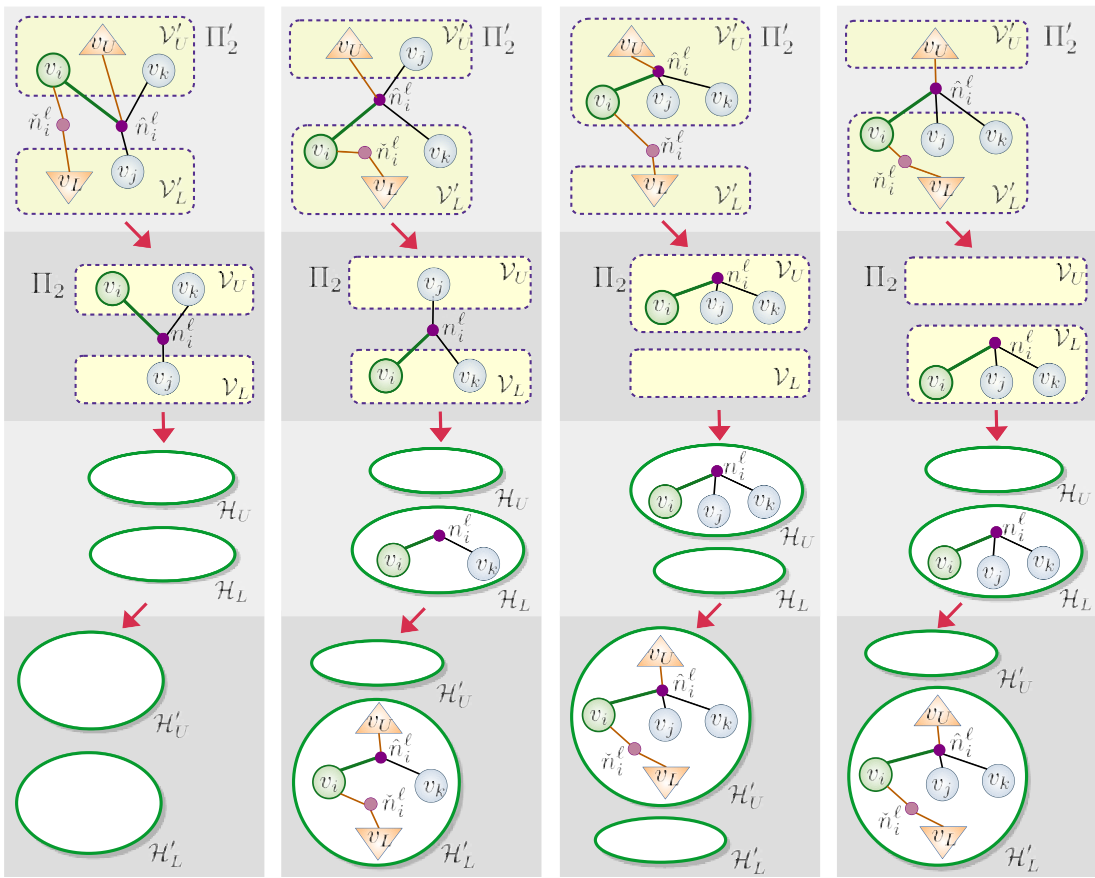
If n_i^ℓ is L-cut, then we apply cut-net removal for n_i. This is because when n_i is L-cut in an RB step, it also becomes L-cut in the final K-way partition. Hence there is no need to track this net anymore and we do not include it in further bipartitions.

If n_i^ℓ is cut but not L-cut, then we apply net removal towards $\mathcal{H}_U$ and *net-L-splitting* towards $\mathcal{H}_L$. That is, n_i^ℓ is added to $\mathcal{H}_L$ as $Pins(n_i^\ell, \mathcal{H}_L) = Pins(n_i^\ell, \mathcal{H}) \cap \mathcal{V}_L$. This is because n_i^ℓ cannot be L-cut in further bipartitionings of $\mathcal{H}_U$ but it has the potential of becoming L-cut in further bipartitionings of $\mathcal{H}_L$. In the extended hypergraph context, this corresponds to adding lcn-net pair $(\hat{n}_i^\ell, \check{n}_i^\ell)$ to $\mathcal{H}'_L$ such that

$$Pins(\hat{n}_i^\ell, \mathcal{H}'_L) = (Pins(n_i^\ell, \mathcal{H}') \cap \mathcal{V}'_L) \cup \{v_U\}, \tag{4.9}$$

$$Pins(\check{n}_i^\ell, \mathcal{H}'_L) = \{v_i, v_L\}. \tag{4.10}$$

Figure 4.3 shows all possible cases for a sample lcn-net. The first, second, third and last horizontal layers respectively show the bipartition Π'_2 of $\mathcal{H}'$; the corresponding bipartition Π_2 of $\mathcal{H}$; $\mathcal{H}_U$ and $\mathcal{H}_L$ induced by Π_2; and the corresponding $\mathcal{H}'_U$ and $\mathcal{H}'_L$ induced by Π'_2. If n_i^ℓ is L-cut in $\mathcal{H}$ as in Figure 4.3a, both $\hat{n}_i^\ell$ and $\check{n}_i^\ell$ are cut in Π'_2. If n_i^ℓ is cut but not L-cut as in Figure 4.3b, or if n_i^ℓ is internal to $\mathcal{V}_L$ as in Figure 4.3d, then only $\hat{n}_i^\ell$ is cut. Otherwise, if n_i^ℓ is internal to $\mathcal{V}_U$ as in Figure 4.3c, then only $\check{n}_i^\ell$ is cut.



(a) n_i^ℓ is L-cut (b) n_i^ℓ is cut, but not L-cut (c) n_i^ℓ is internal to part $\mathcal{V}_U$ (d) n_i^ℓ is internal to part $\mathcal{V}_L$

Figure 4.3: All cases for an lcn-net n_i^ℓ and the corresponding net pair $(\hat{n}_i^\ell, \check{n}_i^\ell)$ after bipartition $\langle \mathcal{V}'_U, \mathcal{V}'_L \rangle$.

Lemma 4.2.3. *Consider the bipartition* $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ *of* $\mathcal{H}$ *induced by a bipartition* $\Pi'_2 = \langle \mathcal{V}'_U, \mathcal{V}'_L \rangle$ *of* $\mathcal{H}'$ *in an RB step. If a net is L-cut in* Π_2*, then it incurs 2 cut nets in* Π'_2*. Conversely, if a net is not L-cut in* Π_2*, then it incurs 1 cut net in* Π'_2*.*

Proof. If n_i^ℓ is L-cut in Π_2 of $\mathcal{H}$, then $v_i \in \mathcal{V}_U$ and n_i^ℓ connects a vertex v_j such that $v_j \in \mathcal{V}_L$. In Π'_2 of $\mathcal{H}'$, $\hat{n}_i^\ell$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_j \in \mathcal{V}'_L$; and $\check{n}_i^\ell$ is also cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$.

If n_i^ℓ is not L-cut and $v_i \in \mathcal{V}_L$ in Π_2, then $\hat{n}_i^\ell$ is cut in Π'_2 because it connects $v_U \in \mathcal{V}'_U$ and $v_i \in \mathcal{V}'_L$; but $\check{n}_i^\ell$ is not cut since both v_i and v_L are in $\mathcal{V}'_L$.

If n_i^ℓ is not L-cut and $v_i \in \mathcal{V}_U$ in Π_2, then n_i^ℓ should be internal to $\mathcal{V}_U$, because otherwise any pin in $\mathcal{V}_L$ would make n_i^ℓ to be L-cut. In Π'_2, net $\hat{n}_i^\ell$ is internal to $\mathcal{V}'_U$ since both v_i and v_U are in $\mathcal{V}'_U$; but $\check{n}_i^\ell$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$. □

Proposition 4.2.4. *Minimizing the conventional cut-net metric for the bipartition* Π'_2 *of* $\mathcal{H}'$ *encodes minimizing* $cost_{RB}(\Pi_2)$ *defined in (4.6).*

Proof. By Lemma 4.2.3, each L-cut net in Π_2 incurs 2 cut nets in Π'_2, whereas all remaining nets in Π_2 incur 1 cut net in Π'_2. Since the cost of lcn-nets is α, the cutsize incurred by lcn-nets in Π'_2 is $\alpha(|\mathcal{N}_{Lcut}|+|\mathcal{N}|)$. Since conn-nets are of unit cost, they incur $|\mathcal{N}_{cut}|$ to the cutsize of Π'_2. Hence the total cutsize of Π'_2 is $|\mathcal{N}_{cut}|+\alpha|\mathcal{N}_{Lcut}|+\alpha|\mathcal{N}|$. Since $\alpha|\mathcal{N}|$ is constant, minimizing the cutsize of Π'_2 is equivalent to minimizing $|\mathcal{N}_{cut}|+\alpha|\mathcal{N}_{Lcut}|$, which is $cost_{RB}(\Pi_2)$. □

Figure 4.4 shows an example 2-level RB in terms of lcn-nets in $\mathcal{H}$ and the corresponding 4-way matrix partitioning. The L-cut nets n_1^ℓ, n_2^ℓ and n_6^ℓ and the corresponding L-linking columns c_1, c_2, and c_6 of A are colored in red background. n_2^ℓ is L-cut in the first level RB and discarded in the future bipartitions. This is because column c_2 is already counted as L-linking due to nonzero $A(6,2)$ and should not be counted as L-linking again due to nonzero $A(4,2)$ in further bipartitions.

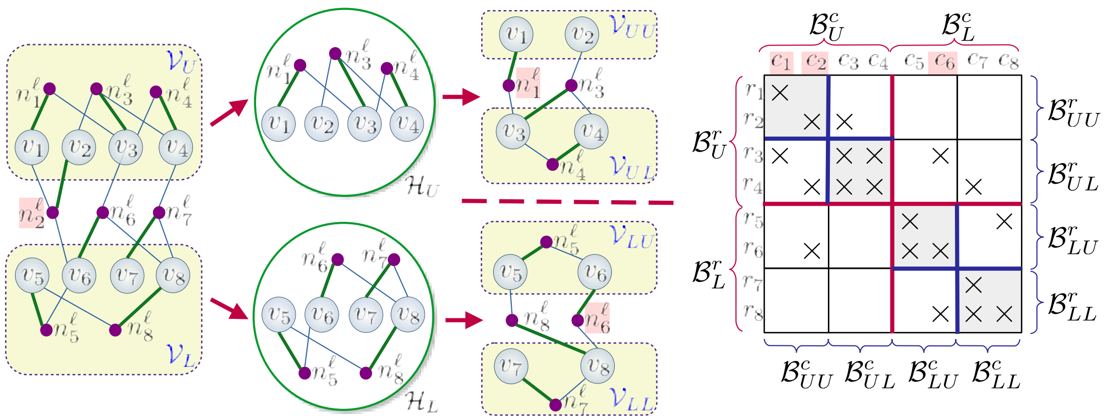


Figure 4.4: Sample 2-level RB showing lcn-nets and corresponding matrix partitioning.

Note that the L-cut net definition can be considered to be similar to the left-cut net defined in [100] for encapsulating the profile minimization, but the net splitting and removal strategies are quite different for encapsulating the objective of our problem.

Theorem 4.2.5. *Recursively bipartitioning $\mathcal{H}'$ by minimizing the cutsize according to the cut-net metric and applying the proposed net splitting and removal strategies until reaching K parts encodes minimizing the partitioning objective (4.3).*

Proof. By Proposition 4.2.4, recursively bipartitioning $\mathcal{H}'$ by minimizing the conventional cut-net metric encodes minimizing $cost_{RB}(\Pi_2)$ at each RB step. We show that this encodes minimizing $cost_{conn+Lcut}(\Pi_K)$. Proposed net splitting and removal strategies ensure that an L-cut net in Π_K is also L-cut in Π_2 in exactly one RB step. Since an L-cut net contributes α to both $cost_{RB}(\Pi_2)$ and $cost_{conn+Lcut}(\Pi_K)$, minimizing $\alpha|\mathcal{N}_{Lcut}|$ in each bipartition Π_2 encodes minimizing $\alpha|\mathcal{N}_{Lcut}|$ in Π_K. Furthermore, minimizing the number of cut nets $|\mathcal{N}_{cut}|$ at each RB step and applying the cut-net splitting procedure encodes minimizing the connectivity metric $\sum_{n\in\mathcal{N}_{cut}}(\lambda(n)-1)$ [55]. Therefore, minimizing the cutsize for each bipartition Π_2' of $\mathcal{H}'$ encodes minimizing $cost_{conn+Lcut}(\Pi_K)$; hence by Proposition 4.2.2, this corresponds to the partitioning objective (4.3). □

4.2.2 Reordering within Row Blocks

Consider the K-way block structure of A (as in Figure 4.1) induced by the partial symmetric row-column permutation obtained by the HP model (Section 4.2.1). We perform row reordering within the k^{th} row block of A by considering nonzeros of the k^{th} row block R_k of R. The resulting row reordering within the k^{th} row block of A is symmetrically applied to the columns of the k^{th} column block of A. R_k is an $m_k \times z_k$ matrix where $z_k = \sum_{i=1}^{k-1} m_i$. For simplicity, we assume a local indexing for the rows of R_k so that R_k consists of rows r_i with $1 \leq i \leq m_k$. Let $\mathcal{C}_k$ be the subset of $\mathcal{C}$ corresponding to the row indices in R_k.

Recall that in stSPIKE, fill-in may arise below the top nonzero of each spike in R_k. The top nonzero of a spike c_j in R_k is the nonzero with row index

$$\mathit{top}(c_j, R_k) = \min\{i : R_k(i,j) \neq 0,\ 1 \leq i \leq m_k\}. \tag{4.11}$$

We define the *height* of a spike c_j in R_k as the number of reduced system row indices between $\mathit{top}(c_j, R_k)$ and m_k inclusively, i.e.,

$$\text{height}(c_j, R_k) = |\{i : \mathit{top}(c_j, R_k) \leq i \leq m_k,\ i \in \mathcal{C}_k\}|, \tag{4.12}$$

since only the rows with indices in $\mathcal{C}_k$ may contribute to the nonzero count of $\widehat{S}$. The height of a spike in R_k constitutes an upper bound on the nonzero count (including the fill-in) of the corresponding column in $\widehat{S}$. In Figure 2.1b, the heights of the spikes are as follows: height(c_1, R_2)=3, height(c_3, R_2)=2; height(c_1, R_3)=1, height(c_4, R_3)=2, height(c_6, R_3)=1, height(c_7, R_3)=2. The height of a non-spike column is assumed to be zero. The objective of in-block reordering is to minimize the *total height*

$$\sum_{k=2}^{K-1} \sum_{j=1}^{z_k} \mathit{height}(c_j, R_k), \tag{4.13}$$

which constitutes an upper bound on the nonzero count in off-diagonal blocks of $\widehat{S}$. The last block R_K does not contribute nonzeros to $\widehat{S}$ since $\mathcal{C}_K$ is empty. Reordering within different blocks are completely independent and can be done concurrently.

One straightforward approach is placing the rows whose indices are not among $\mathcal{C}_k$ to the bottom of R_k to avoid the nonzeros of the rows that are not in $\mathcal{C}_k$ to contribute to (4.13). Let $\overline{R}_k = R_k(\mathcal{C}_k, :)$ be the $|\mathcal{C}_k| \times z_k$ submatrix of R_k consisting of the rows with indices in $\mathcal{C}_k$. Then the problem is reduced to reorder only those rows of $\overline{R}_k$ since the rest of the rows on the bottom of R_k do not have an impact on (4.13).

The reordering objective for each $\overline{R}_k$ is to minimize $\sum_{j=1}^{z_k} height(c_j, \overline{R}_k)$, with a simplified height definition, $height(c_j, \overline{R}_k) = |\mathcal{C}_k| + 1 - top(c_j, \overline{R}_k)$. This is equivalent to maximize $\sum_{j=1}^{z_k} top(c_j, \overline{R}_k)$, since $\sum_{j=1}^{z_k}(|\mathcal{C}_k|+1)$ is constant. Notice here the resemblance of this problem with the profile minimization problem [101, 102], which is known to be NP-Hard [103, 104]. The objective of profile minimization is symmetrically reordering a symmetric matrix $T \in \mathbb{R}^{n \times n}$ with nonzero diagonal entries to minimize $\sum_{j=1}^{n}(j - top(c_j, T))$, or equivalently to maximize $\sum_{j=1}^{n} top(c_j, T)$, since $\sum_{j=1}^{n} j$ is constant. Since the column reordering itself has no effect on the sum $\sum_{j=1}^{n} top(c_j, T)$, profile minimization is a special case of our problem for symmetric matrices assuming a symmetric row-column reordering. Therefore, the profile minimization problem can be reduced to the total height minimization problem in polynomial time; and could be solved in polynomial time if there had been a polynomial-time solution to the total height minimization problem. But since the profile minimization is NP-Hard, then so is the total height minimization problem.

We propose a heuristic for minimization of the total height whose pseudocode is presented in Algorithm 5. Its efficient implementation requires accessing the nonzeros of both rows and columns of $\overline{R}_k$, so it is stored both in CSR (Compressed Sparse Row) and CSC (Compressed Sparse Column) formats. $Cols(r_i)$ denotes the set of columns in row r_i, whereas $Rows(c_j)$ denotes the set of rows in column c_j. Degree of a row or column is defined as the number of nonzeros in that row or column, i.e., $deg(r_i) = |Cols(r_i)|$ and $deg(c_j) = |Rows(c_j)|$. In lines 3-6, we define the *load* of each row r_i as the sum of degrees of columns c_j such that $\overline{R}_k(i, j) \neq 0$.

The greedy choice utilized in the proposed heuristic is to order the rows with

Algorithm 5 Proposed in-block reordering for R_k where $2 \leq k \leq K-1$

Input: $R_k \in \mathbb{R}^{m_k \times z_k}$ and set of reduced-system row indices $\mathcal{C}_k$ of R_k.
Output: the permutation vector *perm* of R_k.

Place the rows r_i with $i \notin \mathcal{C}_k$ to the last $m_k - |\mathcal{C}_k|$ indices in any order
Consider submatrix $\overline{R}_k = R_k(\mathcal{C}_k, :)$ of R_k consisting of rows r_i with $i \in \mathcal{C}_k$
for each row r_i of $\overline{R}_k$ **do**
  $load(r_i) \leftarrow 0$
  for each column $c_j \in Cols(r_i)$ **do**
    $load(r_i) \leftarrow load(r_i) + deg(c_j)$
for $d \leftarrow 0$ **to** max_row_deg **do**
  $\mathcal{S}(d) \leftarrow \{r_i : deg(r_i) = d\}$
$indx \leftarrow 0$
while $indx < |\mathcal{C}_k|$ **do**
  $d^* \leftarrow \min\{d : \mathcal{S}(d) \neq \varnothing\}$
  $r_{i^*} \leftarrow \text{argmax}_{r_i \in \mathcal{S}(d^*)}\, load(r_i)$ ▷ Select $r_{i^*} \in \mathcal{S}(d^*)$ with maximum load
  $indx \leftarrow indx + 1$
  $perm(indx) \leftarrow r_{i^*}$
  $\mathcal{S}(d^*) \leftarrow \mathcal{S}(d^*) - \{r_{i^*}\}$
  for each column $c_j \in Cols(r_{i^*})$ **do**
    $Rows(c_j) \leftarrow Rows(c_j) - \{r_{i^*}\}$
    $Cols(r_{i^*}) \leftarrow Cols(r_{i^*}) - \{c_j\}$
    for each row $r_{i'} \in Rows(c_j)$ **do**
      $Cols(r_{i'}) \leftarrow Cols(r_{i'}) - \{c_j\}$
      $load(r_{i'}) \leftarrow load(r_{i'}) - deg(c_j)$
      $\mathcal{S}(deg(r_{i'})) \leftarrow \mathcal{S}(deg(r_{i'})) - \{r_{i'}\}$
      $deg(r_{i'}) \leftarrow deg(r_{i'}) - 1$
      $\mathcal{S}(deg(r_{i'})) \leftarrow \mathcal{S}(deg(r_{i'})) \cup \{r_{i'}\}$

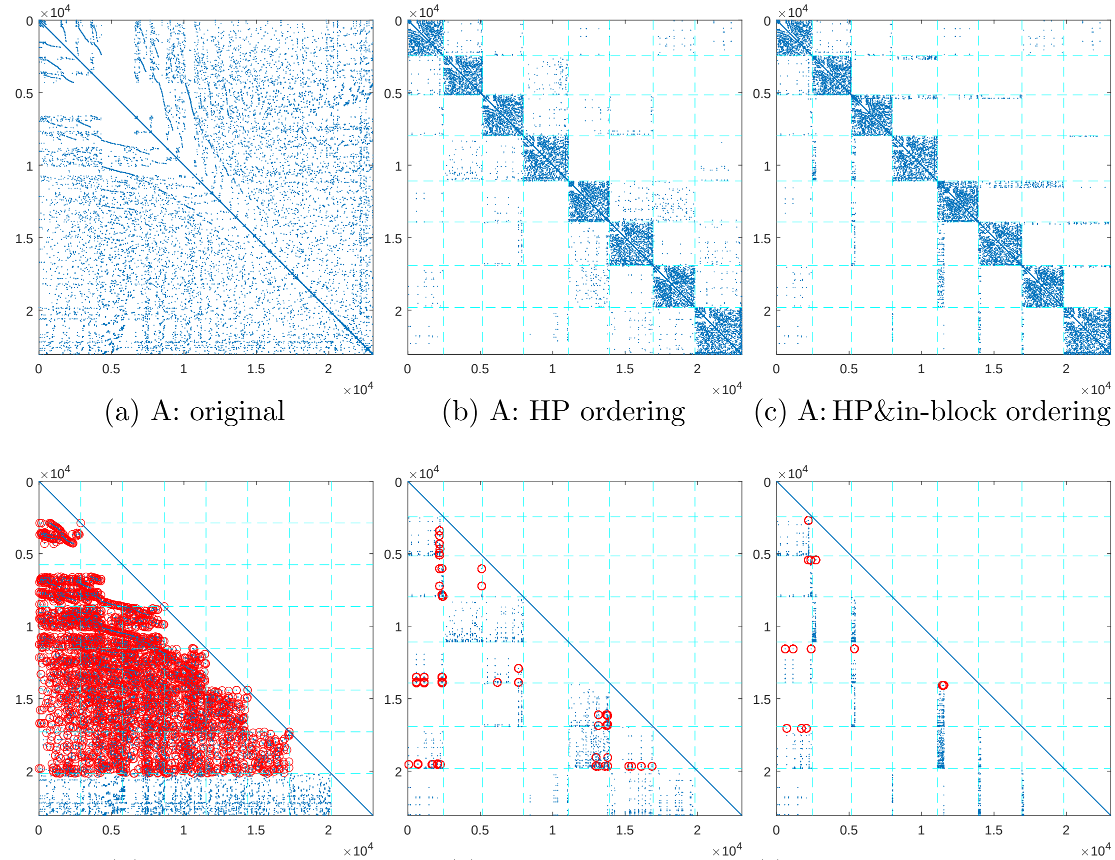
smaller degrees to upper positions of $\overline{R}_k$ since placing denser rows to upper positions incurs more height in (4.13). We further improve our greedy approach by using dynamic row degrees during the row selection process. When a row is selected, the degree of each unselected row is decremented by the number of its nonzeros having the same column index with the nonzeros in the selected row. Since the nonzeros in a selected row already determine the heights of the respective columns, we do not need to consider the rest of the nonzeros of these columns in future row selections. When selecting a row among rows with the same degree, load values of the rows are used as a tie-breaking strategy. A row with a higher load is preferred to be selected since it will lead to a larger amount of decrease on the degrees of unselected rows.

In Algorithm 5, $\mathcal{S}(d)$ denotes the set of rows with degree d. Due to dynamic row degrees, at each iteration we find the minimum degree d^* (line 11). Then we choose the row r_{i^*} in $\mathcal{S}(d^*)$ with the maximum load (line 12). After r_{i^*} is selected, all remaining nonzeros in each column c_j with $R_k(i^*, j) \neq 0$ are deleted as in lines 17-20. For each unselected row $r_{i'}$ with $R_k(i', j) \neq 0$, we dynamically update the load and degree of $r_{i'}$, and the respective degree sets (lines 21-24).

Recall that forming $\widehat{S}$ in dmpGS requires the computation of nonzeros up to the largest reduced system row index and any entry beyond that is not required to be computed for each row block. Hence the total height (4.13) also gives the computational cost of forming $\widehat{S}$ since we place $\overline{R}_k$ at the top of R_k for each $1 < k < K$.

4.2.3 Illustration

We provide the illustrations showing the effect of the proposed partitioning model and the reordering method on three real sparse matrices from the SuiteSparse Matrix Collection [105]. Figures 4.5, 4.6 and 4.7 illustrate the effect of applying the proposed partitioning and reordering model for K=8 on the matrices `msc23052`, `ACTIVSg10K`, and `mult_dcop_01` respectively. The nonzero pattern of the original matrix, the pattern obtained after applying the proposed HP model and the final



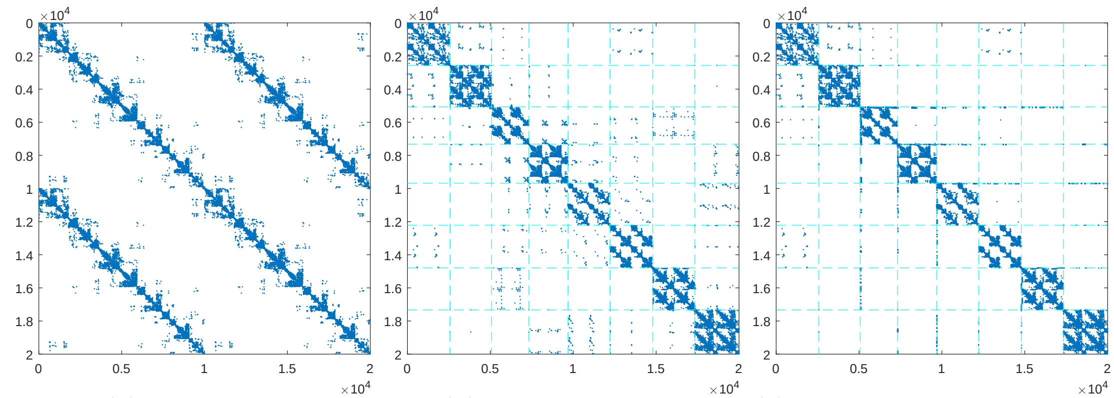
(a) A: original (b) A: HP ordering (c) A: HP&in-block ordering

(d) S: original $\text{nnz}(\widehat{S}-I) = 277{,}113$ (e) S: HP ordering $\text{nnz}(\widehat{S}-I) = 3{,}593$ (f) S: HP&in-block ordering $\text{nnz}(\widehat{S}-I) = 811$

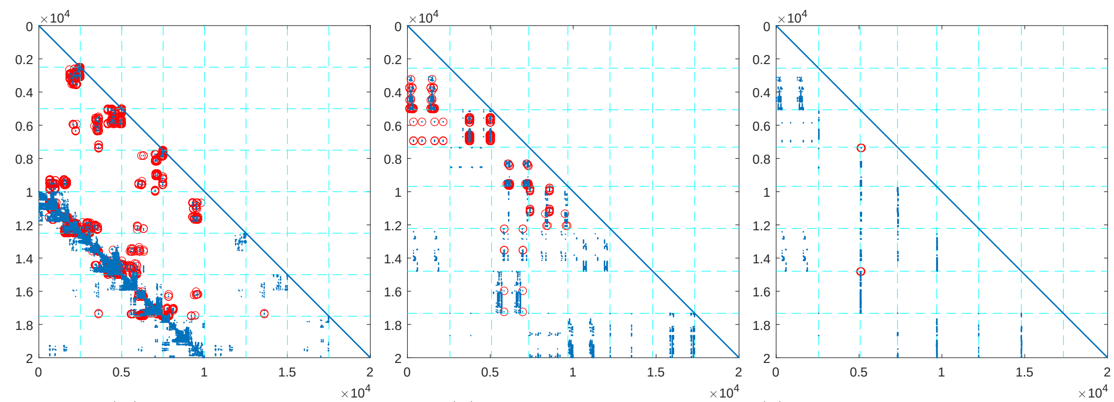
Figure 4.5: Nonzero pattern of `msc23052`: (a) before ordering, (b) after HP for $K = 8$, (c) after HP and in-block reordering; (d),(e),(f) the respective Spike (S) matrices (the reduced system ($\widehat{S}$) nonzeros are circled in red color).

pattern after the proposed in-block reordering are shown in order. Below each ordering of A, the resulting Spike matrix (S) is shown, including the nonzeros of the reduced system ($\widehat{S}$) which are highlighted with red circles. As seen in the figure, the proposed partitioning and reordering model significantly reduces the nonzero count of the reduced system. For example, the number of nonzeros in $\widehat{S}-I$ in Figures 4.5d, 4.5e, and 4.5f are 277,113, 3,593, and 811, respectively. Note that these numbers may seem to be much larger than the ones appearing in the figures because of the overlapping red circles.

Notice that the proposed HP model gathers most of the nonzeros to the diagonal blocks so that the off-diagonal blocks become very sparse. Then, the proposed



(a) A: original (b) A: HP ordering (c) A: HP&in-block ordering

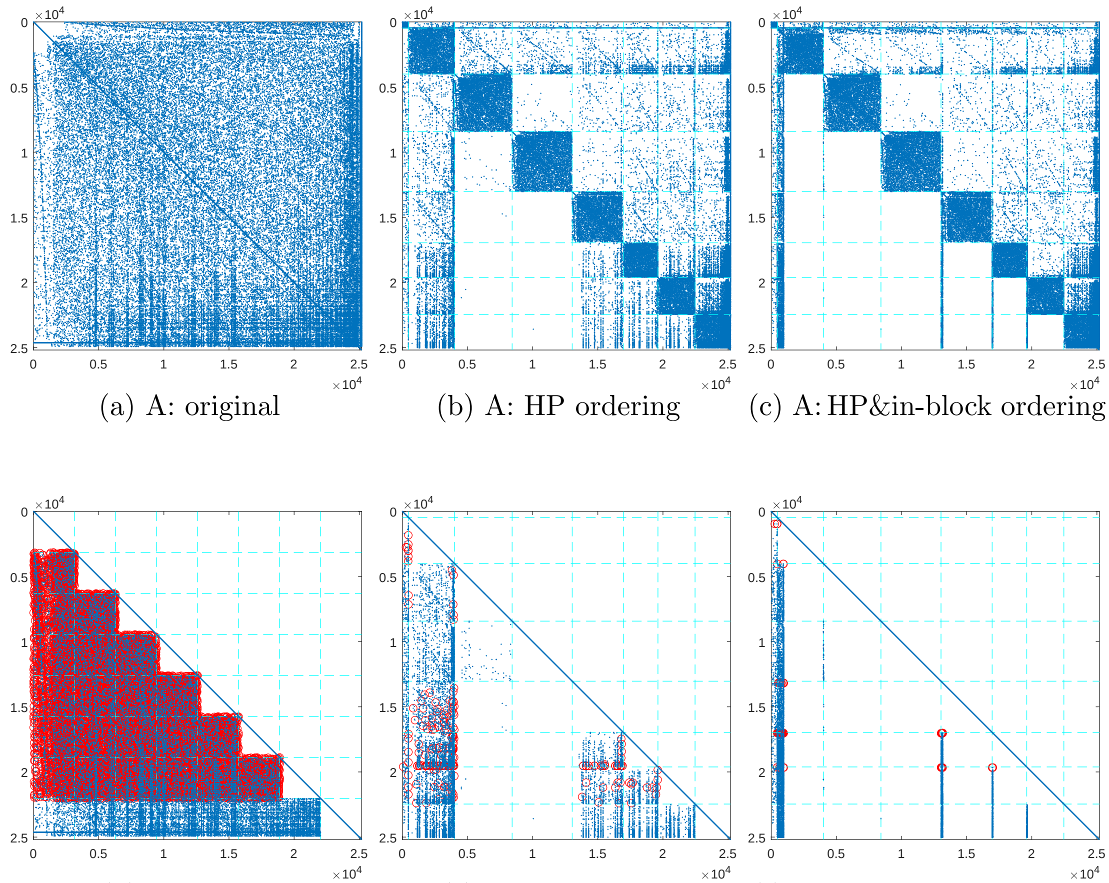


(d) S: original
$\text{nnz}(\widehat{S}-I) = 3{,}678$

(e) S: HP ordering
$\text{nnz}(\widehat{S}-I) = 256$

(f) S: HP&in-block ordering
$\text{nnz}(\widehat{S}-I) = 8$

Figure 4.6: Nonzero pattern of `ACTIVSg10K`: (a) before ordering, (b) after HP for $K = 8$, (c) after HP and in-block reordering; (d),(e),(f) the respective Spike (S) matrices (the reduced system ($\widehat{S}$) nonzeros are circled in red color).



(a) A: original (b) A: HP ordering (c) A: HP&in-block ordering

(d) S: original nnz($\widehat{S}-I$) = 28,063 (e) S: HP ordering nnz($\widehat{S}-I$) = 232 (f) S: HP&in-block ordering nnz($\widehat{S}-I$) = 116

Figure 4.7: Nonzero pattern of `mult_dcop_01`: (a) before ordering, (b) after HP for $K = 8$, (c) after HP and in-block reordering; (d),(e),(f) the respective Spike (S) matrices (the reduced system ($\widehat{S}$) nonzeros are circled in red color).

in-block reordering method gathers the reduced-system nonzeros within each off-diagonal block to the upper left corner of the respective off-diagonal block (Figure 4.5f). This is because we agglomerate the reduced system row indices to the top within each block, and we apply the resulting row reordering to the columns symmetrically. Within each off-diagonal block, gathering the rows with reduced-system indices to the top corresponds to agglomerating the columns with these indices, which are actually all the columns having nonzeros, to the left. An exception is the first column block since no row reordering is performed for the first row block.

4.3 Experiments

We use the HSL software package MC64 [106] for scaling and permuting the coefficient matrices to avoid a singular L. We select the MC64 option that maximizes the product of the diagonal entries and then scales to make the absolute value of diagonal entries one and the off-diagonal entries less than or equal to one. For symmetric matrices, in order not to destroy the symmetry, we apply the symmetric MC64 if the main diagonal is already zero-free. Otherwise, we apply the nonsymmetric MC64 to obtain a zero-free main diagonal. For unsymmetric matrices, we just apply the nonsymmetric MC64.

The experiments are conducted on an extensive dataset obtained from the SuiteSparse Matrix Collection [105]. For sufficiently coarse-grained parallel processing, we select real square matrices that have more than 20,000 rows and between 100,000 and 20,000,000 nonzeros. There are 199 symmetric and 208 unsymmetric matrices in SuiteSparse satisfying these properties at the time of experimentation. 44 symmetric and 4 unsymmetric matrices are eliminated because they are singular. The remaining are 155 symmetric and 204 unsymmetric, a total of 359 sparse matrices on which we conduct experiments. Table 4.1 shows the number of instances for each matrix kind. Kinds are sorted in decreasing order of instance count. The kinds having less than 5 instances in our dataset (acoustics, chemical oceanography, counter-example and data analytics) are grouped as

Table 4.1: Properties of test instances grouped by different matrix kinds.

kind ID	kind name	number of instances sym	unsym	total	avg size	avg nnz
1	structural	48	4	52	139,490	6,130,225
2	circuit simulation	2	46	48	176,964	1,032,036
3	economic	1	33	34	40,917	317,049
4	semiconductor device	0	33	33	99,011	1,341,221
5	computational fluid dynamics	6	27	33	79,052	1,700,320
6	2D/3D	19	9	28	105,005	1,448,801
7	power network	14	13	27	72,873	1,399,642
8	optimization	20	3	23	29,590	922,477
9	model reduction	13	3	16	114,294	6,574,878
10	chemical process simulation	0	15	15	52,025	1,668,867
11	theoretical/quantum chemistry	14	0	14	113,299	1,616,691
12	electromagnetics	6	4	10	92,019	593,073
13	thermal	5	4	9	134,034	1,123,944
14	materials	2	4	6	42,358	1,124,164
15	weighted graph	1	5	6	179,045	2,277,744
16	acoustics, oceanography, counter-ex., analytics	4	1	5	136,591	2,089,456
	All	155	204	359	92,079	1,484,609

one kind.

4.3.1 Partitioning Quality

We tested the performance of the proposed partitioning algorithm described in Section 4.2.1 against the partitioning quality of the conventional column-net HP with connectivity metric (cnHP) and graph partitioning (GP) models. For both cnHP and GP, vertex weights are set as the number of nonzeros in the respective rows whereas nets and edges are assigned unit cost. In cnHP, the objective is to minimize the number of nonzero off-diagonal column segments. In GP, the objective is to minimize the number of nonzeros in the off-diagonal blocks. For unsymmetric matrices, GP is applied on $|A|+|A^T|$. The well-known partitioning tools METIS [107] and PaToH [58] are used for GP and cnHP models, respectively.

In the proposed model, we use PaToH as the HP tool in each bipartitioning step. Experiments are conducted with different scaling factors $\alpha=1, 2, 5$ and 10 for lcn-net cost assignment. We set the maximum allowable imbalance ratio in

Table 4.2: Averages of total communication volume and the reduced system size in dmpGS, both normalized with respect to the number of rows.

	K	GP	cnHP	proposed HP model (Sec. 4.2.1) $\alpha=1$	$\alpha=2$	$\alpha=5$	$\alpha=10$
Comm. vol.	8	0.158	0.132	0.140	0.139	0.139	0.145
	16	0.253	0.217	0.223	0.224	0.224	0.232
	32	0.380	0.329	0.332	0.332	0.337	0.347
	64	0.547	0.477	0.479	0.479	0.491	0.505
	128	0.767	0.681	0.679	0.680	0.697	0.719
	256	1.062	0.955	0.948	0.953	0.977	1.012
Red. sys. size	8	0.048	0.041	0.033	0.032	0.029	0.029
	16	0.075	0.066	0.051	0.049	0.045	0.045
	32	0.109	0.094	0.074	0.070	0.065	0.064
	64	0.149	0.129	0.102	0.097	0.090	0.088
	128	0.197	0.174	0.136	0.129	0.119	0.116
	256	0.252	0.227	0.177	0.168	0.154	0.149

each bipartitioning as $\epsilon = 0.05$. As both METIS and PaToH involve randomized algorithms in the coarsening phase, five partitioning runs are performed for each instance with different seeds and the averages are reported. We conduct experiments for $K = 8, 16, 32, 64, 128$ and 256 parts (processors).

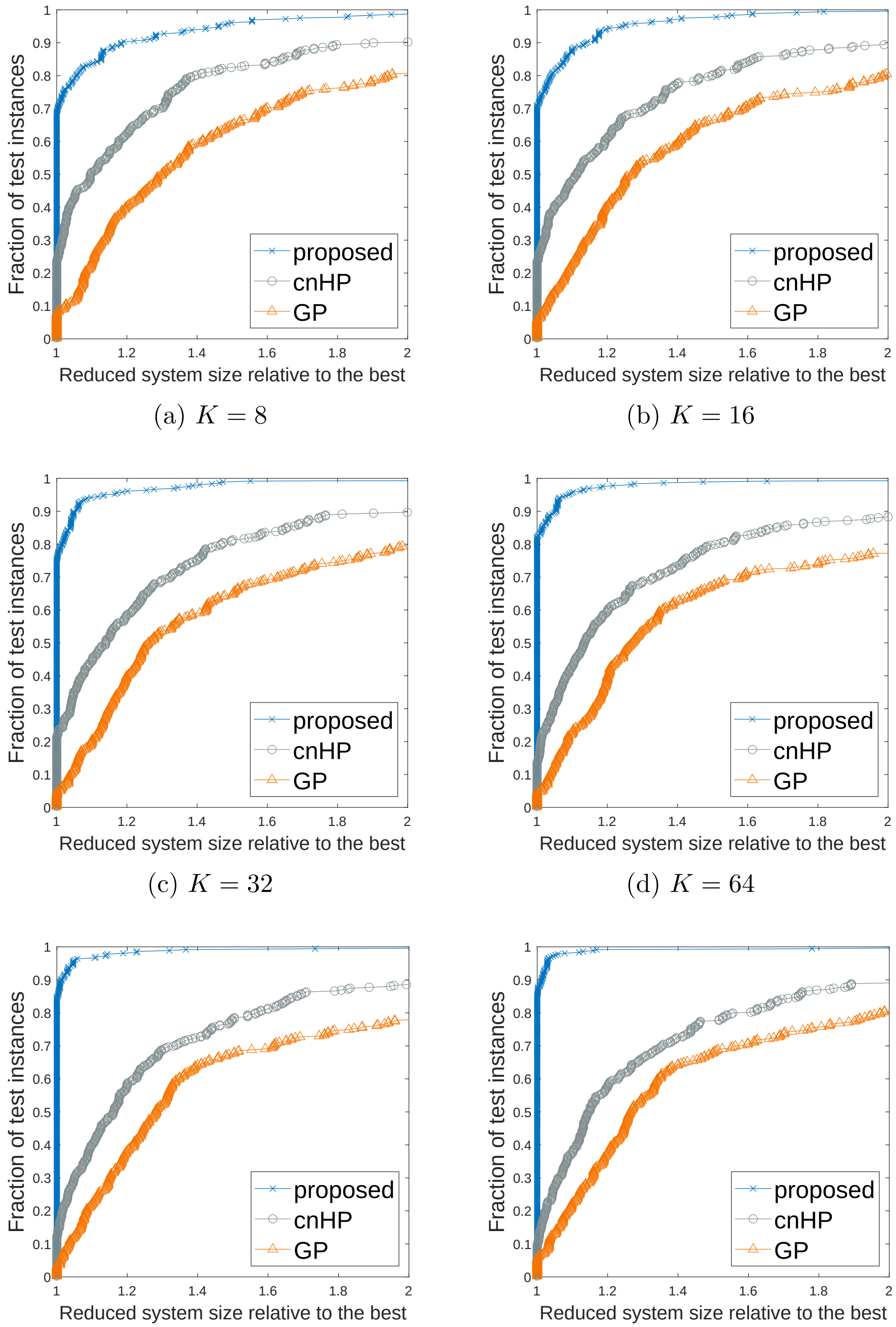
Table 4.2 shows the results of the comparison experiments in terms of the communication volume and the reduced system size metrics for dmpGS utilizing the partitions generated by GP, cnHP and the proposed model. For each test instance, these metrics are normalized with respect to the number of rows and the average for all matrices are given for each K. Here and hereafter, all averages are given as geometric means.

As seen in Table 4.2, cnHP achieves considerably low communication volume and reduced system size than GP as expected. The average improvement of cnHP over GP is approximately 10% for both metrics on $K = 256$. In fact, cnHP is equivalent to the proposed HP model for $\alpha = 0$. As seen in the table, there is a trade-off between the reduced system size and the communication volume for varying values of α for the proposed HP model. Yet the rate of increase in the communication volume is observed to be larger than the rate of decrease in the reduced system size with increasing α. For example for $K = 64$, compared to the cnHP model, the proposed model slightly increases the communication volume by 0.4%, 0.5%, 2.9% and 5.9% whereas it significantly decreases the reduced system size by 21.5%, 25.2%, 30.7% and 32.0% for $\alpha=1, 2, 5$ and 10, respectively. Here, $\alpha = 2$ seems to be a balanced choice since it significantly decreases the reduced system size while it slightly increases the communication volume. This is reflected in the parallel scalability of the proposed algorithm as will be shown in Section 4.3.3, thus we set $\alpha=2$ in the upcoming results.

In Figure 4.8, we provide the performance profiles comparing GP, cnHP and the proposed model in terms of the reduced system size. A performance profile [108] shows the comparison of different models relative to the best performing one for each data instance. On a profile, a point (x, y) means that the respective model is within x factor of the best result for a fraction y of the instances. For example, the point (1.20, 0.60) on the curve of cnHP means that cnHP yields 20% more reduced system size than the smallest reduced system size achieved for 60% of the dataset. Therefore, the model closest to the top left corner is interpreted as the model with best performance.

As seen in Figure 4.8, the proposed model outperforms the baseline algorithms in terms of the reduced system size in the majority of the test instances. As K increases, the performance gap between GP and cnHP decreases, whereas the performance gap between the proposed model and both of the baseline models increases significantly. The proposed model yields the best performance for 69%, 71%, 75%, 82%, 85% and 86% of the dataset for $K = 8, 16, 32, 64, 128$ and 256, respectively.

The proposed HP model yields very sparse off-diagonal blocks. For example, the number of nonzeros in any lower off-diagonal block R_k is at most 0.51%, 0.44%, 0.35%, 0.26%, 0.19%, and 0.13% of the total nonzero count of A for K=8, 16, 32, 64, 128, and 256 parts on the average, respectively. As the HP model maintains balance on the nonzero counts of the whole row blocks, these low nonzero counts in off-diagonal blocks do not disturb the computational load balance among processors considerably.



(a) $K = 8$ (b) $K = 16$

(c) $K = 32$ (d) $K = 64$

(e) $K = 128$ (f) $K = 256$

Figure 4.8: Performance profiles that compare GP, cnHP and the proposed HP model in terms of the reduced system size.

Table 4.3: Total height and nonzero count averages in the off-diagonal blocks of the reduced system ($\widehat{S}$). The values are the ratios of the results attained by the baseline over the proposed in-block reordering.

kind ID	K = 8		K = 16		K = 32		K = 64		K = 128		K = 256	
	height	nnz	height	nnz	height	nnz	height	nnz	height	nnz	height	nnz
1	1,470.1	518.5	554.1	233.6	263.6	143.1	115.9	67.6	65.9	41.1	37.2	25.4
2	71.9	125.2	63.1	100.6	30.5	61.6	15.8	35.0	8.8	18.7	5.5	11.6
3	1,219.2	331.4	321.2	271.5	296.8	197.2	167.8	152.7	88.2	82.9	46.1	47.8
4	27.7	3.8	16.8	5.3	9.9	5.7	8.8	6.2	6.5	4.5	4.6	3.1
5	260.0	10.0	142.6	9.1	90.3	7.6	63.5	6.3	37.6	4.7	24.5	4.0
6	600.0	123.2	298.3	101.5	148.6	61.3	73.5	37.0	38.7	22.3	22.0	13.6
7	131.8	10.1	67.3	7.4	36.7	5.7	22.8	4.7	14.0	3.8	8.5	3.2
8	513.5	97.3	260.4	59.4	92.2	30.9	48.9	18.5	23.8	11.3	17.0	8.2
9	1,547.9	1,101.3	1,010.9	1,221.2	556.7	641.8	248.6	315.4	102.0	141.6	50.7	70.0
10	29.0	4.8	32.9	10.9	15.0	5.6	12.8	5.2	8.6	3.6	6.0	2.7
11	375.3	619.3	213.3	213.3	112.8	136.6	68.8	89.0	43.2	54.1	25.6	32.0
12	241.2	170.7	121.4	109.1	59.9	66.8	31.8	41.2	18.1	25.2	10.6	14.7
13	18.2	2.7	17.7	3.1	12.7	2.7	13.4	2.6	10.1	2.6	8.2	2.8
14	217.7	231.1	116.5	149.6	59.2	94.0	33.0	54.4	19.1	30.6	12.2	17.7
15	610.4	228.9	277.9	164.6	122.0	91.6	61.8	50.9	31.5	28.3	18.0	16.4
16	15.8	62.2	8.5	31.5	6.0	18.9	4.4	11.9	3.4	7.8	2.7	5.3
All	238.1	57.2	127.1	43.0	65.0	27.8	39.0	18.7	22.7	12.1	14.3	8.3

4.3.2 In-Block Reordering Quality

To our knowledge, no in-block reordering method has been proposed or tested for stSPIKE in the literature. Therefore, we compare the improvement gained by applying the proposed in-block ordering method against an algorithm that does not apply an in-block reordering, which is our baseline algorithm. In this comparison, both the proposed and the baseline reordering methods utilize the partitions obtained by the HP model (Section 4.2.1). Two quality metrics used in this comparison are total height and nonzero count in the off-diagonal blocks of $\widehat{S}$.

Table 4.3 shows the ratios of these quality metrics of the in-block reorderings generated by the baseline to those of the proposed method. For each K value, the results are given as averages grouped by different matrix kinds, and the last row shows the average of all instances in the dataset.

As seen in Table 4.3, the proposed reordering method achieves significant improvement in terms of both quality metrics against the baseline reordering. For example for $K = 64$, on overall average, the proposed method achieves 39× and 18.7× improvement against the baseline ordering in terms of height and nonzero counts, respectively. The improvement rate attained in height does not always directly reflect to the improvement rate in the nonzero counts since height is an upper bound for fill-in and the fill-in also depends on the sparsity of the diagonal blocks.

Although the improvement of the proposed reordering against the baseline ordering tends to degrade with increasing K, this is expected since there are fewer rows per block and there is less room for improvement. For example on overall average, the proposed in-block reordering method achieves 57.2×, 43.0×, 27.8×, 18.7×, 12.1× and 8.3× decrease in the nonzero count for $K = 8, 16, 32, 64, 128$ and 256, respectively.

Note that the reduced system size and the communication volume are determined by the partitioning of the coefficient matrix and do not change with the in-block reordering. This is because the reduced system size is equal to the number of nonzero columns in R, and the communication volume is the sum of the reduced system size and the number of nonzero column segments in A, which are dependent on the block partitioning of A. The in-block reordering algorithm switches the locations of the nonzeros only within the same block, so the number of nonzero columns and column segments remain unchanged. Therefore, the results in Table 4.2 are still valid after applying the in-block reordering method.

The proposed partitioning and reordering model yields very small reduced systems whose nonzero counts are significantly low relative to the original system. For example, the average ratios of the nonzero count of the reduced system over the nonzero count of the coefficient matrix, i.e. $\text{nnz}(\widehat{S})/\text{nnz}(A)$, are 0.05%, 0.12%, 0.26%, 0.49%, 0.87%, and 1.48% for K=8, 16, 32, 64, 128, and 256 parts, respectively. These low nonzero counts of the reduced systems verify the effectiveness of the proposed partitioning and reordering model in terms of alleviating the sequential computational overhead of dmpGS.

Table 4.4 shows the number of nonzeros in $\widehat{S} - I$ obtained by applying the

Table 4.4: Average number of nonzeros in $\widehat{S} - I$ normalized with respect to the minimum possible nonzero count in $\widehat{S} - I$.

method	K=8	K=16	K=32	K=64	K=128	K=256
baseline	74.80	58.05	37.81	25.34	16.27	10.98
proposed	1.33	1.34	1.35	1.34	1.33	1.32

baseline and the proposed methods normalized with respect to the minimum possible nonzero count in $\widehat{S}-I$ after applying HP, as averages of different part counts. As seen in the table, the nonzero count obtained by the proposed method is at most 35% more than the minimum achievable nonzero count of $\widehat{S}-I$.

4.3.3 Parallel Scalability

Parallel experiments are performed on the Sariyer cluster of UHEM (National Center for High Performance Computing) [109] using up to 320 cores over 8 distributed nodes, each containing 40 cores (two Intel Xeon Gold 6148 CPUs) and 192GB memory. The nodes are connected by an InfiniBand EDR 100 Gbps network.

We implement an MPI+OpenMP hybrid parallel dmpGS to demonstrate the effectiveness of using stSPIKE and the proposed model. Throughout this section, the proposed model refers to the proposed partitioning and in-block reordering model (Section 4.2) applied to dmpGS. The number of MPI processes is the same as the number of parts (K) in a partition. For dmpGS, we experimented with different configurations of number of processes and threads. We found that the best configuration is 8 processes per node and 5 threads per process. Therefore, we conduct parallel experiments for dmpGS using 1, 2, 4 and 8 nodes corresponding to 40, 80, 160 and 320 cores and $K=8, 16, 32$ and 64 parts (processes), respectively.

To the best of our knowledge, there is no publicly available true distributed-memory parallel GS implementation. For comparing the performance of dmpGS,

Table 4.5: The properties of the matrices to run dmpGS. The relative residual and runtime results of mtGS are given for 500 iterations on 40 cores.

Matrix	Kind ID	Sym	Size	Nnz	Relative Residual*	mtGS time (s)
msdoor	1	✓	415,863	19,173,163	1.9×10^{-4}	23.1
af_shell1	1	✓	504,855	17,562,051	8.2×10^{-4}	23.4
af_1_k101	1	✓	503,625	17,550,675	1.1×10^{-4}	23.4
CoupCons3D	1		416,800	17,277,420	4.0×10^{-9}	21.8
Freescale1	2		3,428,755	17,052,626	3.0×10^{-4}	72.7
circuit5M_dc	2		3,523,317	14,865,409	1.9×10^{-12}	72.7
CurlCurl_3	9	✓	1,219,574	13,544,618	2.8×10^{-4}	35.4
memchip	2		2,707,524	13,343,948	5.4×10^{-5}	57.5
BenElechi1	6	✓	245,874	13,150,496	6.5×10^{-5}	15.2
pwtk	1	✓	217,918	11,524,432	1.5×10^{-4}	13.6
bmw3_2	1	✓	227,362	11,288,630	1.9×10^{-4}	13.6
bmwcra_1	1	✓	148,770	10,641,602	6.0×10^{-4}	11.9

we also implemented a multi-threaded GS (*mtGS*) by using the multithreaded sparse triangular system solver (mkl_sparse_d_trsm) and sparse matrix vector multiplicator (mkl_sparse_d_mv) of Intel Math Kernel Library (MKL) [110]. As a baseline, we obtain the results of mtGS on 40 threads/cores (1 node) by using the GP reordering since it is shown in [37] that the triangular solution with MKL benefits most from GP.

We tested the parallel scalability of dmpGS for a subset of the dataset since we have limited core hours on the HPC platform. From the dataset, we considered the matrices with at least 100,000 rows and 10,000,000 nonzeros, for which GS converges with a relative residual of less than 10^{-3} in 500 iterations with initial guess $x=[0,\ldots,0]^T$ and right-hand side vector $f = [1/m, 2/m, \ldots, 1]^T$. Then we select only those instances with different sparsity patterns from each matrix group. There were exactly 12 such matrices in our dataset satisfying these criteria. The properties of those matrices are shown in Table 4.5, sorted in decreasing order of nonzero counts. The sixth and the last column respectively show the relative residual and runtime of mtGS after 500 iterations.

Table 4.6 presents the relative residual values of dmpGS on 40, 80 and 160 cores (K=8, 16 and 32) for 500 iterations obtained by applying the original and the proposed ordering on the coefficient matrix. The results suggest that the

Table 4.6: Relative residual of dmpGS obtained by applying the original and the proposed ordering.

	K=8		K=16		K=32	
matrix	original	proposed	original	proposed	original	proposed
msdoor	2.04×10^{-4}	1.92×10^{-4}	2.04×10^{-4}	1.90×10^{-4}	2.04×10^{-4}	1.89×10^{-4}
af_shell1	6.04×10^{-4}	9.80×10^{-4}	6.04×10^{-4}	1.06×10^{-3}	6.04×10^{-4}	1.10×10^{-3}
af_1_k101	9.60×10^{-5}	1.09×10^{-4}	9.60×10^{-5}	1.11×10^{-4}	9.60×10^{-5}	1.12×10^{-4}
CoupCons3D	1.91×10^{-9}	4.39×10^{-9}	1.91×10^{-9}	4.40×10^{-9}	1.91×10^{-9}	4.47×10^{-9}
CurlCurl_3	3.21×10^{-4}	2.75×10^{-4}	3.21×10^{-4}	2.75×10^{-4}	3.21×10^{-4}	2.76×10^{-4}
BenElechi1	6.09×10^{-5}	6.65×10^{-5}	6.09×10^{-5}	6.57×10^{-5}	6.09×10^{-5}	6.60×10^{-5}
pwtk	1.43×10^{-4}	9.61×10^{-5}	1.43×10^{-4}	9.65×10^{-5}	1.43×10^{-4}	9.97×10^{-5}
bmw3_2	2.01×10^{-4}	2.07×10^{-4}	2.01×10^{-4}	2.08×10^{-4}	2.01×10^{-4}	2.09×10^{-4}
bmwcra_1	4.16×10^{-4}	4.45×10^{-4}	4.16×10^{-4}	4.62×10^{-4}	4.16×10^{-4}	4.76×10^{-4}

proposed reordering is successful in terms of sustaining the accuracy.

Table 4.7 shows the average speedup values obtained by dmpGS with GP, cnHP and the proposed model over mtGS. We run dmpGS with the proposed model for $\alpha = 1, 2, 5$ and 10 to observe the effect of scaling factor (α) on the parallel performance. As seen in the table, the proposed model achieves significantly higher speedup for dmpGS over the baseline models for all α. The speedup performance gap between the proposed and baseline models increase with increasing K, thus confirming the effectiveness of the proposed model.

Figure 4.9 depicts the performance profiles comparing the dmpGS runtime using the proposed model for varying α As can be seen in the figure, the relative performance of $\alpha = 2$ is better for larger part counts. For example for K=64,

Table 4.7: Average speedup obtained by dmpGS over mtGS on 40 cores. The best speedup value obtained for each K is shown in bold.

K	number of		GP	cnHP	proposed model			
	nodes	cores			$\alpha = 1$	$\alpha = 2$	$\alpha = 5$	$\alpha = 10$
8	1	40	9.87	8.71	**14.85**	14.71	14.77	14.51
16	2	80	14.65	13.58	**29.07**	28.51	28.47	28.25
32	4	160	17.41	16.11	47.28	**47.86**	47.24	45.89
64	8	320	15.79	17.60	54.96	**55.54**	50.21	50.65

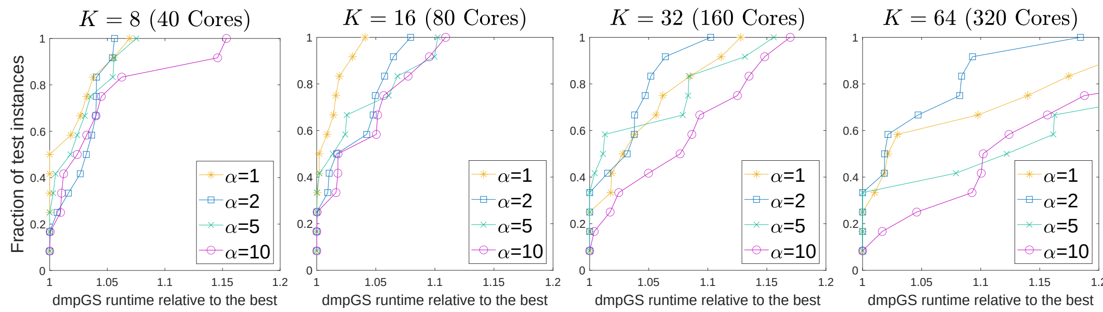


Figure 4.9: Performance profiles in terms of the dmpGS runtime using the proposed model.

even in the cases that $\alpha = 2$ does not give the best result, its performance is at most 1.18× worse than the best one.

We choose α=2 for better scalability of dmpGS since it yields the best performance for larger part counts ($K = 32$ and 64) as seen in both Table 4.7 and Figure 4.9. The proposed model yields average of 1.5×, 1.9×, 2.7× and 3.2× higher speedup relative to the best of the baseline models for $K = 8, 16, 32$ and 64, respectively.

Figure 4.10 shows the results of the strong scaling experiments as speedup curves of dmpGS with GP, cnHP and the proposed model. The proposed model significantly enhances the scalability of dmpGS so that dmpGS scales up to 320 cores on all instances. As seen in the figure, the proposed model outperforms GP and cnHP models for all of the test instances, significantly so in 9 out of 12. It is observed that the remaining 3 of them, namely `Freescale1`, `circuit5M_dc`, and `memchip`, are the only matrices among the test instances that have less than 5 nonzeros per row/column on the average. The closeness of the performances between the proposed and the baseline algorithms for such very sparse matrices was expected since there is less room for improvement. dmpGS using the proposed model achieves up to 122.2 speedup (for `memchip`) on 320 cores over mtGS on 40 cores.

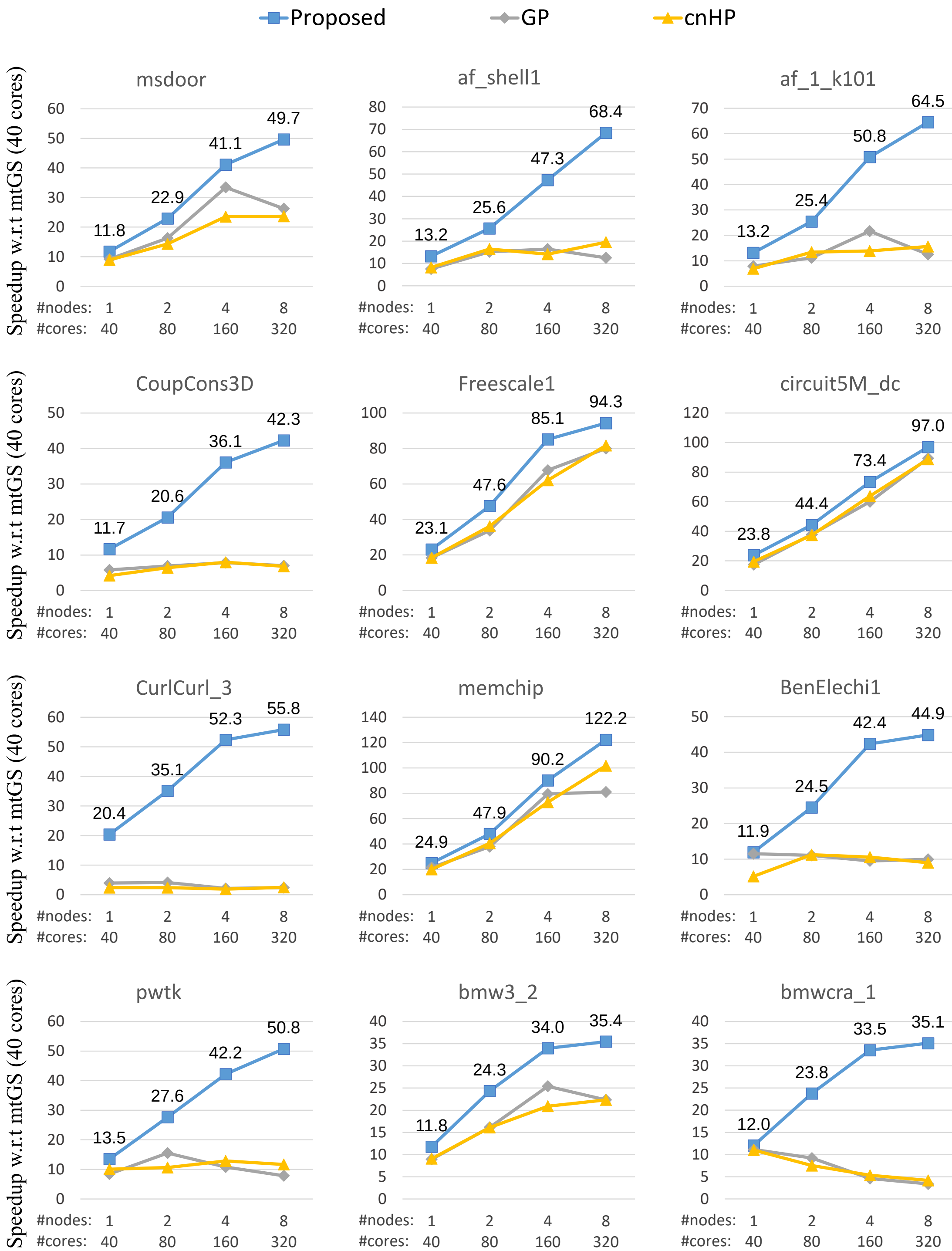


Figure 4.10: Speedup curves of dmpGS with GP, cnHP and the proposed model (for $K=8, 16, 32$ and 64) relative to mtGS on 1 node (40 cores).

4.4 Summary

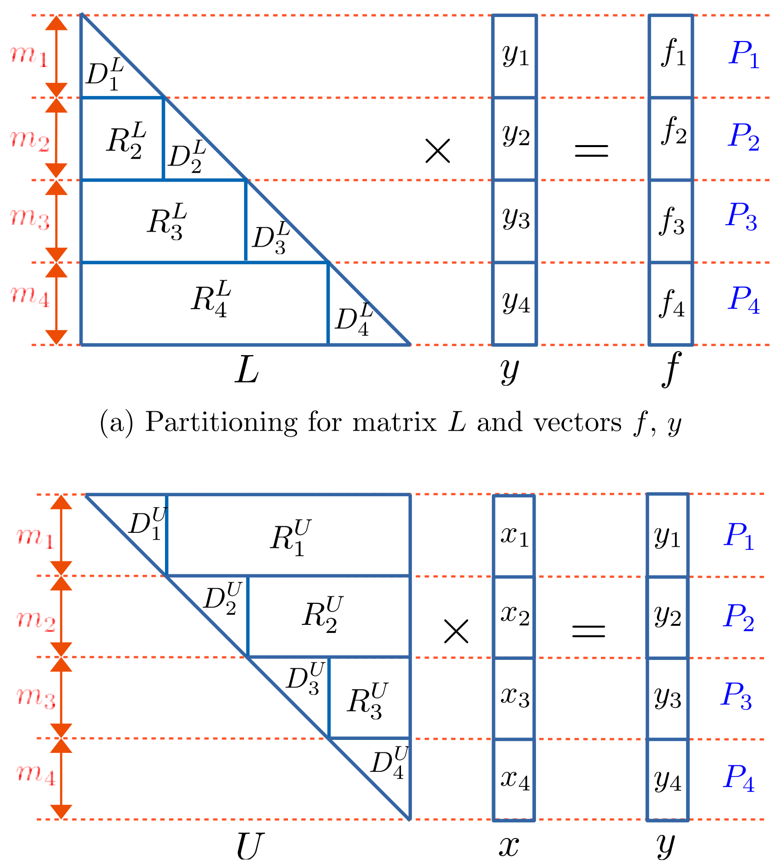
We proposed and implemented an stSPIKE-based distributed-memory parallel GS (dmpGS) algorithm. For improving the scalability of dmpGS, we propose an HP-based partitioning model and an in-block row reordering method. Extensive experiments show that the proposed HP model significantly decreases the reduced system size with respect to the baseline models while attaining comparable communication volume. The proposed in-block reordering method leads to a substantial decrease in the computational cost of both forming and solving the reduced system. Parallel experiments up to 320 cores demonstrate that using the proposed reordering model significantly improves the scalability of dmpGS.

Chapter 5

Partitioning and Reordering for Parallel Solution of Triangular Systems in ILU(0)

We propose a distributed-memory parallel algorithm, dmpGS, for parallel solution of triangular systems in ILU(0) using stSPIKE. For this purpose, we use both lower and upper stSPIKE algorithms to solve the lower and upper triangular systems in parallel. Solving the reduced systems in lower and upper stSPIKE constitutes the sequential computational bottleneck in dmpILU. Although dmpILU can be applied to any kind of incomplete LU, we consider ILU(0) since it guarantees that L and U matrices have the same sparsity pattern with A; so a priori reordering of A determines the size and nonzero count of reduced system in stSPIKE on L and U.

This chapter is organized as follows. In Section 5.1, we introduce the dmpILU algorithm. The proposed partitioning and reordering model for dmpILU is explained in Section 5.2. The experimental results are provided in Section 5.3 and Section 5.4 summarizes.



(a) Partitioning for matrix L and vectors f, y

(b) Partitioning for matrix U and vectors x, y

Figure 5.1: Four-way row-wise partition of matrices and vectors for dmpILU.

5.1 Distributed-Memory Parallel ILU (dmpILU)

Algorithm 6 represents the pseudocode of dmpILU. The superscript L or U letter indicate the case of belonging to upper or lower stSPIKE. For instance, $\mathcal{C}^L$ and $\mathcal{C}^U$ denote the reduced system index sets for stSPIKE on L and U, respectively. R_k^L and D_k^L respectively denote the k^{th} off-diagonal row block and the block diagonal of L; whereas R_k^U and D_k^U respectively denote the k^{th} off-diagonal row block and the block diagonal of U as shown in Figure 5.1.

The sequential computational overhead of solving the reduced systems at each iteration is proportional to the number of nonzeros in the off-diagonals of $\widehat{S}^L$ and $\widehat{S}^U$, i.e., in $\widehat{S}^L - I$ and $\widehat{S}^U - I$, respectively, since the nonzeros of I do not incur

Algorithm 6 dmpILU for processor P_k

Input: Submatrices $R_k^L, D_k^L, R_k^U, D_k^U$, and right-hand side subvector f_k

Output: Subvector x_k

if $2 \le k \le K-1$ **then**
    $G_k^L \leftarrow (D_k^L)^{-1} R_k^L$
    $G_k^U \leftarrow (D_k^U)^{-1} R_k^U$
    Form and send $\widehat{G}_k^L$ and $\widehat{G}_k^U$ to processor P_1
if $k = 1$ **then**
    Receive $\widehat{G}_\ell^L$ and $\widehat{G}_\ell^U$ from P_ℓ for $2 \le \ell \le K-1$ to form $\widehat{G}^L$ and $\widehat{G}^U$
    $\widehat{S}^L \leftarrow \widehat{G}^L + I$
    $\widehat{S}^U \leftarrow \widehat{G}^U + I$
iteratively do
▷ lower triangular stSPIKE
    $g_k \leftarrow (D_k^L)^{-1} f_k$ ▷ local sparse triangular solve
    if $k \neq 1$ **then**
        Send $\{g_k(i)\}_{i \in \mathcal{C}^L}$ to processor P_1
    if $k = 1$ **then**
        Receive $\{g_\ell(i)\}_{i \in \mathcal{C}^L}$ from P_ℓ for $2 \le \ell \le K$ to form $\widehat{g}^L$
        $\widehat{y} \leftarrow (\widehat{S}^L)^{-1} \widehat{g}^L$ ▷ lower triangular reduced system
        Send $\widehat{y}$ entries to requiring processors
    if $k \neq 1$ **then**
        Receive required $\widehat{y}$-entries to form $\bar{y}_k$
    $z_k \leftarrow R_k^L \bar{y}_k$ ▷ local SpMV
    $w_k \leftarrow (D_k^L)^{-1} z_k$ ▷ local sparse triangular solve
    $y_k \leftarrow g_k - w_k$
▷ upper triangular stSPIKE
    $g_k \leftarrow (D_k^U)^{-1} y_k$ ▷ local sparse triangular solve
    if $k \neq 1$ **then**
        Send $\{g_k(i)\}_{i \in \mathcal{C}^U}$ to processor P_1
    if $k = 1$ **then**
        Receive $\{g_\ell(i)\}_{i \in \mathcal{C}^U}$ from P_ℓ for $2 \le \ell \le K$ to form $\widehat{g}^U$
        $\widehat{x} \leftarrow (\widehat{S}^U)^{-1} \widehat{g}^U$ ▷ upper triangular reduced system
        Send $\widehat{x}$ entries to requiring processors
    if $k \neq 1$ **then**
        Receive required $\widehat{x}$-entries to form $\bar{x}_k$
    $z_k \leftarrow R_k^U \bar{x}_k$ ▷ local SpMV
    $w_k \leftarrow (D_k^U)^{-1} z_k$ ▷ local sparse triangular solve
    $x_k \leftarrow g_k - w_k$

any cost while forming or solving the reduced system.

The communication at lines 13-15 and 26-28 of Algorithm 6 are of size $|\mathcal{C}^L|$ and $|\mathcal{C}^U|$, respectively. Furthermore, the reduced system sizes put an upper bound on the nonzero count of the reduced system, that is the sequential computational overhead.

5.2 The Proposed Partitioning and Reordering Model for dmpILU

We propose a partitioning and reordering model that exploits the sparsity of the coefficient matrix for minimizing total size and nonzero count of the reduced systems in dmpILU. The first phase of the proposed model is a row-wise partitioning of the coefficient matrix, and the second phase is a local reordering of the rows within the induced row blocks.

In Section 5.2.1, we propose a novel hypergraph partitioning model that extends and enhances the conventional column-net model for decreasing the reduced system sizes in lower and upper triangular parts simultaneously. Decreasing the reduced system size is important since it contributes to reducing the communication volume, and it relates to decreasing the computational overhead. In order to encode the minimization of total number of nonzero column segments in the lower and upper triangular parts, we introduce new vertex fixing, net anchoring and net splitting schemes within the well-known recursive bipartitioning (RB) framework.

In Section 5.2.2, we propose an in-block row reordering method for decreasing the computational costs of solving the reduced systems, that is the total number of nonzeros in the lower and upper triangular reduced systems.

5.2.1 Hypergraph Partitioning Model

The objective of the proposed partitioning for improving the performance of dmpILU is to minimize the total size of the reduced systems in lower and upper stSPIKE, that is

$$PartObj = |\mathcal{C}^L| + |\mathcal{C}^U|, \tag{5.1}$$

which corresponds to minimizing the total number of off-diagonal nonzero columns in lower and upper triangular parts.

The key point here is to truly calculate the number of nonzero columns in both lower and upper triangular parts separately. Let us consider a column c_i which has nonzeros both in lower and upper triangular parts, versus another column c_j which has nonzeros only in upper triangular part. A usual column-net hypergraph partitioning model with cut-net metric would count 1 as the cutsize incurred by both of these columns. However, c_i should incur a cutsize of 2, while c_j incurs 1. As seen in this simple example, a new cutsize definition is needed. We want the following to hold:

1. If a column links multiple row-parts in off-diagonal blocks only in U, or only in L, then it should incur cutsize of 1.
2. If a column links multiple row-parts in off-diagonal blocks in both U and L then it should incur cutsize of 2.

5.2.1.1 Definitions and Layout

We define a column as *L-linking* or *U-linking* if it links at least one off-diagonal block in the lower or upper triangular part, respectively. That is, a column c_i in k^{th} column block $\mathcal{B}^c_k$ is L-linking if it links a row block $\mathcal{B}^r_\ell$ with $\ell > k$; or is U-linking if it links a row block $\mathcal{B}^r_\ell$ with $\ell < k$. Since L-linking and U-linking columns of A are respectively the nonzero columns of R^L and R^U, the number of L-linking (`L_link_cols`(A)) and U-linking (`U_link_cols`(A)) columns is equal to the total reduced system size, $|\mathcal{C}^L|+|\mathcal{C}^U|$. Therefore, the partitioning objective (5.1) can be rewritten as

$$PartObj = \texttt{L\_link\_cols}(A) + \texttt{U\_link\_cols}(A). \tag{5.2}$$

Let $\mathcal{H}=\mathcal{H}_{CN}(A)=(\mathcal{V},\mathcal{N})$ be the column-net hypergraph of an $m\times m$ sparse matrix A with nonzero diagonal entries. An ordered partition $\Pi_K=\langle\mathcal{V}_1,\mathcal{V}_2,\ldots,\mathcal{V}_K\rangle$ of $\mathcal{H}$ is decoded as a partitioning of A as explained in Section 2.1.2. Each net n_i of $\mathcal{H}$ connects vertex v_i since $A(i,i)\neq 0$ for each $1\leq i\leq m$. A net n_i with $v_i\in\mathcal{V}_k$ is called *L-cut* if it connects at least one vertex part $\mathcal{V}_\ell$ such that $\ell>k$; and called *U-cut* if it connects at least one vertex part $\mathcal{V}_\ell$ such that $\ell<k$. The set of L-cut and U-cut nets are denoted as $\mathcal{N}_{Lcut}$ and $\mathcal{N}_{Ucut}$, respectively. Then we define the *L-cut-net metric* and *U-cut-net metric* as

$$cs_{Lcut}(\Pi_K) = \sum_{n\in\mathcal{N}_{Lcut}} c(n), \tag{5.3}$$

$$cs_{Ucut}(\Pi_K) = \sum_{n\in\mathcal{N}_{Ucut}} c(n). \tag{5.4}$$

The partitioning objective of HP in this problem is minimizing the cost of partition Π_K which is defined as the sum of L-cut-net metric and U-cut-net metric with unit net cost, that is,

$$cost_{LUcut}(\Pi_K) = |\mathcal{N}_{Lcut}| + |\mathcal{N}_{Ucut}|. \tag{5.5}$$

Lemma 5.2.1. *A column c_i of A is L-linking or U-linking if and only if net n_i of $\mathcal{H}$ is L-cut or U-cut, respectively.*

Proof. Due to symmetric row-column ordering, c_i is in $\mathcal{B}_k^c$ if and only if r_i is in $\mathcal{B}_k^r$, which corresponds to $v_i\in\mathcal{V}_k$. Furthermore, c_i links $\mathcal{B}_\ell^r$ if and only if n_i connects $\mathcal{V}_\ell$. Therefore, c_i in $\mathcal{B}_k^c$ links $\mathcal{B}_\ell^r$ for $\ell>k$ if and only if n_i with $v_i\in\mathcal{V}_k$ connects $\mathcal{V}_\ell$, where $\ell>k$. That is, c_i is L-linking if and only if n_i is L-cut. Similarly, c_i in $\mathcal{B}_k^c$ links $\mathcal{B}_\ell^r$ for $\ell<k$ if and only if n_i with $v_i\in\mathcal{V}_k$ connects $\mathcal{V}_\ell$, where $\ell<k$. That is, c_i is U-linking if and only if n_i is U-cut. □

Proposition 5.2.2. *Minimizing $cost_{LUcut}(\Pi_K)$ for a K-way partition Π_K of $\mathcal{H}$ corresponds to minimizing the partitioning objective (5.2).*

Proof. By Lemma 5.2.1, the number of L-cut nets in $\mathcal{H}$ is equal to the number of L-linking columns in A. Thus $|\mathcal{N}_{Lcut}| = \texttt{L\_link\_cols}(A)$. Similarly, the number of U-cut nets in $\mathcal{H}$ is equal to the number of U-linking columns in A. Thus $|\mathcal{N}_{Ucut}| = \texttt{U\_link\_cols}(A)$. Therefore, $|\mathcal{N}_{Lcut}| + |\mathcal{N}_{Ucut}| = \texttt{L\_link\_cols}(A) + \texttt{U\_link\_cols}(A)$. □

In $\mathcal{H}$, each vertex is associated with a weight equal to the number of nonzeros in the respective row of the matrix, i.e., $w(v_i) = nnz(A(i,:))$. By this way, the partitioning constraint of maintaining balance on part weights approximately encodes the computational load balance during aggregate two triangular solves in dmpILU.

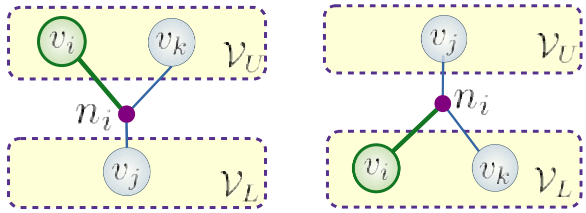
To the best of our knowledge, there exists no partitioning tool or model that can bipartition a given hypergraph with the objective of minimizing the L-cut-net and U-cut-net metrics simultaneously. We propose to use an RB framework with new net anchoring and splitting schemes to formulate the minimization of the sum of L-cut-net and U-cut-net metrics as a conventional hypergraph bipartitioning problem with cut-net metric.

5.2.1.2 Recursive Bipartitioning Model for dmpILU

At each RB step, an ordered bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ of $\mathcal{V}$ is decoded such that vertices in $\mathcal{V}_L$ are ordered after $\mathcal{V}_U$ where $\mathcal{V}_U$ and $\mathcal{V}_L$ denote the upper and lower parts, respectively. In RB, the concept of L-cut or U-cut net takes a special form. In a bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$, a net n_i is called

- L-cut, if v_i is assigned to $\mathcal{V}_U$ and n_i connects at least one vertex v_j such that $v_j \in \mathcal{V}_L$.
- U-cut, if v_i is assigned to $\mathcal{V}_L$ and n_i connects at least one vertex v_j such that $v_j \in \mathcal{V}_U$.

In other words, a cut net n_i is called L-cut if v_i is assigned to $\mathcal{V}_U$, and called



(a) n_i is L-cut (b) n_i is U-cut

Figure 5.2: Sample L-cut and U-cut nets.

U-cut if v_i is assigned to $\mathcal{V}_L$. Figure 5.2 shows sample L-cut and U-cut nets.

For encapsulating the L-cut-net and U-cut-net metrics simultaneously, we extend $\mathcal{H}=(\mathcal{V},\mathcal{N})$ into a hypergraph $\mathcal{H}'=(\mathcal{V}',\mathcal{N}')$ so that minimizing the number of conventional cut nets in $\mathcal{H}'$ encodes minimizing the partitioning objective. We introduce new fixed vertices $v_U\in\mathcal{V}'_U$ and $v_L\in\mathcal{V}'_L$ to form the extended vertex set $\mathcal{V}'=\mathcal{V}\cup\{v_U,v_L\}$. Here, v_U and v_L are fixed to parts $\mathcal{V}_U$ and $\mathcal{V}_L$, respectively.

We represent each net n_i in $\mathcal{H}$ as a pair of nets $\hat{n}_i$ and $\check{n}_i$ in $\mathcal{H}'$. We define three different states that a net can have. At the beginning, all nets are in *0-cut-state* meaning that they have not been in cut yet. When a net becomes L-cut in an RB step, we call it has *L-cut-state*. Being an internal net or L-cut net several times does not change its state. Similarly, when a net becomes U-cut in an RB step, we call it has *U-cut-state*. Being an internal net or U-cut net several times does not change its state. When a net in L-cut-state becomes U-cut, or when a net in U-cut-state becomes L-cut, then we change its state as *LU-cut-state*.

In our HP model, the extension of a net in $\mathcal{H}$ to $\mathcal{H}'$ is done in three different ways, depending on the state of the net as follows.

- If a net n_i of $\mathcal{H}$ is in 0-cut-state, then $\hat{n}_i$ is same as n_i except it is U-anchored (connects v_U), whereas $\check{n}_i$ is same as n_i except it is L-anchored (connects v_L). That is,

$$Pins(\hat{n}_i,\mathcal{H}') = Pins(n_i,\mathcal{H}) \cup \{v_U\}, \tag{5.6}$$

$$Pins(\check{n}_i,\mathcal{H}') = Pins(n_i,\mathcal{H}) \cup \{v_L\}. \tag{5.7}$$

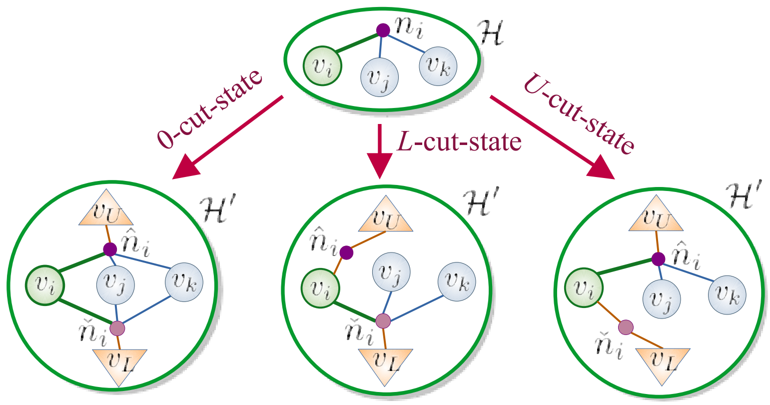


Figure 5.3: Extension of a net in $\mathcal{H}$ to $\mathcal{H}'$ for different states.

- If a net n_i of $\mathcal{H}$ is in L-cut-state, then $\check{n}_i$ is same as n_i except it is L-anchored (connects v_L), whereas $\hat{n}_i$ is a 2-pin U-anchored net which connects v_U and v_i. That is,

$$Pins(\hat{n}_i, \mathcal{H}') = \{v_i, v_U\}, \tag{5.8}$$

$$Pins(\check{n}_i, \mathcal{H}') = Pins(n_i, \mathcal{H}) \cup \{v_L\}. \tag{5.9}$$

- If a net n_i of $\mathcal{H}$ is in U-cut-state, then $\hat{n}_i$ is same as n_i except it is U-anchored (connects v_U), whereas $\check{n}_i$ is a 2-pin L-anchored net which connects v_L and v_i. That is,

$$Pins(\hat{n}_i, \mathcal{H}') = Pins(n_i, \mathcal{H}) \cup \{v_U\}, \tag{5.10}$$

$$Pins(\check{n}_i, \mathcal{H}') = \{v_i, v_L\}. \tag{5.11}$$

Figure 5.3 illustrates these three different kinds for extending a net in $\mathcal{H}$ to $\mathcal{H}'$ depending on its state. Extension of the L-cut-state and U-cut-state cases are defined for encapsulating the L-cut net and U-cut net metrics, respectively. The extension of nets having 0-cut-state to $\mathcal{H}'$ is for maintaining the equality of the effect of nets over the total cutsize.

Although we bipartition the original hypergraph at the beginning and do the extension on the hypergraphs starting from the second RB step, we can assume extending the hypergraph at the beginning for the ease of expression. We explain the proposed net splitting and removal methods on $\mathcal{H}$, and show the correspondence on $\mathcal{H}'$. For this purpose, we assume that each bipartition $\Pi_2' = \langle \mathcal{V}_U', \mathcal{V}_L' \rangle$ of $\mathcal{H}'$ induces a bipartition $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ of $\mathcal{H}$. The new hypergraphs $\mathcal{H}_U$ and $\mathcal{H}_L$ are constructed according to $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ as follows.

If a net is internal in $\mathcal{V}_L$, then it is included in $\mathcal{N}_L$ as is. Similarly, if a net is internal in $\mathcal{V}_U$, then it is included in $\mathcal{N}_U$ as is. The single-pin nets are discarded since they cannot contribute to the cutsize in the following RB steps.

We introduce a hybrid cut-net splitting/removal method in order to correctly encapsulate the L-cut-net and U-cut-net metrics at the same time. At each RB step, for each net pair $(\hat{n}_i, \check{n}_i)$ in a bipartition Π', we consider the state of n_i in Π for ease of understanding. If a net n_i is not internal, then it can be either L-cut or U-cut.

If a net n_i falls into the LU-cut-state, we immediately apply cut-net removal for n_i. This is because when n_i is LU-cut in an RB step, it also becomes both L-cut and U-cut in the final K-way partition. In the context of matrix partitioning, it means that the column c_i becomes both L-linking and U-linking. Therefore there is no need to track this respective net anymore and we do not include it in further bipartitions.

Otherwise, when n_i is U-cut in an RB step, we apply net removal towards $\mathcal{H}_U$ and *net-L-splitting* towards $\mathcal{H}_L$. That is, n_i is added to $\mathcal{H}_L$ as

$$Pins(n_i, \mathcal{H}_L) = Pins(n_i, \mathcal{H}) \cap \mathcal{V}_L. \tag{5.12}$$

This is because we do not need to keep track of n_i in further bipartitionings of $\mathcal{H}_U$ since it is already counted as U-cut, but it has the potential of becoming L-cut in further bipartitionings of $\mathcal{H}_L$. In the extended hypergraph, this corresponds to adding net pair $(\hat{n}_i, \check{n}_i)$ to $\mathcal{H}_L'$ such that

$$Pins(\hat{n}_i, \mathcal{H}_L') = (Pins(n_i, \mathcal{H}) \cap \mathcal{V}_L) \cup \{v_U\}, \tag{5.13}$$

$$Pins(\check{n}_i, \mathcal{H}_L') = \{v_i, v_L\}. \tag{5.14}$$

Conversely, when n_i is L-cut in an RB step, we apply net removal towards $\mathcal{H}_L$

and *net-U-splitting* towards $\mathcal{H}_U$. That is, n_i is added to $\mathcal{H}_U$ as

$$Pins(n_i, \mathcal{H}_U) = Pins(n_i, \mathcal{H}) \cap \mathcal{V}_U. \tag{5.15}$$

This is because we do not need to keep track of n_i in further bipartitionings of $\mathcal{H}_L$ since it is already counted as L-cut, yet it has the potential of becoming U-cut in further bipartitionings of $\mathcal{H}_U$. In the extended hypergraph, this corresponds to adding net pair $(\hat{n}_i, \check{n}_i)$ to $\mathcal{H}'_U$ such that

$$Pins(\hat{n}_i, \mathcal{H}'_U) = (Pins(n_i, \mathcal{H}') \cap \mathcal{V}'_U) \cup \{v_L\}, \tag{5.16}$$

$$Pins(\check{n}_i, \mathcal{H}'_U) = \{v_i, v_U\}. \tag{5.17}$$

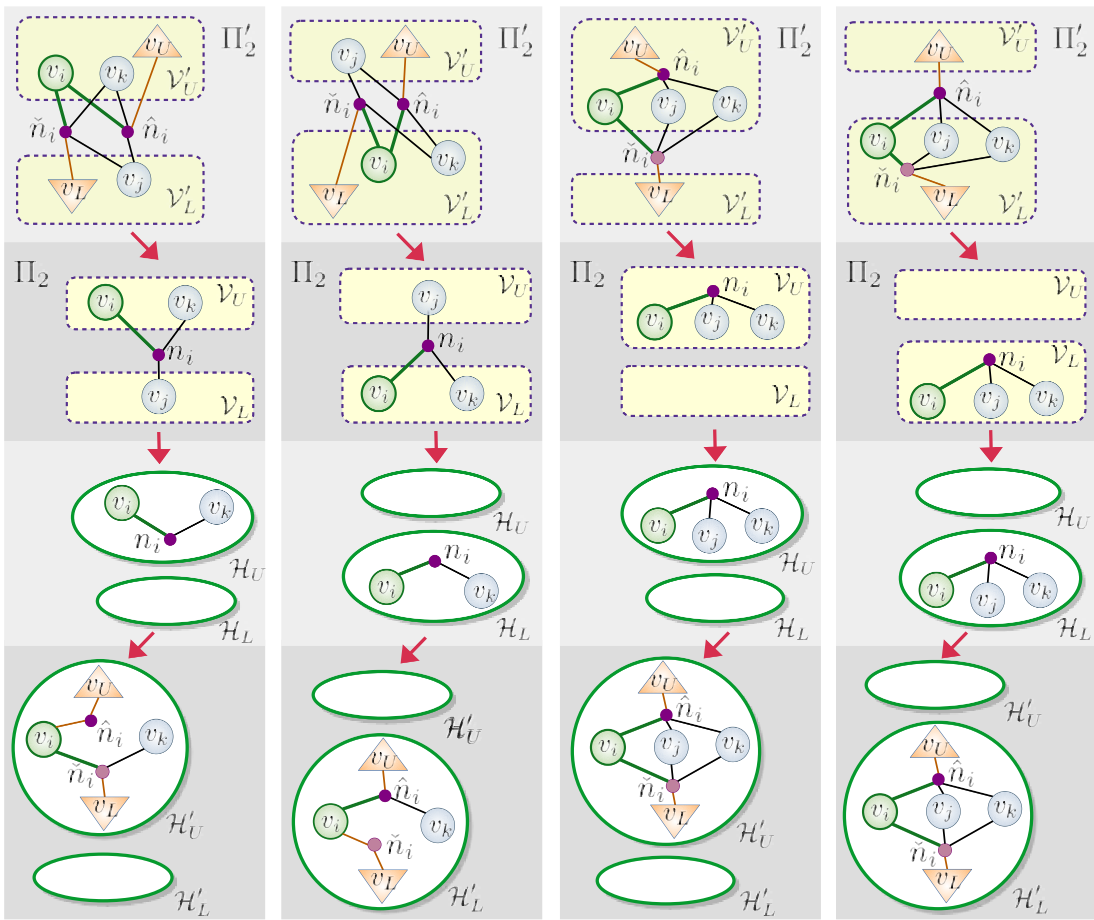
When a net turns into LU-cut-state from L-cut-state or U-cut-state; or it turns into L-cut-state or U-cut-state from 0-cut-state; we refer this situation as a *change of state* for that net. We denote the set of nets that encounter a change of state in a bipartition Π_2 as $\mathcal{N}_{cst}(\Pi_2)$.

Figures 5.4, 5.5 and 5.6 illustrate all possible cases for a sample net in 0-, L- and U- cut-states, respectively. The top horizontal layer shows a bipartition Π'_2 of the current hypergraph $\mathcal{H}'$. The second layer shows the corresponding bipartition Π_2 of $\mathcal{H}$. The third layer shows $\mathcal{H}_U$ and $\mathcal{H}_L$ induced by Π_2. Finally, the bottom layer shows the corresponding $\mathcal{H}'_U$ and $\mathcal{H}'_L$ induced by Π'_2.

In Figure 5.4, net n_i is assumed to have 0-cut-state before the bipartitioning for all cases. If n_i becomes L-cut as in Figure 5.4a, then it is extended to further hypergraphs as a net in L-cut-state. If n_i becomes U-cut as in Figure 5.4b, then it is extended to further hypergraphs as a net in U-cut-state. If n_i is internal as in Figure 5.4c or 5.4d, then it is extended to further hypergraphs as a net having 0-cut-state.

In Figure 5.5, net n_i is assumed to have L-cut-state before the bipartitioning for all cases. If n_i becomes U-cut as in Figure 5.5b, then it is not included in any further hypergraphs. Otherwise, it is still extended to further hypergraphs as a net having L-cut-state.

In Figure 5.6, net n_i is assumed to have U-cut-state before the bipartitioning



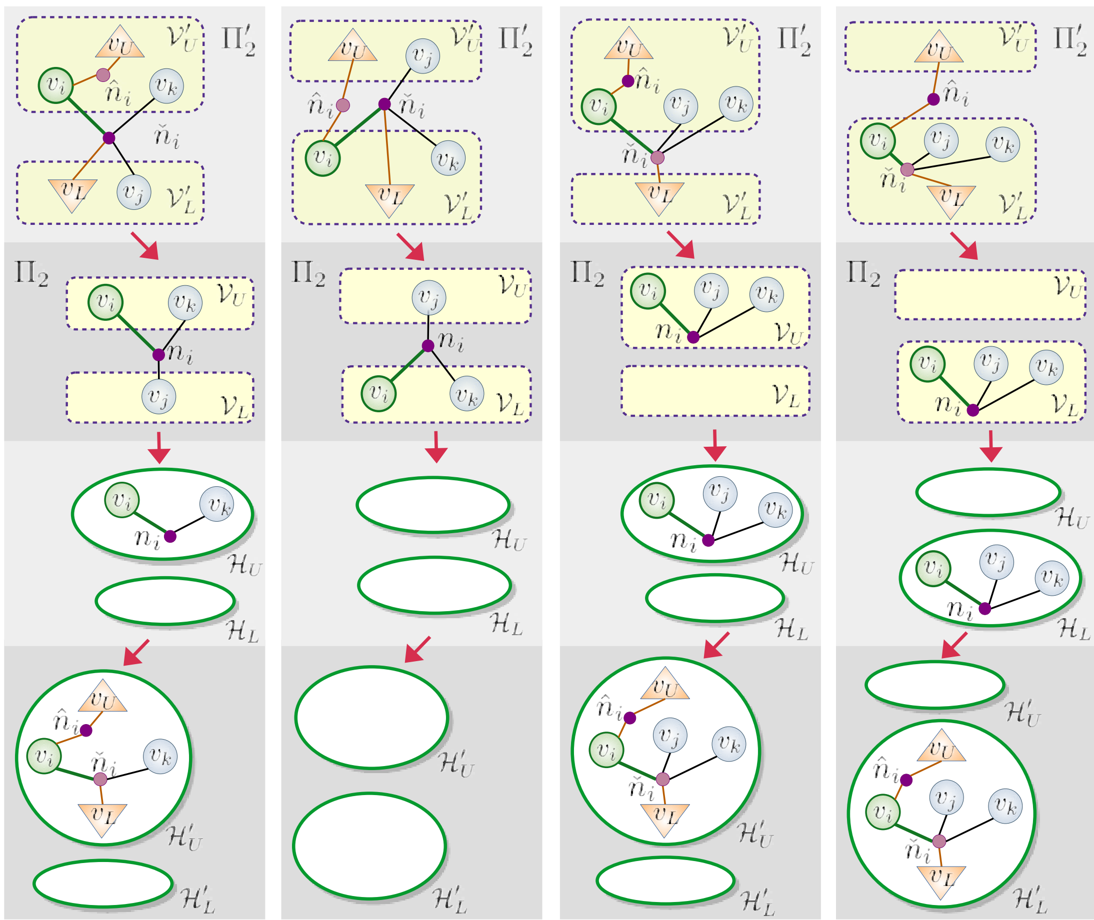
(a) n_i is L-cut

(b) n_i is U-cut

(c) n_i is internal to part $\mathcal{V}_U$

(d) n_i is internal to part $\mathcal{V}_L$

Figure 5.4: All cases for a net n_i in 0-cut-state and the corresponding net pair $(\hat{n}_i, \check{n}_i)$ after bipartition $\Pi'_2 = \langle \mathcal{V}'_U, \mathcal{V}'_L \rangle$.



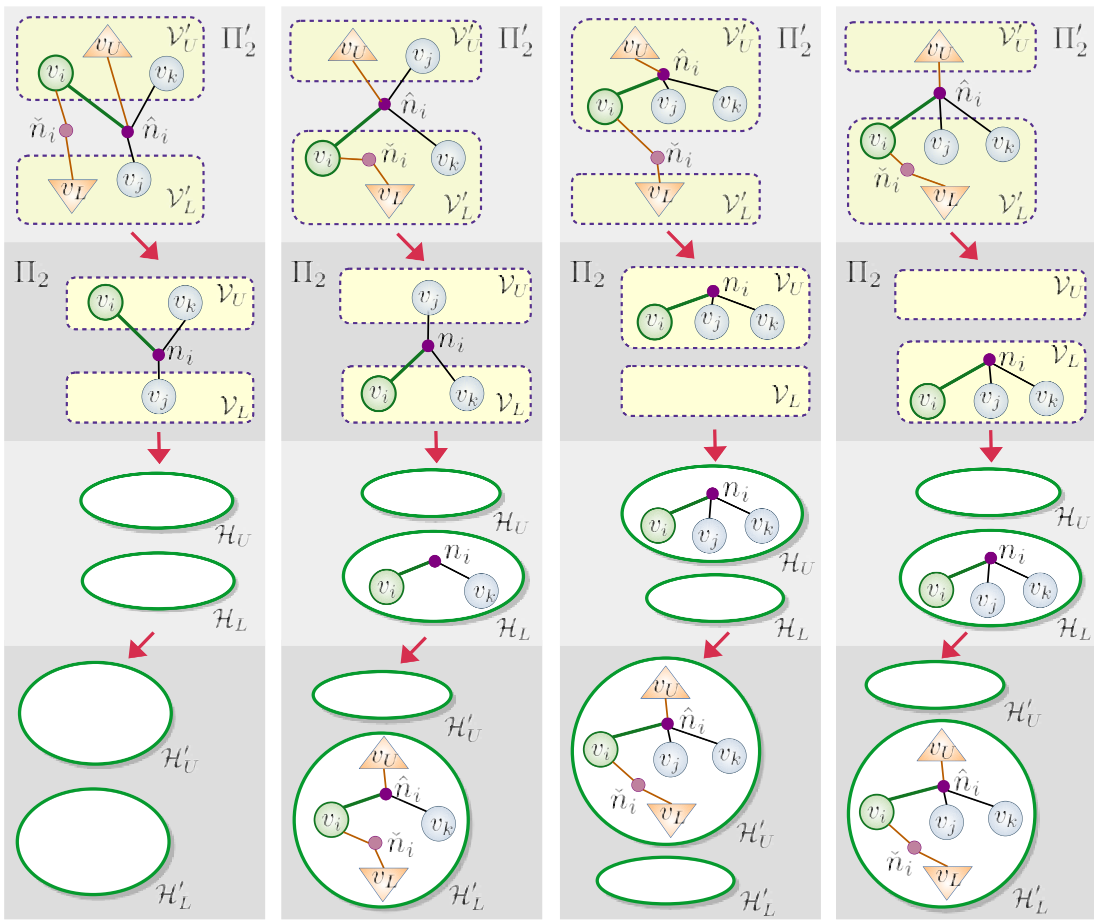
(a) n_i is L-cut

(b) n_i is U-cut

(c) n_i is internal to part $\mathcal{V}_U$

(d) n_i is internal to part $\mathcal{V}_L$

Figure 5.5: All cases for a net n_i in L-cut-state and the corresponding net pair $(\hat{n}_i, \check{n}_i)$ after bipartition $\Pi_2' = \langle \mathcal{V}_U', \mathcal{V}_L' \rangle$.



(a) n_i is L-cut

(b) n_i is U-cut

(c) n_i is internal to part $\mathcal{V}_U$

(d) n_i is internal to part $\mathcal{V}_L$

Figure 5.6: All cases for a net n_i in U-cut-state and the corresponding net pair $(\hat{n}_i, \check{n}_i)$ after bipartition $\Pi_2' = \langle \mathcal{V}_U', \mathcal{V}_L' \rangle$.

for all cases. If n_i becomes L-cut as in Figure 5.6a, then it is not included in any further hypergraphs. Otherwise, it is still extended to further hypergraphs as a net having U-cut-state.

Proposition 5.2.3. *Consider the bipartition* $\Pi_2 = \langle \mathcal{V}_U, \mathcal{V}_L \rangle$ *of* $\mathcal{H}$ *induced by a bipartition* $\Pi'_2 = \langle \mathcal{V}'_U, \mathcal{V}'_L \rangle$ *of* $\mathcal{H}'$ *in an RB step. The net pair* $(\hat{n}_i, \check{n}_i)$ *incurs 2 cut nets in* Π'_2 *only when the corresponding net* n_i *is in* $\mathcal{N}_{cst}(\Pi_2)$*. Otherwise, the net pair* $(\hat{n}_i, \check{n}_i)$ *incurs 1 cut net in* Π'_2*.*

Proof. For a net n_i having 0-cut-state, a change of state occurs when n_i becomes L-cut, or U-cut. There are four cases for a 0-cut-state as follows.

- If n_i is L-cut in Π_2 of $\mathcal{H}$, then $v_i \in \mathcal{V}_U$ and n_i connects a vertex v_j such that $v_j \in \mathcal{V}_L$. In Π'_2 of $\mathcal{H}'$, $\hat{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_j \in \mathcal{V}'_L$; and $\check{n}_i$ is also cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$.
- If n_i is U-cut in Π_2 of $\mathcal{H}$, then $v_i \in \mathcal{V}_L$ and n_i connects a vertex v_j such that $v_j \in \mathcal{V}_U$. In Π'_2 of $\mathcal{H}'$, $\hat{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_L$ and $v_j \in \mathcal{V}'_U$; and $\check{n}_i$ is also cut since it connects $v_i \in \mathcal{V}'_L$ and $v_U \in \mathcal{V}'_U$.
- If n_i is internal to $\mathcal{V}_L$, then in Π'_2, net $\hat{n}_i$ is also internal to $\mathcal{V}'_L$ since both v_i and v_L are in $\mathcal{V}'_L$; but $\check{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_L$ and $v_U \in \mathcal{V}'_U$.
- If n_i is internal to $\mathcal{V}_U$, then in Π'_2, net $\hat{n}_i$ is also internal to $\mathcal{V}'_U$ since both v_i and v_U are in $\mathcal{V}'_U$; but $\check{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$.

For a net n_i having L-cut-state, a change of state occurs only when n_i becomes U-cut. There are three different cases for a net in L-cut-state as follows.

- If n_i is U-cut in Π_2 of $\mathcal{H}$, then $v_i \in \mathcal{V}_L$ and n_i connects a vertex v_j such that $v_j \in \mathcal{V}_U$. In Π'_2 of $\mathcal{H}'$, $\hat{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_L$ and $v_j \in \mathcal{V}'_U$; and $\check{n}_i$ is also cut since it connects $v_i \in \mathcal{V}'_L$ and $v_U \in \mathcal{V}'_U$.
- If n_i is not U-cut and $v_i \in \mathcal{V}_U$ in Π_2, then $\hat{n}_i$ is cut in Π'_2 because it connects $v_L \in \mathcal{V}'_L$ and $v_i \in \mathcal{V}'_U$; but $\check{n}_i$ is not cut since both v_i and v_U are in $\mathcal{V}'_U$.
- If n_i is not U-cut and $v_i \in \mathcal{V}_L$ in Π_2, then n_i should be internal to $\mathcal{V}_L$; because otherwise, there would be at least one pin in $\mathcal{V}_U$ which would make n_i to be U-cut. In Π'_2, net $\hat{n}_i$ is internal to $\mathcal{V}'_L$ since both v_i and v_L are in $\mathcal{V}'_L$; but $\check{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_L$ and $v_U \in \mathcal{V}'_U$.

For a net n_i having U-cut-state, a change of state occurs only when n_i becomes L-cut. There are three cases for a net in U-cut-state as follows.

- If n_i is L-cut in Π_2 of $\mathcal{H}$, then $v_i \in \mathcal{V}_U$ and n_i connects a vertex v_j such that $v_j \in \mathcal{V}_L$. In Π'_2 of $\mathcal{H}'$, $\hat{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_j \in \mathcal{V}'_L$; and $\check{n}_i$ is also cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$.

- If n_i is not L-cut and $v_i \in \mathcal{V}_L$ in Π_2, then $\hat{n}_i$ is cut in Π'_2 because it connects $v_U \in \mathcal{V}'_U$ and $v_i \in \mathcal{V}'_L$; but $\check{n}_i$ is not cut since both v_i and v_L are in $\mathcal{V}'_L$.

- If n_i is not L-cut and $v_i \in \mathcal{V}_U$ in Π_2, then n_i should be internal to $\mathcal{V}_U$; because otherwise, there would be at least one pin in $\mathcal{V}_L$ which would make n_i to be L-cut. In Π'_2, net $\hat{n}_i$ is internal to $\mathcal{V}'_U$ since both v_i and v_U are in $\mathcal{V}'_U$; but $\check{n}_i$ is cut since it connects $v_i \in \mathcal{V}'_U$ and $v_L \in \mathcal{V}'_L$. □

Theorem 5.2.4. *Recursively bipartitioning $\mathcal{H}'$ by minimizing the cutsize according to the conventional cut-net metric and applying the proposed net splitting and removal strategies until reaching K parts encodes minimizing the partitioning objective (5.2).*

Proof. By Proposition 5.2.3, for a bipartition Π_2, each net in $\mathcal{N}_{cst}(\Pi_2)$ incurs 2 cut nets in Π'_2, whereas all remaining nets in Π_2 incur 1 cut net in Π'_2. Thus, the cutsize in Π'_2 is equal to $|\mathcal{N}_{cst}(\Pi_2)|+|\mathcal{N}|$. Since $|\mathcal{N}|$ is constant, minimizing the cutsize of Π'_2 is equivalent to minimizing $|\mathcal{N}_{cst}(\Pi_2)|$ for each bipartition Π_2. A change of state for a net means the first occurrence of being an L-cut or U-cut net. Furthermore, proposed net splitting and removal strategies ensure that an L-cut or U-cut net in Π_K is also L-cut in Π_2, and vice versa. Therefore, the sum of $|\mathcal{N}_{cst}(\Pi_2)|$ values over all bipartitions Π_2 gives the exact number of L-cut and U-cut nets in Π_K, which is equal to $|\mathcal{N}_{Lcut}| + |\mathcal{N}_{Ucut}| = cost_{LUcut}(\Pi_K)$ (5.5). Finally by Proposition 5.2.2, minimizing $cost_{LUcut}(\Pi_K)$ encodes minimizing the partitioning objective (5.2). □

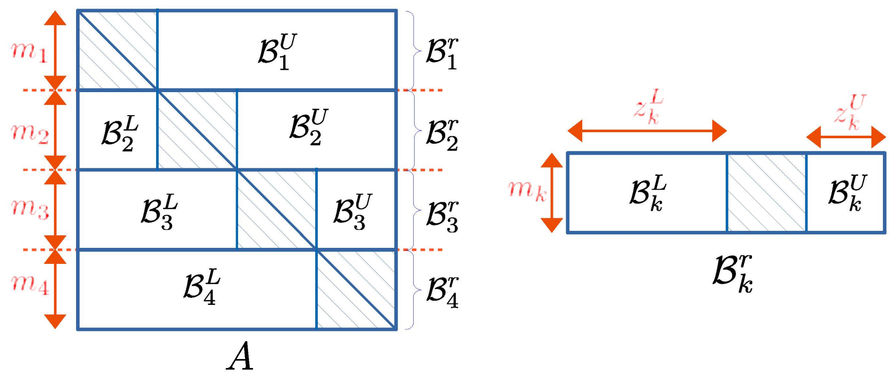


Figure 5.7: A sample row-wise partitioning of matrix A and a focus on a single row block $\mathcal{B}_k^r$.

5.2.2 Reordering within Row Blocks

We consider the K-way block structure of A induced by the partial symmetric row-column permutation obtained by the HP model in Section 5.2.1. We illustrate a sample row-block partition of A and a focus on a single row block $\mathcal{B}_k^r$ in Figure 5.7. Here, $\mathcal{B}_k^L$ and $\mathcal{B}_k^U$ respectively denote the submatrices of $\mathcal{B}_k^r$ lying in the lower and upper block triangular parts of A. The sizes of $\mathcal{B}_k^L$ and $\mathcal{B}_k^U$ are respectively $m_k \times z_k^L$ and $m_k \times z_k^U$ where

$$z_k^L = \sum_{i=1}^{k-1} m_i \quad \text{and} \quad z_k^U = \sum_{i=k+1}^{K} m_i. \tag{5.18}$$

Note that the nonzero pattern of $\mathcal{B}_k^L$ and $\mathcal{B}_k^U$ are respectively same as the nonzero pattern of R_k^L and R_k^U since the nonzero pattern of L and U are same as the nonzero pattern of A in ILU(0).

We perform row reordering within the k^{th} row block $\mathcal{B}_k^r$ by considering the nonzeros of both $\mathcal{B}_k^L$ and $\mathcal{B}_k^U$. At the end, the row reordering within k^{th} row block $\mathcal{B}_k^r$ is symmetrically applied to the columns of the k^{th} column block $\mathcal{B}_k^c$. For simplicity, we assume a local indexing for the rows of $\mathcal{B}_k^r$ so that it consists of rows r_i with $1 \le i \le m_k$. Let $\mathcal{C}_k^L$ and $\mathcal{C}_k^U$ denote the subset of $\mathcal{C}^L$ and $\mathcal{C}^U$ corresponding to the row indices in $\mathcal{B}_k^L$ and $\mathcal{B}_k^U$, respectively.

Recall that, fill-in may arise below the top nonzero of each spike in $\mathcal{B}_k^L$ for lower stSPIKE and above the bottom nonzero of each spike in $\mathcal{B}_k^U$ for upper stSPIKE.

The top nonzero of a spike c_j in $\mathcal{B}_k^L$ is defined as the nonzero with row index

$$top(c_j, \mathcal{B}_k^L) = \min\{i : \mathcal{B}_k^L(i,j) \neq 0,\ 1 \leq i \leq m_k\}. \tag{5.19}$$

The bottom nonzero of a spike c_j in $\mathcal{B}_k^U$ is defined as the nonzero with row index

$$bottom(c_j, \mathcal{B}_k^U) = \max\{i : \mathcal{B}_k^U(i,j) \neq 0,\ 1 \leq i \leq m_k\}. \tag{5.20}$$

We define *height*^{L} of a spike c_j in $\mathcal{B}_k^L$ as the number of reduced system row indices between $top(c_j, \mathcal{B}_k^L)$ and m_k inclusively, i.e.,

$$\text{height}^L(c_j, \mathcal{B}_k^L) = |\{i : top(c_j, \mathcal{B}_k^L) \leq i \leq m_k,\ i \in \mathcal{C}_k^L\}|, \tag{5.21}$$

since only the rows with indices in $\mathcal{C}_k^L$ may contribute to the nonzero count of $\widehat{S}^L$. Conversely, we define *height*^{U} of a spike c_j in $\mathcal{B}_k^U$ as the number of reduced system row indices between 1 and $top(c_j, \mathcal{B}_k^U)$ inclusively, i.e.,

$$\text{height}^U(c_j, \mathcal{B}_k^U) = |\{i : 1 \leq i \leq bottom(c_j, \mathcal{B}_k^U),\ i \in \mathcal{C}_k^U\}|, \tag{5.22}$$

since only the rows with indices in $\mathcal{C}_k^U$ may contribute to the nonzero count of $\widehat{S}^U$. The height of a spike in $\mathcal{B}_k^L$ or $\mathcal{B}_k^L$ constitutes an upper bound on the nonzero count (including the fill-in) of the corresponding column in $\widehat{S}^L$ and $\widehat{S}^U$, respectively. We assume the height of a non-spike column as zero. The objective of in-block reordering is to minimize the *total height*

$$\sum_{k=2}^{K-1} \left(\sum_{j=1}^{z_k^L} \text{height}^L(c_j, \mathcal{B}_k^L) + \sum_{j=m-z_k^U}^{m} \text{height}^U(c_j, \mathcal{B}_k^U) \right), \tag{5.23}$$

which constitutes an upper bound on the total nonzero count in off-diagonal blocks of the lower and upper triangular reduced systems. The first and last blocks do not contribute nonzeros to reduced systems since the reduced system index sets are empty.

We define the degree of a row in $\mathcal{B}_k^L$ or $\mathcal{B}_k^U$ as the number of nonzeros of that row lying in $\mathcal{B}_k^L$ or $\mathcal{B}_k^U$, that is,

$$deg_k^L(r_i) = |\{j : A(i,j) \neq 0,\ 1 \leq j \leq z_k^L\}|,\ \text{and} \tag{5.24}$$

$$deg_k^U(r_i) = |\{j : A(i,j) \neq 0,\ m - z_k^U \leq j \leq m\}|. \tag{5.25}$$

Note that if we were to reorder the rows of $\mathcal{B}_k^r$ by just considering the nonzeros of $\mathcal{B}_k^L$, we would aim to order the rows with smaller degrees in $\mathcal{B}_k^L$ to upper positions of $\mathcal{B}_k^r$ since placing denser rows to upper positions incurs more height in lower stSPIKE. In that case, it would be better to place the rows whose indices are not among $\mathcal{C}_k^L$ to the bottom of $\mathcal{B}_k^r$ to avoid the nonzeros of the rows that are not in $\mathcal{C}_k^L$ to contribute to the total height. We call this method, that considers only the lower triangular part and places the rows whose indices are not among $\mathcal{C}_k^L$ to the top of $\mathcal{B}_k^r$ in increasing order of deg_k^L, as `incr_L`.

Conversely, if we were to reorder the rows of $\mathcal{B}_k^r$ by just considering the nonzeros of $\mathcal{B}_k^U$, we would aim to order the rows with smaller degrees in $\mathcal{B}_k^U$ to lower positions of $\mathcal{B}_k^r$ since placing denser rows to lower positions incurs more height in upper stSPIKE. In that case, it would be better to place the rows whose indices are not among $\mathcal{C}_k^U$ to the top of $\mathcal{B}_k^r$ to avoid the nonzeros of the rows that are not in $\mathcal{C}_k^U$ to contribute to the total height. We adopt an in-between strategy that takes the nonzeros in both $\mathcal{B}_k^L$ and $\mathcal{B}_k^L$ into account. We call this method, that considers only the upper triangular part and places the rows whose indices are not among $\mathcal{C}_k^U$ to the bottom of $\mathcal{B}_k^r$ in decreasing order of deg_k^U, as `decr_U`.

Algorithm 7 presents the pseudocode of the proposed reordering method for the rows of $\mathcal{B}_k^r$. Note that determining the orderings within different row blocks are independent and can be done concurrently. First, we place the rows that have no nonzero in $\mathcal{B}_k^L$ and does not belong to $\mathcal{C}_k^U$ to the top of $\mathcal{B}_k^r$ as in lines 4-6. Conversely, we place the rows that have no nonzero in $\mathcal{B}_k^U$ and does not belong to $\mathcal{C}_k^L$ to the bottom of $\mathcal{B}_k^r$ as in lines 7-9. For the rest of the rows, we use three priority queues to keep track of three different kinds of rows. The priority queues are implemented as min-heap. We place the rows which belong to $\mathcal{C}_k^L$ but not to $\mathcal{C}_k^U$ to the top of the remaining places in increasing order of their $deg_k^L(r_i)$ values. Then we place the rows which belong to $\mathcal{C}_k^U$ but not to $\mathcal{C}_k^L$ to the bottom of the remaining places in decreasing order of their $deg_k^U(r_i)$ values. We place the remaining rows to remaining positions in increasing order of their $deg_k^L(r_i) - deg_k^U(r_i)$ values, with the aim of placing the rows with lower $deg_k^L(r_i)$ and higher $deg_k^U(r_i)$ values to the upper positions.

Algorithm 7 Proposed in-block reordering for row block $\mathcal{B}_k^r$ where $2 \le k \le K-1$

Input: $\mathcal{B}_k^r$ and reduced-system index sets $\mathcal{C}_k^L$ and $\mathcal{C}_k^U$.

Output: the permutation vector *perm* for rows of $\mathcal{B}_k^r$.

$indx \leftarrow 1$
$indx\_back \leftarrow m_k$
for each row r_i of $\mathcal{B}_k^r$ **do**
  if $deg_k^L(r_i) = 0$ and $i \notin \mathcal{C}_k^U$ **then**
    $perm(indx) = i$
    $indx \leftarrow indx+1$
  else if $deg_k^U(r_i) = 0$ and $i \notin \mathcal{C}_k^L$ **then**
    $perm(indx\_back) = i$
    $indx\_back \leftarrow indx\_back-1$
  else if $i \in \mathcal{C}_k^L$ but $i \notin \mathcal{C}_k^U$ **then**
    INSERT$(heap^L, i, deg_k^L(r_i))$
  else if $i \in \mathcal{C}_k^U$ but $i \notin \mathcal{C}_k^L$ **then**
    INSERT$(heap^U, i, deg_k^U(r_i))$
  else
    INSERT$(heap^{LU}, i, deg_k^L(r_i) - deg_k^U(r_i))$
while $heap^L$ is not empty **do**
  $perm(indx) =$ EXTRACT_MIN$(heap^L)$
  $indx \leftarrow indx+1$
while $heap^{LU}$ is not empty **do**
  $perm(indx) =$ EXTRACT_MIN$(heap^{LU})$
  $indx \leftarrow indx+1$
while $heap^U$ is not empty **do**
  $perm(indx\_back) =$ EXTRACT_MIN$(heap^U)$
  $indx\_back \leftarrow indx\_back-1$

5.3 Experimental Results

The experiments are conducted on the dataset which is described in Section 5.3. It consists of 359 sparse matrices whose properties are summarized in Table 4.1. We use the HSL software package MC64 [106] for scaling and permuting the coefficient matrices to avoid zero value on the diagonal.

5.3.1 Partitioning Quality

We tested the performance of the proposed hypergraph partitioning algorithm described in Section 5.2.1 against the partitioning quality of the conventional column-net HP with cut-net metric (cnHP) and graph partitioning (GP) models. For all models, vertex weights are set as the number of nonzeros in the respective rows whereas nets and edges are assigned unit cost. The objective of partitioning in cnHP is minimizing the number of linking columns whereas in the proposed model, it is minimizing the total number of L-linking and U-linking columns. In GP, the objective is to minimize the number of nonzeros in the off-diagonal blocks. For GP and cnHP models, the partitioning tools METIS [107] and PaToH [58] are used, respectively. In the proposed HP model, we use PaToH as a tool to bipartition the hypergraph at each RB step. The maximum allowable imbalance ratio in each bipartitioning is set as $\epsilon = 0.05$.

We investigate the comparison of partitioning qualities in terms of total reduced system size of dmpILU utilizing the partitions generated by the original ordering, GP, cnHP and the proposed model. The experiments are conducted for $K = 8, 16, 32, 64, 128$ and 256 parts for each instance in the dataset.

Table 5.1 shows the average improvements gained by using the proposed and baseline partitioning models against the original ordering. The results for each K are given as averages of 359 sparse matrices in the dataset. In the table, we use the notation rss() as a function of total reduced system size obtain by a model. rss(org) means the total reduced system size obtained with the original matrix

Table 5.1: Improvement averages of total reduced system size (rss) in upper and lower stSPIKE, as ratios with respect to the original ordering, i.e. the ratio of rss(org) / rss(model) where model ∈ {GP, cnHP, prop}.

		improvement over org		
K	rss(org)	GP	cnHP	prop
8	32,352	3.59	4.04	3.94
16	45,675	3.30	3.60	3.64
32	61,796	3.08	3.31	3.49
64	79,359	2.90	3.10	3.33
128	96,654	2.67	2.83	3.10
256	111,512	2.40	2.53	2.81

ordering. For instance for $K = 32$, GP, cnHP and the proposed model respectively yield 3.08, 3.31, and 3.39 times improvement over the original ordering.

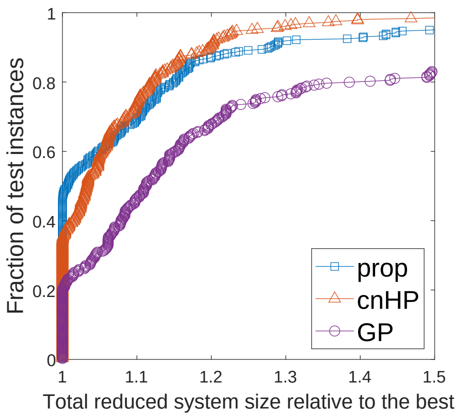
Table 5.2 presents the improvement gained by using the proposed model over the original and baseline models. The values indicate the total reduced system size of each model, all of which are normalized with respect to the result obtained by the proposed model. As can be seen in the Table, the proposed model outperforms the baseline models especially after 16 partitions. Furthermore, as K increases, the degree of superiority increases as well. The partitioning performance of proposed model is 1.10, 1.10, 1.13, 1.15, 1.16, 1.17 times better than the performance of GP; whereas it is 0.98, 1.01, 1.05, 1.07, 1.10, 1.11 times better than the performance of cnHP for $K = 8, 16, 32, 64, 128, 256$ parts, respectively.

Table 5.2: Averages of total reduced system size in upper and lower stSPIKE obtained by the partitioning models normalized with respect to the proposed HP model.

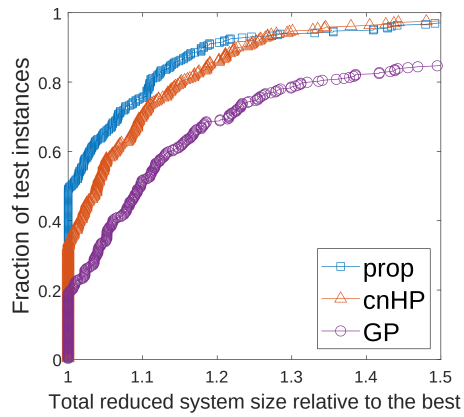
K	org	GP	cnHP	prop
8	3.94	1.10	0.98	1.00
16	3.64	1.10	1.01	1.00
32	3.49	1.13	1.05	1.00
64	3.33	1.15	1.07	1.00
128	3.10	1.16	1.10	1.00
256	2.81	1.17	1.11	1.00

Figure 5.8 provides the performance profiles comparing GP, cnHP and the proposed model in terms of the total reduced system size. As seen in the figure, the proposed model outperforms the baseline algorithms in terms of the reduced system size in majority of the test instances. Furthermore, the performance gap between the proposed model and the baseline models increases significantly with increasing K.

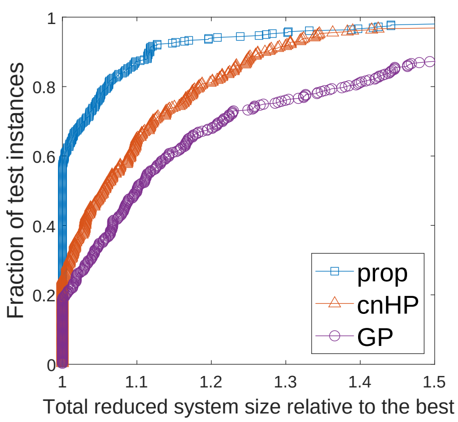
Table 5.3 shows the detailed comparison of the performance of the partitioning models in terms of total reduced system size as averages of different matrix kinds. All values are normalized with respect to the size of A (m). The last row gives the results as overall average. For instance for $K = 32$, total reduced system size obtained by GP, cnHP and the proposed model are 0.218, 0.203, 0.192 of the size of the original system, respectively.



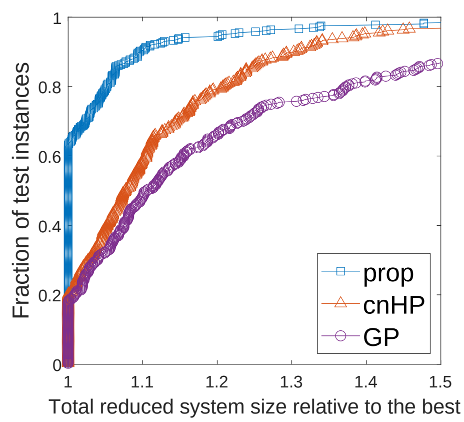
(a) $K = 8$



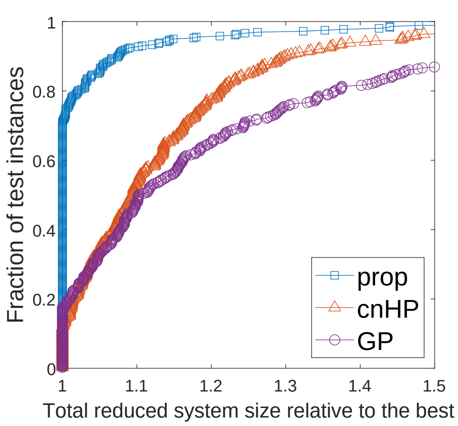
(b) $K = 16$



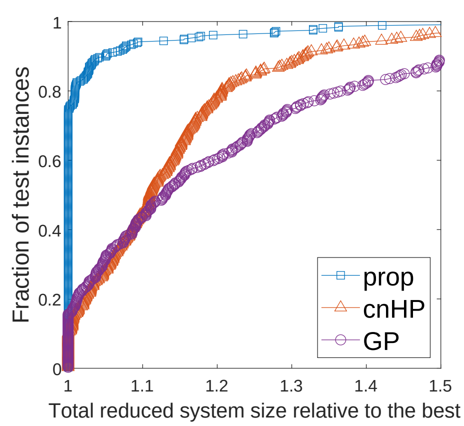
(c) $K = 32$



(d) $K = 64$



(e) $K = 128$



(f) $K = 256$

Figure 5.8: Performance profiles that compare GP, cnHP and the proposed HP model in terms of the total reduced system size in lower and upper stSPIKE.

Table 5.3: Averages of total reduced system size in lower and upper triangular stSPIKE normalized with respect to the number of rows.

kind	$K = 8$			$K = 16$			$K = 32$			$K = 64$			$K = 128$			$K = 256$		
ID	GP	cnHP	prop	GP	cnHP	prop	GP	cnHP	prop	GP	cnHP	prop	GP	cnHP	prop	GP	cnHP	prop
1	0.074	0.073	0.076	0.124	0.125	0.128	0.188	0.192	0.190	0.272	0.278	0.271	0.380	0.389	0.370	0.516	0.529	0.495
2	0.105	0.104	0.110	0.146	0.152	0.147	0.189	0.201	0.185	0.230	0.252	0.224	0.279	0.309	0.267	0.331	0.376	0.316
3	0.128	0.093	0.091	0.203	0.148	0.141	0.306	0.201	0.183	0.414	0.255	0.228	0.518	0.320	0.283	0.618	0.396	0.355
4	0.154	0.141	0.137	0.234	0.217	0.214	0.334	0.312	0.300	0.452	0.426	0.400	0.597	0.564	0.515	0.759	0.718	0.644
5	0.072	0.075	0.075	0.119	0.122	0.119	0.185	0.191	0.183	0.271	0.280	0.262	0.382	0.394	0.366	0.520	0.537	0.491
6	0.045	0.044	0.045	0.073	0.070	0.071	0.112	0.107	0.105	0.165	0.157	0.153	0.234	0.225	0.214	0.325	0.311	0.296
7	0.109	0.112	0.110	0.173	0.181	0.174	0.256	0.273	0.252	0.354	0.376	0.347	0.488	0.520	0.473	0.641	0.691	0.621
8	0.116	0.078	0.089	0.180	0.144	0.141	0.303	0.260	0.240	0.407	0.362	0.315	0.530	0.481	0.405	0.669	0.605	0.494
9	0.756	0.531	0.492	0.993	0.745	0.665	1.218	0.986	0.838	1.397	1.236	0.997	1.575	1.460	1.146	1.713	1.654	1.267
10	0.097	0.064	0.071	0.141	0.104	0.108	0.195	0.157	0.161	0.274	0.227	0.232	0.369	0.313	0.315	0.483	0.409	0.407
11	0.090	0.089	0.085	0.140	0.139	0.133	0.201	0.208	0.193	0.281	0.290	0.264	0.376	0.390	0.350	0.490	0.506	0.450
12	0.073	0.065	0.063	0.102	0.091	0.089	0.140	0.129	0.119	0.188	0.172	0.158	0.250	0.229	0.209	0.321	0.294	0.267
13	0.023	0.021	0.024	0.038	0.035	0.039	0.060	0.057	0.058	0.089	0.085	0.086	0.130	0.127	0.125	0.187	0.183	0.181
14	0.141	0.142	0.138	0.235	0.234	0.225	0.361	0.366	0.338	0.512	0.524	0.479	0.697	0.730	0.654	0.904	0.950	0.839
15	0.187	0.158	0.164	0.271	0.235	0.237	0.350	0.313	0.299	0.426	0.390	0.356	0.504	0.465	0.408	0.574	0.538	0.463
16	0.111	0.101	0.104	0.172	0.151	0.156	0.252	0.224	0.210	0.343	0.310	0.280	0.437	0.397	0.359	0.581	0.519	0.460
All	0.098	0.087	0.089	0.150	0.138	0.136	0.218	0.203	0.192	0.297	0.278	0.259	0.393	0.371	0.338	0.504	0.480	0.432

Figure 5.9 illustrates the averages of total reduced system size normalized with respect to the size of A as a comparison of different matrix kinds. As can be seen in the figure, some kinds such as model reduction (Kind ID = 9) and materials (Kind ID = 14) have larger averages of total reduced system size.

5.3.2 In-Block Reordering Quality

The proposed in-block reordering method considers the nonzeros in both lower and upper triangular parts to obtain a balanced ordering. We compare its performance with the methods that consider the nonzeros only in the lower or upper triangular parts, namely `incr_L` and `decr_U`, respectively, as described in Section 5.2.2.

Table 5.4 shows the average improvements gained by using the proposed and baseline partitioning models against the original ordering in terms of the total number of nonzeros in the reduced systems. For both methods, including the original ordering, the coefficient matrix is assumed to be partitioned and accordingly partially ordered by using the proposed HP model beforehand. The results for each K are given as averages of all instances in the dataset. In the table, we use the notation `rs_nnz()` as a function of total number of nonzeros in the reduced system obtain by a model. `rs_nnz`(org) means the total number of nonzeros in the reduced system obtained by the original ordering, after applying the HP partitioning. For instance for $K = 32$, `incr_L`, `decr_U` and the proposed method respectively yield 20.9, 17.7, and 30.7 times improvement over the original ordering. As can be observed from the table, the proposed method yields a significant improvement over the original ordering and outperforms the baseline methods. Although the improvement of the proposed method against the original ordering tends to degrade with increasing K, it is expected since there are fewer rows per block and hence there is less room for improvement.

Table 5.5 shows the detailed comparison of the performance of the partitioning models in terms of total number of nonzeros in the reduced systems as averages of different matrix kinds. All values are normalized with respect to the number of

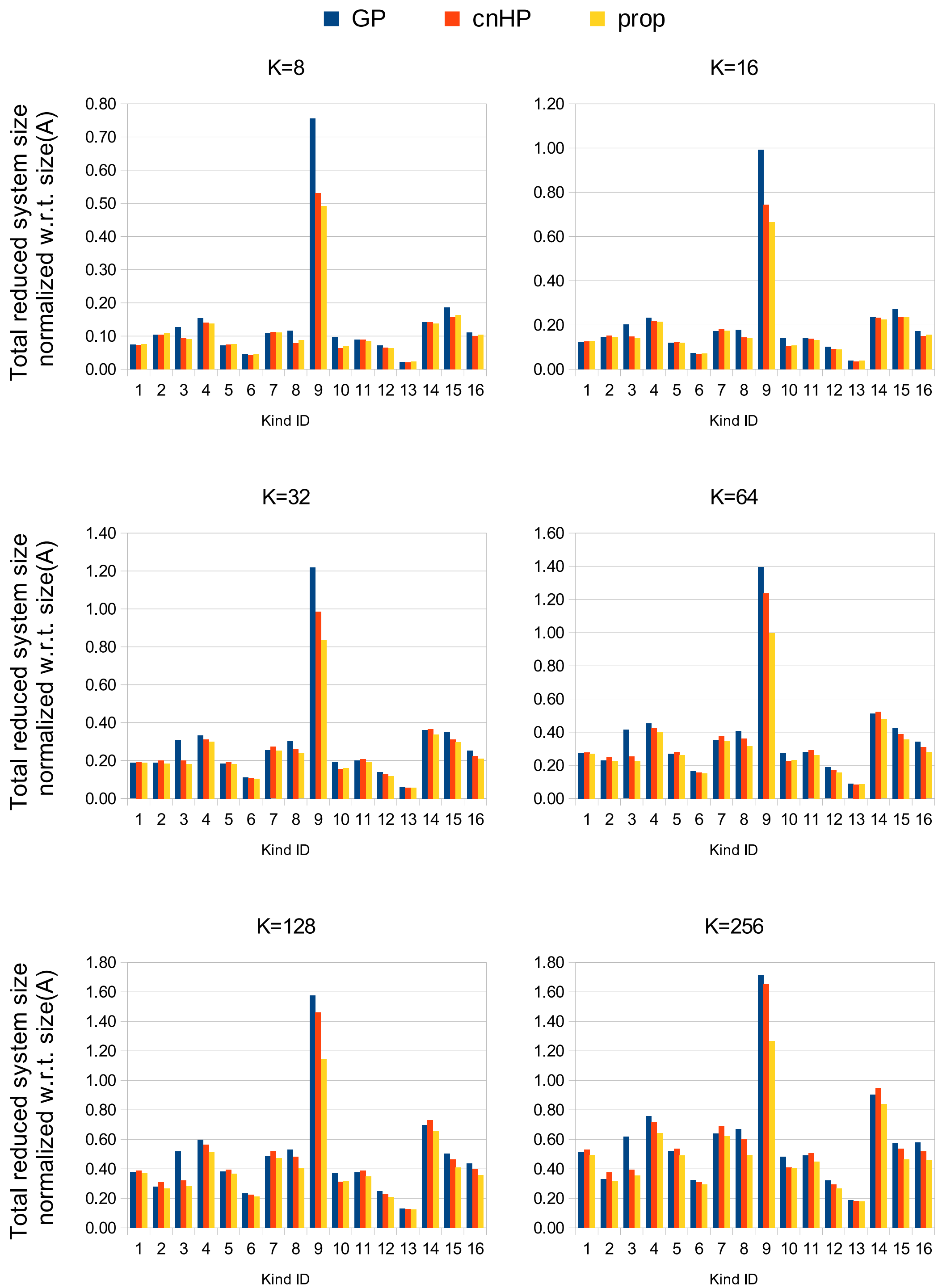


Figure 5.9: Total reduced system size normalized with respect to the coefficient matrix size (m) as averages of different matrix kinds.

Table 5.4: Improvement rates in terms of total number of nonzeros in the reduced systems (`rs_nnz`) in upper and lower stSPIKE, as ratios with respect to the original ordering (after partitioned with the proposed HP model), i.e. the ratio of rs_nnz(org) / rs_nnz(model) where model $\in$ {`incr_L`, `decr_U`, proposed}.

		improvement over org		
K	rs_nnz(org)	`incr_L`	`decr_U`	prop.
8	150,848	36.7	28.8	67.1
16	263,505	29.9	24.0	48.3
32	371,600	20.9	17.7	30.7
64	462,834	14.5	12.5	19.7
128	528,515	10.4	9.4	13.4
256	586,483	7.4	6.8	9.0

nonzeros in A. The last row gives the results as overall average. As can be seen in this table, total number of nonzeros in the reduced systems is much smaller than the nonzero count of the original system. For instance for $K = 32$, total number of nonzeros in the reduced systems obtained by `incr_L`, `decr_U` and the proposed model are 0.013, 0.015, 0.009 of the nonzero count of the original system, respectively. This verifies the feasibility of the dmpILU algorithm by applying the proposed partitioning.

Table 5.5: Averages of total nonzero counts in the off-diagonal blocks of $\widehat{S}^L$ and $\widehat{S}^U$ normalized with respect to the nonzero count of the coefficient matrix.

kind	K=8			K=16			K=32			K=64			K=128			K=256		
ID	incr_L	decr_U	prop	incr_L	decr_U	prop	incr_L	decr_U	prop	incr_L	decr_U	prop	incr_L	decr_U	prop	incr_L	decr_U	prop
1	0.000	0.000	0.000	0.001	0.001	0.001	0.003	0.003	0.003	0.008	0.008	0.007	0.016	0.016	0.015	0.031	0.031	0.030
2	0.010	0.013	0.009	0.015	0.018	0.013	0.017	0.020	0.016	0.025	0.029	0.023	0.033	0.036	0.030	0.042	0.045	0.040
3	0.015	0.051	0.005	0.036	0.091	0.017	0.069	0.130	0.040	0.102	0.166	0.068	0.134	0.187	0.096	0.176	0.224	0.133
4	0.006	0.009	0.003	0.010	0.014	0.006	0.018	0.023	0.013	0.031	0.037	0.024	0.049	0.055	0.039	0.071	0.076	0.061
5	0.003	0.003	0.001	0.007	0.009	0.002	0.016	0.018	0.005	0.022	0.024	0.009	0.022	0.024	0.012	0.038	0.040	0.025
6	0.000	0.001	0.000	0.001	0.001	0.001	0.002	0.002	0.001	0.004	0.005	0.002	0.008	0.009	0.004	0.013	0.015	0.009
7	0.001	0.001	0.001	0.003	0.003	0.003	0.006	0.007	0.006	0.014	0.014	0.013	0.027	0.028	0.026	0.052	0.052	0.050
8	0.005	0.005	0.001	0.016	0.023	0.005	0.053	0.089	0.029	0.105	0.177	0.075	0.164	0.252	0.131	0.271	0.355	0.212
9	0.225	0.220	0.195	0.489	0.493	0.434	0.837	0.848	0.750	1.205	1.226	1.087	1.569	1.609	1.433	1.844	1.887	1.689
10	0.011	0.009	0.005	0.013	0.011	0.007	0.025	0.023	0.015	0.041	0.040	0.027	0.066	0.067	0.047	0.099	0.106	0.076
11	0.001	0.001	0.001	0.001	0.001	0.001	0.003	0.003	0.002	0.006	0.006	0.005	0.010	0.010	0.009	0.018	0.018	0.016
12	0.002	0.002	0.002	0.002	0.002	0.002	0.004	0.003	0.003	0.006	0.006	0.005	0.010	0.009	0.008	0.015	0.014	0.013
13	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.001	0.000	0.001	0.001	0.001	0.002	0.002	0.001	0.004	0.004	0.002
14	0.001	0.002	0.001	0.005	0.006	0.004	0.011	0.014	0.010	0.026	0.030	0.023	0.053	0.057	0.049	0.101	0.103	0.096
15	0.014	0.019	0.013	0.035	0.044	0.033	0.079	0.103	0.069	0.138	0.179	0.127	0.200	0.284	0.189	0.256	0.320	0.243
16	0.004	0.007	0.001	0.012	0.020	0.007	0.018	0.027	0.010	0.034	0.048	0.021	0.044	0.059	0.031	0.062	0.069	0.046
All	0.003	0.004	0.002	0.006	0.008	0.004	0.013	0.015	0.009	0.022	0.025	0.016	0.034	0.038	0.027	0.054	0.058	0.044

Figure 5.10 illustrates the averages of total number of nonzeros in reduced systems normalized with respect to the nonzero count of A as a comparison of different matrix kinds. As can be seen in the figure, some matrix kinds such as economic (Kind ID=3), model reduction (Kind ID=9), chemical process simulation (Kind ID=10) and weighted graph (Kind ID=15) have proportionally larger averages of total number of nonzeros in the reduced systems, so their sequential computational overhead are expected to be relatively high.

5.3.3 Parallel Scalability

Parallel experiments are performed on Sariyer cluster of UHEM [109] using up to 160 cores over 4 distributed nodes where each node contains 40 cores (two Intel Xeon Gold 6148 CPUs).

We implement an MPI+OpenMP hybrid parallel dmpILU algorithm to demonstrate the effectiveness of using stSPIKE and the proposed model. We refer the proposed HP and in-block reordering model (Section 5.2) applied to dmpILU as the proposed model throughout this section. The number of MPI processes is the same as the number of parts (K) in a partition. For the experiments of dmpILU, we assign 8 processes per node and 5 threads per process. Therefore, we conduct parallel experiments for dmpILU using 1, 2, and 4 nodes corresponding to 40, 80, and 160 cores and $K = 8, 16$ and 32 parts (processes), respectively.

For comparing the performance of dmpILU, we also implemented a multithreaded ILU ($mtILU$) in which we use by using the multithreaded sparse triangular system solver (mkl_sparse_d_trsm) of Intel MKL [110]. As a baseline, we obtain the results of mtILU on 40 threads/cores (1 node) by using the GP reordering. We run both dmpILU and mtILU for 100 iterations with the initial guess $x = [0, \ldots, 0]^T$ and the right-hand side vector $f = [1/m, 2/m, \ldots, 1]^T$.

We tested the parallel scalability of dmpILU for a subset of the dataset since the provided core hours on the HPC platform are limited. From the dataset,

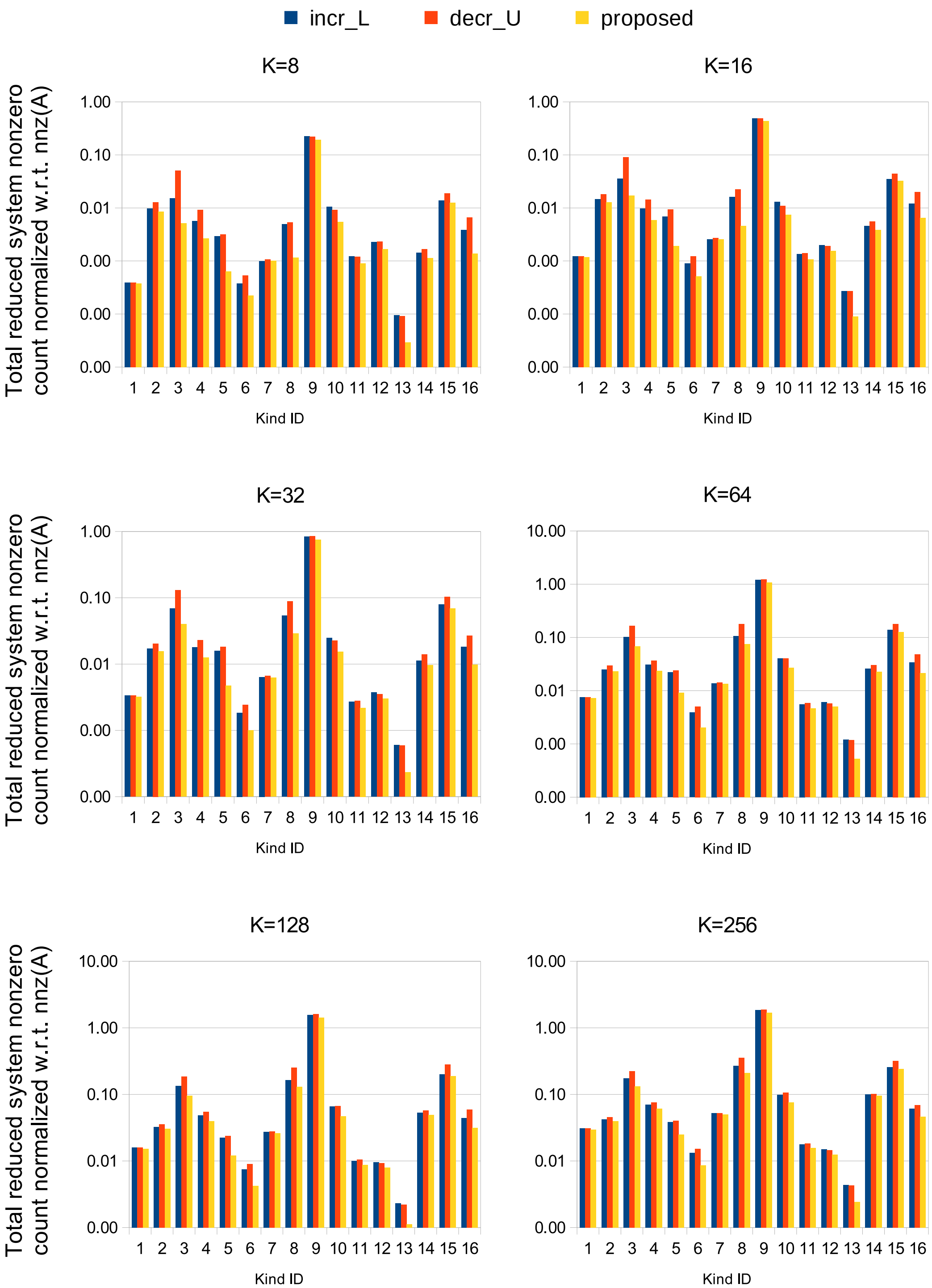


Figure 5.10: Total number of nonzeros in the lower and upper triangular reduced systems normalized with respect to the number of nonzeros in the coefficient matrix as averages of different matrix kinds.

Table 5.6: The properties of matrices to conduct parallel experiments. Runtime results of mtILU are taken on 40 cores for 100 iterations.

Matrix	Kind ID	Sym	Size	Nnz	mtILU time (s)
msdoor	1	✓	415,863	19,173,163	9.8
atmosmodl	5		1,489,752	10,319,760	13.4
test1	4		392,908	9,447,535	6.5
thermal2	13	✓	1,228,045	8,580,313	12.6
G3_circuit	2	✓	1,585,478	7,660,826	13.9
cage13	15		445,315	7,479,343	6.2

we considered the matrices with at most 2,000,000 rows due to the memory constraints. We selected six different kinds, namely structural (Kind ID = 1), circuit simulation (Kind ID = 2), semiconductor device (Kind ID = 4), computational fluid dynamics (Kind ID = 5), thermal (Kind ID = 13), and weighted graph (Kind ID = 15). Then we select the instances that has the most number of nonzeros within each of these kinds. The properties of those matrices are shown in Table 5.6, sorted in decreasing order of their nonzero counts. The last column of the table shows the runtime of mtILU after 100 iterations.

Figure 5.11 shows the results of the scaling experiments as speedup curves of dmpILU using GP, cnHP and the proposed model. As seen in the figure, the proposed model highly increases the scalability of dmpILU so that it scales up to 160 cores on all instances. Furthermore, the proposed model significantly outperforms the baseline GP and cnHP models for all of the test instances. dmpILU with the proposed model achieves up to 70.2 speedup on 160 cores over mtGS on 40 cores.

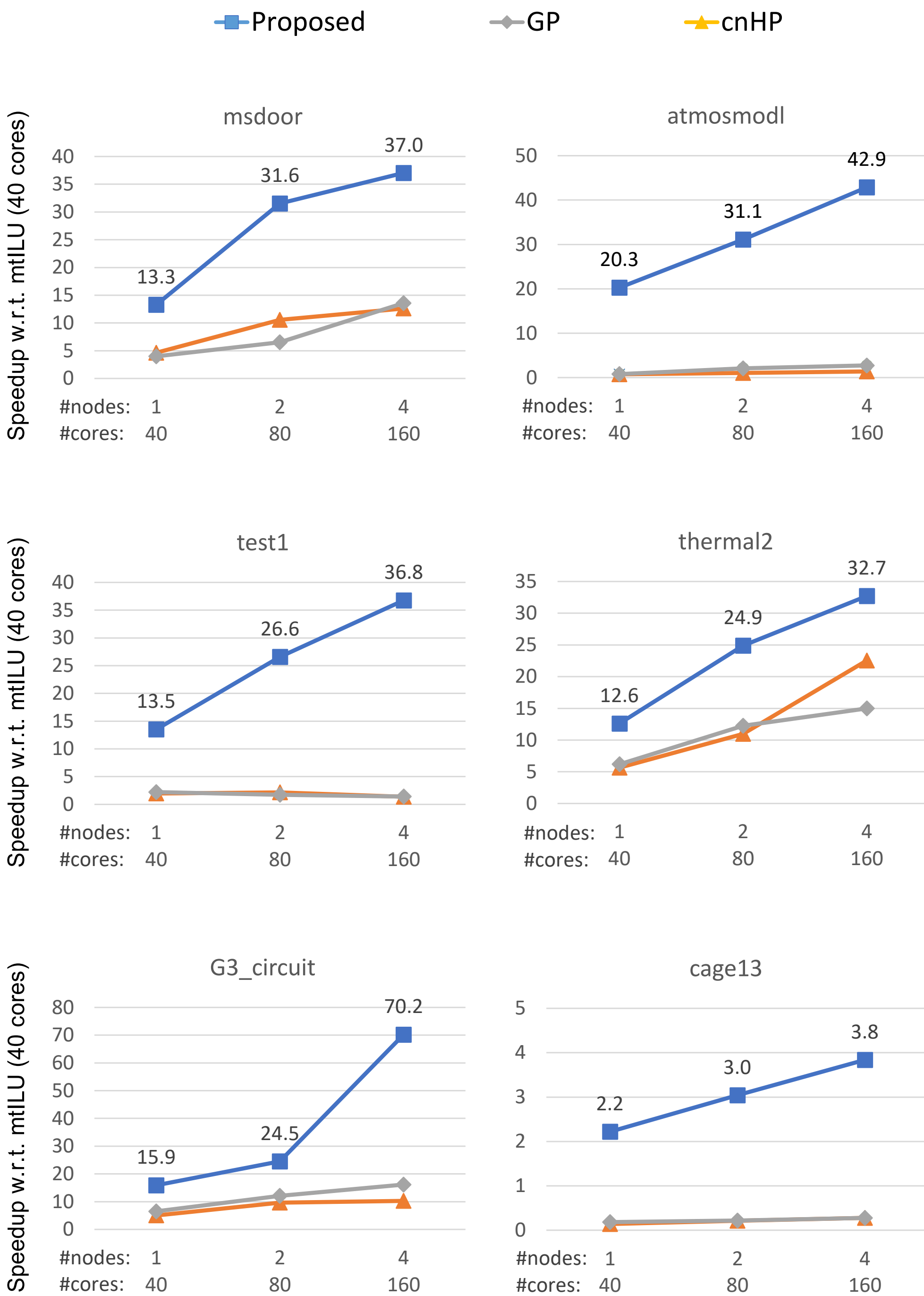


Figure 5.11: Speedup curves of dmpILU with GP, cnHP and the proposed model (for $K=8, 16$, and 32) relative to mtILU on 1 node (40 cores).

5.4 Summary

We propose a distributed-memory parallel ILU(0) solution algorithm (dmpILU) by using stSPIKE for solving the lower and upper triangular systems. The reduced systems in both lower and upper stSPIKE constitute the sequential bottleneck of dmpILU. For alleviating this bottleneck, we propose a two-phase partitioning and reordering model. The first phase is a novel hypergraph partitioning model whose partitioning objective encapsulates the minimization of the total size of lower and upper triangular reduced systems. For this purpose, we exploit the recursive bipartitioning framework by introducing new types of net anchoring and splitting schemes. The second phase is an in-block reordering method for minimizing total number of nonzeros in the lower and upper triangular reduced systems. Experiments demonstrate that the proposed hypergraph partitioning model indeed decreases total reduced system size with respect to the baseline models. The proposed in-block reordering method yields a high benefit in terms of decreasing the total number of nonzeros in the reduced systems. Parallel experiments also demonstrate that using the proposed partitioning and reordering model improves the scalability of dmpILU.

Chapter 6

Hypergraph Partitioning for Scalable Sparse Tensor Decomposition [1]

The success of medium-grain CPD-ALS algorithm adopting the multi-dimensional cartesian tensor partitioning is due to its nice upper bounds on communication overheads. However, this model does not utilize the sparsity pattern of the tensor to reduce the total communication volume. Our objective is to fill this literature gap.

We describe the communication volume requirement of a given cartesian partition of a tensor in Section 6.1. We propose a novel HP model, CartHP, for minimizing the total communication volume of medium-grain CPD-ALS in Section 6.2. For ease of understanding, we first discuss cartesian partition and CartHP for a three-mode tensor. Then we discuss the extension of the proposed model to more than three modes. We also provide a discussion on the direct extension of CBHP for tensors and its deficiency in Section 6.3. Section 6.4 provides the experimental results and Section 6.5 summarizes. Throughout this chapter, we denote tensors, matrices and vectors by calligraphic ($\mathcal{X}$), bold capital ($\mathbf{A}$) and

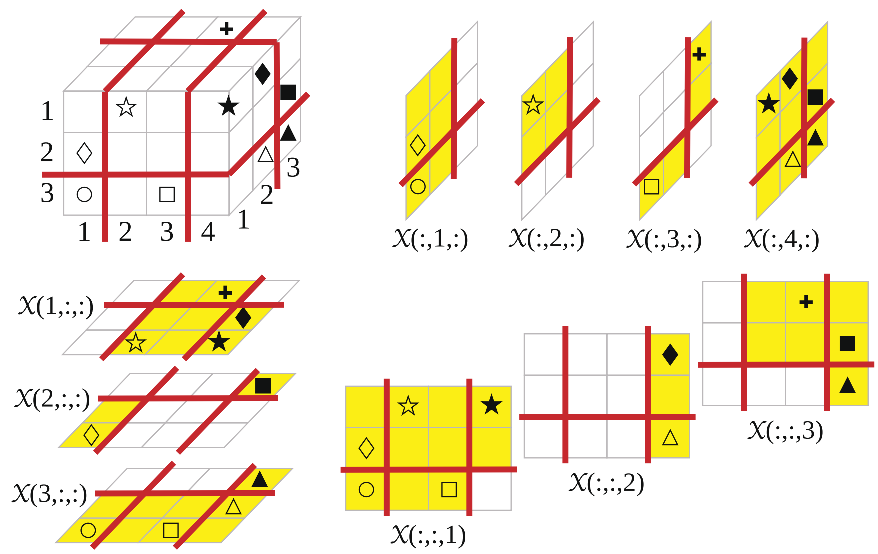


Figure 6.1: A 3D cartesian partition of a $3 \times 4 \times 3$ tensor for a $2 \times 3 \times 2$ virtual processor mesh.

bold lowercase (**a**) letters, respectively.

6.1 Communication Volume Requirement

A given cartesian partition of the tensor divides each slice/fiber into subslices/subfibers, each of which is owned by a different processor. We denote any (sub)tensor γ owned by a set α of processor(s) by γ_α. For instance, $\mathcal{X}(i,:,:)_{q,r,s}$ and $\mathcal{X}(i,j,:)_{q,r,s}$ respectively denote the subslice of $\mathcal{X}(i,:,:)$ and the subfiber of $\mathcal{X}(i,j,:)$ which are owned by processor $p_{q,r,s}$. Similarly, $\mathcal{X}(i,:,:)_{:,r,:}$ denotes the subslice of $\mathcal{X}(i,:,:)$ owned by processor layer $p_{:,r,:}$. To differentiate the subslices owned by a single processor from those owned by multiple processors, we refer to the former ones as *unshared* subslices. A (sub)slice/(sub)fiber containing at least one nonzero element is called a *nonzero* (sub)slice/(sub)fiber. Figure 6.1 displays a cartesian partition of a $3{\times}4{\times}3$ tensor for a $2{\times}3{\times}2$ virtual processor mesh and the respective divisions of slices into subslices induced by this partition. In this figure, each tensor nonzero is denoted by a different symbol and each nonzero subslice is highlighted. For example, slice $\mathcal{X}(1,:,:)$ contains 4 nonzero elements and 3 nonzero unshared subslices.

Let Z_i^A, Z_j^B and Z_k^C respectively denote the sets of nonzero unshared subslices of $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$. For the example given in Figure 6.1, $Z_1^A = \{\mathcal{X}(1,:,:)_{1,2,1}, \mathcal{X}(1,:,:)_{1,2,2}, \mathcal{X}(1,:,:)_{1,3,1}\}$. We assume that each slice contains at least one nonzero, hence, these sets are nonempty. In the first phase of medium-grain CPD-ALS, only the processors that own a subslice in Z_i^A produce partial results for $\hat{\mathbf{A}}(i,:)$. Similarly in the second and third phases, only the processors that own a subslice in Z_j^B and Z_k^C produce partial results for $\hat{\mathbf{B}}(j,:)$ and $\hat{\mathbf{C}}(k,:)$, respectively.

We assume that each factor-matrix row is assigned to a processor which owns a nonzero subslice in the corresponding slice. We refer to this assumption as the *consistency condition* for the correctness of our hypergraph model to be proposed in Section 6.2. Let $\hat{\mathbf{A}}(i,:)$ be assigned to a processor, say p, that owns a nonzero subslice in Z_i^A. Each of the other processors that own a nonzero subslice in Z_i^A sends a partial result for $\hat{\mathbf{A}}(i,:)$ to p in the fold step. Then, the communication volume regarding the fold operation on $\hat{\mathbf{A}}(i,:)$ amounts to $(|Z_i^A|-1)F$. In a dual manner, p sends the updated row $\mathbf{A}(i,:)$ to these processors in the expand step, incurring a communication of volume $(|Z_i^A|-1)F$ again. Since the same volume of communication is incurred regarding the expand operation on $\mathbf{A}(i,:)$ and the fold operation on $\hat{\mathbf{A}}(i,:)$, we only consider the one regarding $\hat{\mathbf{A}}(i,:)$ and formulate it as

$$vol_i^A = (|Z_i^A| - 1)F. \tag{6.1}$$

Then, the total volume in the first phase is the sum of the volumes regarding the rows of $\hat{\mathbf{A}}$, that is,

$$vol^A = \sum_{i=1}^{I} vol_i^A = (\sum_{i=1}^{I}(|Z_i^A| - 1))F.$$

With similar discussions for the second and third phases, we obtain $vol^B = (\sum_{j=1}^{J}(|Z_j^B| - 1))F$ and $vol^C = (\sum_{k=1}^{K}(|Z_k^C| - 1))F$. Then, $vol^A + vol^B + vol^C$ gives the overall total volume per iteration.

In Figure 6.1, the volume of communication regarding $\hat{\mathbf{A}}(1,:)$ is $vol_1^A = (|Z_1^A| -$

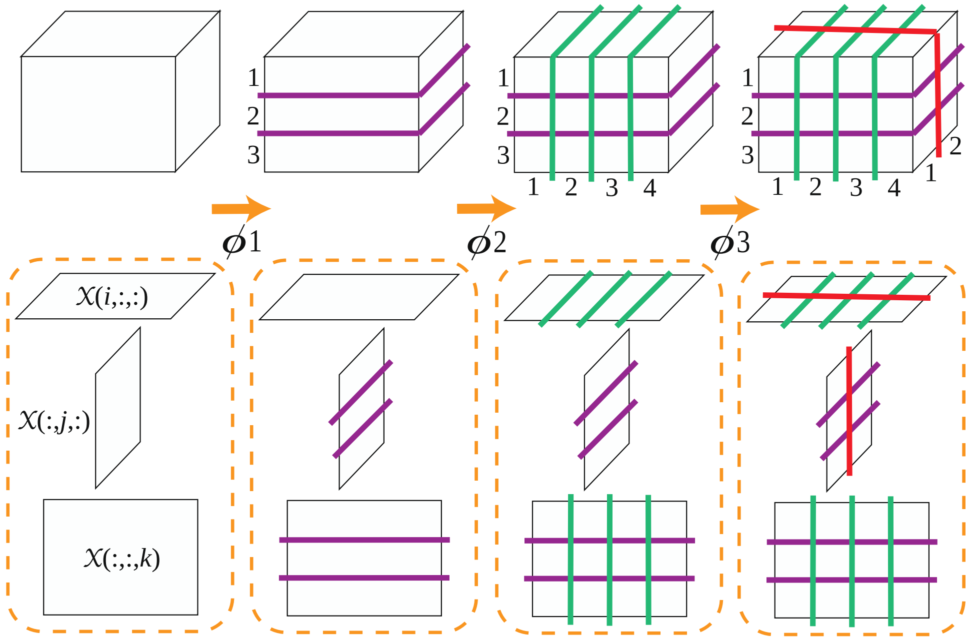


Figure 6.2: Slice chunks obtained in phases $\phi1$, $\phi2$ and $\phi3$ and (sub)subslices of $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ divided by these chunks.

$1)F = 2F$. The total volume in the first phase is $vol^A = (2+1+3)F = 6F$ and the overall total volume is $(6+5+7)F = 18F$.

6.2 CartHP: Proposed HP Model

For a given tensor $\mathcal{X}$ and a $Q{\times}R{\times}S$ virtual mesh of processors, CartHP contains partitioning phases $\phi1$, $\phi2$ and $\phi3$, in which hypergraphs $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$ are constructed with vertex sets representing the horizontal, lateral and frontal slices of $\mathcal{X}$, respectively. In $\phi1$, CartHP obtains a Q-way partition of $\mathcal{H}^A$ and uses this partition to reorder the horizontal slices to form Q horizontal chunks. These horizontal chunks divide each lateral and frontal slice into Q subslices along mode 1. Similarly in $\phi2$, CartHP obtains an R-way partition of $\mathcal{H}^B$ and uses this partition to reorder the lateral slices to form R lateral chunks. These lateral chunks divide each horizontal slice into R subslices along mode 2 and each frontal subslice into R subsubslices along mode 2. Note that each frontal slice has $Q{\times}R$ subsubslices

Algorithm 8 CartHP

Input: tensor $\mathcal{X}$, 3D processor mesh size $Q \times R \times S$, imbalance ratios $\epsilon_1, \epsilon_2, \epsilon_3$.

1: $\phi1(\mathcal{X}, Q, \epsilon_1)$ ▷ obtains Q horizontal chunks
2: $\phi2(\mathcal{X}, R, \epsilon_2)$ ▷ obtains R lateral chunks
3: $\phi3(\mathcal{X}, S, \epsilon_3)$ ▷ obtains S frontal chunks
4: **for each** subtensor $\mathcal{X}_{q,r,s}$ **do**
5: Assign $\mathcal{X}_{q,r,s}$ to processor $p_{q,r,s}$

at the end of $\phi2$. Finally in $\phi3$, CartHP obtains an S-way partition of $\mathcal{H}^C$ and uses this partition to reorder the frontal slices to form S frontal chunks. These frontal chunks divide each horizontal and lateral subslice into S subsubslices along mode 3. Note that each horizontal and lateral slice have $R \times S$ and $Q \times S$ subsubslices at the end of $\phi3$, respectively. Figure 6.2 illustrates a tensor which is partitioned by CartHP for a $2 \times 4 \times 3$ virtual processor mesh and three sample slices along different modes.

Algorithm 8 displays the basic layout of CartHP. Here, we abuse the notation for simplicity and use the same symbol $\mathcal{X}$ for the original tensor (line 1) and the reordered tensors (lines 2-3). Consequently, each subtensor $\mathcal{X}_{q,r,s}$ (line 4) is the intersection of the respective chunks of the reordered tensor.

Algorithm 9 displays phase $\phi1$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^A = (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$ to obtain Q horizontal chunks (lines 15-17). In $\mathcal{H}^A$, $\mathcal{V}^A = \{v_1^A, \ldots, v_I^A\}$ contains a vertex v_i^A for each horizontal slice $\mathcal{X}(i,:,:)$. $\mathcal{N}^B$ contains a net n_j^B for each nonzero lateral slice $\mathcal{X}(:,j,:)$, whereas $\mathcal{N}^C$ contains a net n_k^C for each nonzero frontal slice $\mathcal{X}(:,:,k)$. Since all slices are assumed to have at least one nonzero element, $\mathcal{N}^B = \{n_1^B, \ldots, n_J^B\}$ and $\mathcal{N}^C = \{n_1^C, \ldots, n_K^C\}$. Net n_j^B connects vertex v_i^A if the intersection of $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,j,:)$ contains at least one nonzero (lines 7-8). Similarly, n_k^C connects v_i^A if the intersection of $\mathcal{X}(i,:,:)$ and $\mathcal{X}(:,:,k)$ has at least one nonzero (lines 11-12). Each vertex v_i^A is assigned a single weight $w(v_i^A) = nnz(\mathcal{X}(i,:,:))$ (lines 2-3). Here, $nnz(\cdot)$ denotes the number of nonzeros of the given (sub)tensor. Then, a Q-way partition Π^A of $\mathcal{H}^A$ is obtained (line 14).

Algorithm 9 $\phi 1(\mathcal{X}, Q, \epsilon_1)$

$\mathcal{V}^A \leftarrow \{v_1^A, \dots, v_I^A\}$
for each horizontal slice $\mathcal{X}(i,:,:)$ **do**
    $w(v_i^A) \leftarrow nnz(\mathcal{X}(i,:,:))$
$\mathcal{N}^B \leftarrow \mathcal{N}^C \leftarrow \emptyset$
for each lateral slice $\mathcal{X}(:,j,:)$ **do**
    $\mathcal{N}^B \leftarrow \mathcal{N}^B \cup \{n_j^B\}$ with $Pins(n_j^B) = \emptyset$
    for each nonzero fiber $\mathcal{X}(i,j,:)$ **do**
        $Pins(n_j^B) \leftarrow Pins(n_j^B) \cup \{v_i^A\}$
for each frontal slice $\mathcal{X}(:,:,k)$ **do**
    $\mathcal{N}^C \leftarrow \mathcal{N}^C \cup \{n_k^C\}$ with $Pins(n_k^C) = \emptyset$
    for each nonzero fiber $\mathcal{X}(i,:,k)$ **do**
        $Pins(n_k^C) \leftarrow Pins(n_k^C) \cup \{v_i^A\}$
$\mathcal{H}^A \leftarrow (\mathcal{V}^A, \mathcal{N}^B \cup \mathcal{N}^C)$
$\Pi^A = \{\mathcal{V}_1^A, \dots, \mathcal{V}_Q^A\} \leftarrow \text{HP}(\mathcal{H}^A, Q, \epsilon_1)$
for $q \leftarrow 1$ **to** Q **do**
    for each $v_i^A \in \mathcal{V}_q^A$ **do**
        Assign slice $\mathcal{X}(i,:,:)$ to chunk $\mathcal{X}_{q,:,:}$

Algorithm 10 displays phase $\phi 2$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^B = (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$ to obtain R lateral chunks (lines 15-17). In $\mathcal{H}^B$, $\mathcal{V}^B = \{v_1^B, \dots, v_J^B\}$ contains a vertex v_j^B for each lateral slice $\mathcal{X}(:,j,:)$. $\mathcal{N}^A$ contains a net n_i^A for each nonzero horizontal slice $\mathcal{X}(i,:,:)$, that is, $\mathcal{N}^A = \{n_i^A, \dots, n_I^A\}$. Net n_i^A connects vertex v_j^B if the intersection of $\mathcal{X}(:,j,:)$ and $X(i,:,:)$ contains at least one nonzero (lines 7-8). The nets in $\mathcal{N}^A$ are similar to those in $\phi 1$ since horizontal slices are not yet divided into subslices. Frontal slices, on the other hand, have been divided into Q subslices along mode 1 by the horizontal chunks formed in $\phi 1$. Instead of a single net, each frontal slice $\mathcal{X}(:,:,k)$ is represented by a number of nets as many as the number of its nonzero subslices. $\mathcal{N}^C$ contains a net $n_{k(q)}^C$ for each nonzero subslice $\mathcal{X}(:,:,k)_{q,:,:}$ (lines 9-10). We only include nets for nonzero subslices as the zero subslices do not incur any increase in the number of nonzero unshared subslices. Net $n_{k(q)}^C$ connects vertex v_j^B if the intersection of $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)_{q,:,:}$ contains at least one nonzero (lines 11-12). Since each slice $\mathcal{X}(:,j,:)$ contains Q subslices, each vertex v_j^B is assigned Q weights $w_q(v_j^B) = nnz(\mathcal{X}(:,j,:)_{q,:,:})$ for $q = 1, \dots, Q$ (lines 2-3). Then,

Algorithm 10 $\phi 2(\mathcal{X}, R, \epsilon_2)$

1: $\mathcal{V}^B \leftarrow \{v_1^B, \ldots, v_J^B\}$
2: **for each** lateral subslice $\mathcal{X}(:, j, :)_{q,:,:}$ **do**
3: $\quad w_q(v_j^B) \leftarrow nnz(\mathcal{X}(:, j, :)_{q,:,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^C \leftarrow \emptyset$
5: **for each** horizontal slice $\mathcal{X}(i, :, :)$ **do**
6: $\quad \mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_i^A\}$ with $Pins(n_i^A) = \emptyset$
7: $\quad$ **for each** nonzero fiber $\mathcal{X}(i, j, :)$ **do**
8: $\quad\quad Pins(n_i^A) \leftarrow Pins(n_i^A) \cup \{v_j^B\}$
9: **for each** nonzero frontal subslice $\mathcal{X}(:, :, k)_{q,:,:}$ **do**
10: $\quad \mathcal{N}^C \leftarrow \mathcal{N}^C \cup \{n_{k(q)}^C\}$ with $Pins(n_{k(q)}^C) = \emptyset$
11: $\quad$ **for each** nonzero subfiber $\mathcal{X}(:, j, k)_{q,:,:}$ **do**
12: $\quad\quad Pins(n_{k(q)}^C) \leftarrow Pins(n_{k(q)}^C) \cup \{v_j^B\}$
13: $\mathcal{H}^B \leftarrow (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$
14: $\Pi^B = \{\mathcal{V}_1^B, \ldots, \mathcal{V}_R^B\} \leftarrow$ MC-HP$(\mathcal{H}^B, R, \epsilon_2)$
15: **for** $r \leftarrow 1$ **to** R **do**
16: $\quad$ **for each** $v_j^B \in \mathcal{V}_r^B$ **do**
17: $\quad\quad$ Assign slice $\mathcal{X}(:, j, :)$ to chunk $\mathcal{X}_{:,r,:}$

an R-way partition Π^B of $\mathcal{H}^B$ is obtained by multi-constraint HP (MC-HP) (line 14).

Algorithm 11 displays phase $\phi 3$, in which we construct (lines 1-13) and partition (line 14) $\mathcal{H}^C = (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$ to obtain S frontal chunks (lines 15-17). In $\mathcal{H}^C$, $\mathcal{V}^C = \{v_1^C, \ldots, v_K^C\}$ contains a vertex v_k^C for each frontal slice $\mathcal{X}(:, :, k)$. As in $\phi 2$, each divided slice is represented by a number of nets as many as the number of its nonzero subslices. Note that each horizontal slice has been divided into R subslices along mode 2 in $\phi 2$, whereas each lateral slice has been divided into Q subslices along mode 1 in $\phi 1$. $\mathcal{N}^A$ contains a net $n_{i(r)}^A$ for each nonzero subslice $\mathcal{X}(i, :, :)_{:,r,:}$, whereas $\mathcal{N}^B$ contains a net $n_{j(q)}^B$ for each nonzero subslice $\mathcal{X}(:, j, :)_{q,:,:}$ (lines 5-6 and 9-10). Net $n_{i(r)}^A$ connects vertex v_k^C if the intersection of $\mathcal{X}(:, :, k)$ and $\mathcal{X}(i, :, :)_{:,r,:}$ contains at least one nonzero (lines 7-8). Similarly, $n_{j(q)}^B$ connects v_k^C if the intersection of $\mathcal{X}(:, :, k)$ and $\mathcal{X}(:, j, :)_{q,:,:}$ contains at least one nonzero (lines 11-12). Since each slice $\mathcal{X}(:, :, k)$ contains $Q \times R$ subsubslices, each vertex v_k^C is assigned $Q \times R$ weights $w_{q,r}(v_k^C) = nnz(\mathcal{X}(:, :, k)_{q,r,:})$ for $q = 1, \ldots, Q$

Algorithm 11 $\phi 3(\mathcal{X}, S, \epsilon_3)$

1: $\mathcal{V}^C \leftarrow \{v_1^C, \ldots, v_K^C\}$
2: **for each** frontal subslice $\mathcal{X}(:,:,k)_{q,r,:}$ **do**
3: $\quad w_{q,r}(v_k^C) \leftarrow nnz(\mathcal{X}(:,:,k)_{q,r,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^B \leftarrow \emptyset$
5: **for each** nonzero horizontal subslice $\mathcal{X}(i,:,:)_{:,r,:}$ **do**
6: $\quad \mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_{i(r)}^A\}$ with $Pins(n_{i(r)}^A) = \emptyset$
7: $\quad$ **for each** nonzero subfiber $\mathcal{X}(i,:,k)_{:,r,:}$ **do**
8: $\quad\quad Pins(n_{i(r)}^A) \leftarrow Pins(n_{i(r)}^A) \cup \{v_k^C\}$
9: **for each** nonzero lateral subslice $\mathcal{X}(:,j,:)_{q,:,:}$ **do**
10: $\quad \mathcal{N}^B \leftarrow \mathcal{N}^B \cup \{n_{j(q)}^C\}$ with $Pins(n_{j(q)}^C) = \emptyset$
11: $\quad$ **for each** nonzero subfiber $\mathcal{X}(:,j,k)_{q,:,:}$ **do**
12: $\quad\quad Pins(n_{j(q)}^B) \leftarrow Pins(n_{j(q)}^B) \cup \{v_k^C\}$
13: $\mathcal{H}^C \leftarrow (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$
14: $\Pi^C = \{\mathcal{V}_1^C, \ldots, \mathcal{V}_S^C\} \leftarrow$ MC-HP$(\mathcal{H}^C, S, \epsilon_3)$
15: **for** $s \leftarrow 1$ **to** S **do**
16: $\quad$ **for each** $v_k^C \in \mathcal{V}_s^C$ **do**
17: $\quad\quad$ Assign slice $\mathcal{X}(:,:,k)$ to chunk $\mathcal{X}_{:,:,s}$

and $r = 1, \ldots, R$ (lines 2-3). Then, an S-way partition Π^C of $\mathcal{H}^C$ is obtained by MC-HP (line 14).

All nets in the hypergraphs constructed in our model are assigned a cost of F. That is, $c(n) = F$ for each net n in $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$.

In partitioning $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$, the maximum allowed imbalance ratios are set to ϵ_1, ϵ_2 and ϵ_3, respectively. It can be shown that at the end of three partitoning phases, the number of nonzeros assigned to a processor is bounded above by

$$(1/P)nnz(\mathcal{X})(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3). \tag{6.2}$$

The derivation of equation (6.2) is given in Section 6.2.5.

Figure 6.3 illustrates an example for CartHP applied on a $4 \times 4 \times 3$ tensor $\mathcal{X}$ for a $2 \times 2 \times 2$ virtual mesh of processors. The vertices that represent horizontal, lateral and frontal slices are colored with purple, green and red, respectively. The same color encoding also applies to the nets in each phase. In ϕm, the

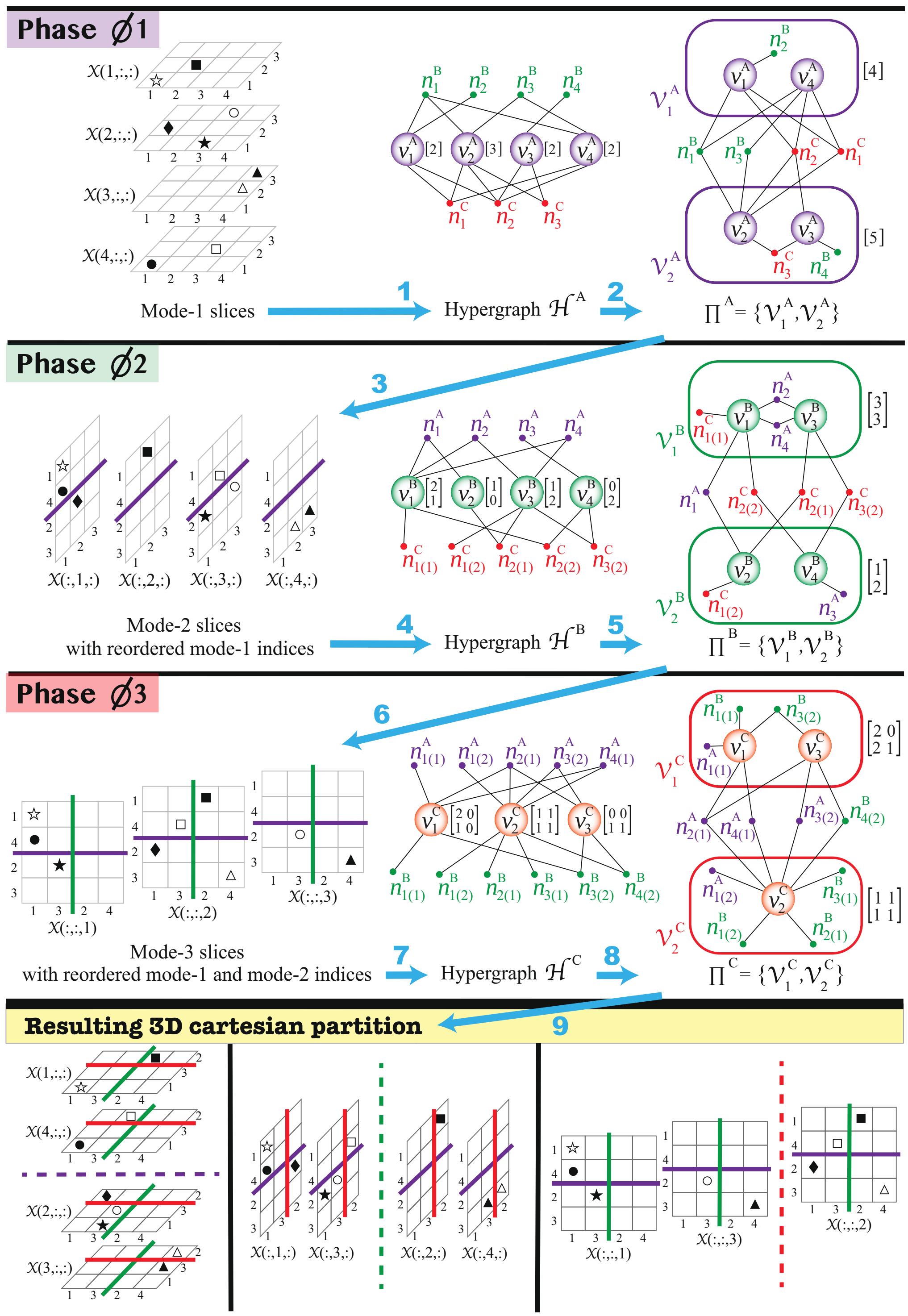


Figure 6.3: CartHP on a 4×4×3 tensor $\mathcal{X}$ for a 2×2×2 virtual mesh of processors. ϕ1: horizontal slices of $\mathcal{X}$. ϕ2: lateral slices of $\mathcal{X}$ with reordered mode-1 indices. ϕ3: frontal slices of $\mathcal{X}$ with reordered mode-1 and mode-2 indices. Bottom: slices of $\mathcal{X}$ reordered along all modes.

tensor is displayed in terms of mode-m slices. For each hypergraph, the array of weights associated to each vertex/part is displayed next to the corresponding vertex/part. For example, consider v_3^B in $\phi2$. Vertex v_3^B is connected by nets n_2^A and n_4^A due to nonzero fibers $\mathcal{X}(2,3,:)$ and $\mathcal{X}(4,3,:)$, respectively, and by nets $n_{1(2)}^C$, $n_{2(1)}^C$ and $n_{3(2)}^C$ due to nonzero subfibers $\mathcal{X}(:,3,1)_{2,:,:}$, $\mathcal{X}(:,3,2)_{1,:,:}$ and $\mathcal{X}(:,3,3)_{2,:,:}$, respectively. Since $nnz(\mathcal{X}(:,3,:)_{1,:,:})=1$ and $nnz(\mathcal{X}(:,3,:)_{2,:,:})=2$, $w_1(v_3^B)=1$ and $w_2(v_3^B)=2$. Since $v_3^B \in \mathcal{V}_1^B$ in Π^B, slice $\mathcal{X}(:,3,:)$ is reordered in chunk $\mathcal{X}_{:,1,:}$.

6.2.1 Correctness of CartHP

In this section, we show the correctness of the proposed CartHP model in minimizing the total communication volume of medium-grain CPD-ALS.

Suppose that we have a cartesian partition of $\mathcal{X}$ obtained by CartHP and consider a horizontal slice $\mathcal{X}(i,:,:)$. Note that $\mathcal{X}(i,:,:)$ is not divided into any subslices in $\phi1$. In $\phi2$, $\mathcal{X}(i,:,:)$ is divided into R subslices $\mathcal{X}(i,:,:)_{:,r,:}$ for $r = 1,\ldots,R$ along mode 2. Let $Z^B(i,:,:)$ denote the set of mode-2 indices of the nonzero subslices among these R subslices, i.e.,

$$Z^B(i,:,:) = \{r \mid \mathcal{X}(i,:,:)_{:,r,:} \text{ is a nonzero subslice}\}.$$

Note that $Z^B(i,:,:) \subseteq \{1,\ldots,R\}$. For example in Figure 6.3, $Z^B(1,:,:) = \{1,2\}$ and $Z^B(2,:,:) = \{1\}$. In $\phi3$, each subslice $\mathcal{X}(i,:,:)_{:,r,:}$ is divided into S subsubslices $\mathcal{X}(i,:,:)_{:,r,s}$ for $s = 1,\ldots,S$ along mode 3. Let $Z^C(i,:,:)_{:,r,:}$ denote the set of mode-3 indices of the nonzero subslices among these S subsubslices, that is,

$$Z^C(i,:,:)_{:,r,:} = \{s \mid \mathcal{X}(i,:,:)_{:,r,s} \text{ is a nonzero subslice}\}.$$

Note that $Z^C(i,:,:)_{:,r,:} \subseteq \{1,\ldots,S\}$. For example in Figure 6.3, $Z^C(1,:,:)_{:,1,:} = \{1\}$ and $Z^C(1,:,:)_{:,2,:} = \{2\}$.

$\mathcal{X}(i,:,:)$ is represented by a single net n_i^A in $\phi2$ and by at most R nets $n_{i(r)}^A$ in $\phi3$, but not represented by any nets in $\phi1$. Let $cs_i^A(\phi)$ denote the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in a phase ϕ. Since $\lambda(n_i^A)$ in $\phi2$ amounts to the number of $\mathcal{X}(i,:,:)$'s nonzero subslices along mode 2, which is $|Z^B(i,:,:)|$, the cutsize incurred by n_i^A in $\phi2$ is

$$cs_i^A(\phi2) = (\lambda(n_i^A) - 1)c(n_i^A) = (|Z^B(i,:,:)| - 1)F.$$

$\lambda(n_{i(r)}^A)$ in $\phi3$ amounts to the number of $\mathcal{X}(i,:,:)_{:,r,:}$'s nonzero unshared subslices, which is $|Z^C(i,:,:)_{:,r,:}|$. Then, the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in $\phi3$ is

$$\begin{aligned} cs_i^A(\phi^3) &= \textstyle\sum_{r\in Z^B(i,:,:)}(\lambda(n_{i(r)}^A) - 1)c(n_{i(r)}^A) \\ &= \textstyle\sum_{r\in Z^B(i,:,:)}(|Z^C(i,:,:)_{:,r,:}| - 1)F \\ &= \left(\textstyle\sum_{r\in Z^B(i,:,:)} |Z^C(i,:,:)_{:,r,:}| - |Z^B(i,:,:)|\right)F. \end{aligned}$$

Let cs_i^A denote the total cutsize incurred by the nets representing $\mathcal{X}(i,:,:)$ in all phases. Since $cs_i^A = cs_i^A(\phi2) + cs_i^A(\phi3)$ and the term $|Z^B(i,:,:)|F$ is cancelled out in this summation, we obtain

$$cs_i^A = \left(\sum_{r\in Z^B(i,:,:)} |Z^C(i,:,:)_{:,r,:}| - 1\right)F.$$

Note that the sum of the number of nonzero subsubslices in $Z^C(i,:,:)_{:,r,:}$ for all $r \in Z^B(i,:,:)$ gives the total number of unshared subslices in $\mathcal{X}(i,:,:)$. Then,

$$cs_i^A = (|Z_i^A| - 1)F. \tag{6.3}$$

By equations (6.1) and (6.3), we obtain

$$cs_i^A = vol_i^A.$$

These findings apply to mode-2 and mode-3 slices as follows: $cs_j^B = cs_j^B(\phi1) + cs_j^B(\phi3)$ and $cs_k^C = cs_k^C(\phi1) + cs_k^C(\phi2)$, where cs_j^B and cs_k^C denote total cutsizes incurred by the nets representing $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ in all phases, respectively. Then, $cs_j^B = (|Z_j^B| - 1)F = vol_j^B$ and $cs_k^C = (|Z_k^C| - 1)F = vol_k^C$. That is, the total cutsizes incurred by the nets representing $\mathcal{X}(i,:,:)$, $\mathcal{X}(:,j,:)$ and $\mathcal{X}(:,:,k)$ are equal to the communication volumes regarding factor-matrix rows $\hat{\mathbf{A}}(i,:)$, $\hat{\mathbf{B}}(j,:)$ and $\hat{\mathbf{C}}(k,:)$, respectively. Since the overall cutsize of CartHP is equal to the sum of the cutsizes of the nets representing individual slices in all phases, minimizing the overall cutsize corresponds to minimizing the total communication volume.

In Figure 6.3, consider slice $\mathcal{X}(:,:,2)$ and the nets that represent this slice. In $\phi1$, the cutsize incurred by n_2^C is $cs_2^C(\phi1) = (2-1)F = F$. In $\phi2$, the total cutsize incurred by nets $n_{2(1)}^C$ and $n_{2(2)}^C$ is $cs_2^C(\phi2) = (2-1)F + (2-1)F = 2F$. Then, the total cutsize of $cs_2^C = 3F$ incurred by the nets representing $\mathcal{X}(:,:,2)$ is equal to the communication volume regarding $\hat{\mathbf{C}}(2,:)$, which is given by $vol_2^C = (|Z_2^C|-1)F = 3F$. Similarly, $cs_1^C = vol_1^C = F$ and $cs_3^C = vol_3^C = F$. Then, the total cutsize incurred by the nets representing the frontal slices is $5F$, which is equal to the total communication volume in the third phase of medium-grain CPD-ALS , i.e., $vol^C = 5F$. With similar discussions for the first and second phases, the total cutsize of $12F$ in CartHP is equal to the total communication volume.

6.2.2 1D Factor Matrix Partitioning

Recall that the correctness of CartHP in encapsulating total communication volume depends on the consistency condition. In order to satisfy this condition, we assign each factor-matrix row to one of the processors that own a nonzero subslice in the corresponding slice.

The rows of a factor matrix are partitioned among processors, independently for each factor matrix. Note that the communications regarding each row chunk (e.g., $\mathbf{A}_q$) are confined to a distinct processor layer (e.g., $p_{q,:,:}$). Hence, the rows in a chunk are partitioned among the processors in the corresponding layer, independently for each chunk. For partitioning the rows in a chunk, we adopt the best-fit-decreasing heuristic used for solving the P-feasible bin-packing problem [112]. The rows are considered in decreasing order of the number of their nonzero unshared subslices. That is, $\mathbf{A}(i,:)$ is processed earlier than $\mathbf{A}(i',:)$ if $|Z_i^A| \geq |Z_{i'}^A|$. The best-fit criterion corresponds to assigning a row to a processor that currently has the minimum communication volume among the processors that own a nonzero subslice in the corresponding slice. After assigning a row to a processor, the volumes of the respective processors are increased accordingly.

6.2.3 Mode Processing Order

In our model, we determine the number of chunks along each mode, i.e., Q, R and S values, to be proportional to the tensor dimension in that mode, i.e., I, J and K values, as proposed in [48]. Recall that CartHP introduces the number of chunks along a mode as a multiplicative factor to the number of constraints in each further partitioning phase. For example, Q chunks obtained in $\phi 1$ lead to Q and $Q \times R$ constraints in $\phi 2$ and $\phi 3$, respectively. However, the performance of the multi-constraint partitioning tools is known to degrade with increasing number of constraints [113]. In order to have fewer constraints, the modes with fewer chunks should be processed earlier. For this purpose, CartHP processes the modes in increasing order of the number of chunks.

6.2.4 Extension to More Than Three Modes

For an M-mode tensor $\mathcal{X}$ and an $P_1 \times \cdots \times P_M$ virtual mesh of processors, CartHP consists of M partitioning phases. In phase ϕm, hypergraph $\mathcal{H}^m = (\mathcal{V}^m, \bigcup_{1 \leq k \leq M, k \neq m} \mathcal{N}^k)$ is constructed and partitioned into P_m parts. In $\mathcal{H}^m$, each mode-m slice is represented by a vertex with $\Pi_{i=1}^{m} P_{i-1}$ weights (with $P_0 = 1$) in $\mathcal{V}^m$, whereas each nonzero mode-k (sub)slice is represented by a net in $\mathcal{N}^k$ for $k = 1, \ldots, m-1, m+1, \ldots, M$. Net n connects vertex v if the intersection of the (sub)slices represented by v and n contains at least one nonzero. Here, the slices are $M-1$ dimensional, hence the intersection of two slices along different modes is $M-2$ dimensional.

A P_m-way partition of $\mathcal{H}^m$ induces P_m slice chunks along mode m. As a result, each slice along a mode different than mode m is divided into P_m subslices along mode m. In $\mathcal{H}^m$, each nonzero mode-k subslice is represented by a net in $\mathcal{N}^k$ in order to correctly encapsulate the communication volume. Here, these nonzero subslices are the smallest possible subslices divided by the chunks. Similarly, the number of nonzeros in each subslice of a mode-m slice constitutes a different weight to the vertex representing that slice for achieving computational load balance via multi-constraint partitioning.

6.2.5 Balancing Constraint of CartHP

In partitioning $\mathcal{H}^A$, $\mathcal{H}^B$ and $\mathcal{H}^C$, the maximum allowed imbalance ratios are set to ϵ_1, ϵ_2 and ϵ_3, respectively. Maintaining balance on the weights of the parts in Π^A corresponds to allowing a maximum of

$$\frac{nnz(\mathcal{X})(1+\epsilon_1)}{Q}$$

nonzeros in each horizontal chunk $\mathcal{X}_{q,:,:}$. Consider chunk $\mathcal{X}_{q,:,:}$, which is partititoned into R subchunks by Π^B along mode 2. Maintaining balance on the weights of the parts in Π^B corresponds to allowing a maximum of

$$\frac{nnz(\mathcal{X}_{q,:,:})(1+\epsilon_2)}{R} \leq \frac{nnz(\mathcal{X})(1+\epsilon_1)(1+\epsilon_2)}{Q \times R}$$

nonzeros in each subchunk $\mathcal{X}_{q,r,:}$. Consider subchunk $\mathcal{X}_{q,r,:}$, which is partititoned into S subsubchunks by Π^C along mode 3. Maintaining balance on the weights of the parts in Π^C corresponds to allowing a maximum of

$$\frac{nnz(\mathcal{X}_{q,r,:})(1+\epsilon_3)}{S} \leq \frac{nnz(\mathcal{X})(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)}{Q \times R \times S}$$

nonzeros in each subsubchunk $\mathcal{X}_{q,r,s}$. Hence, the number of nonzeros assigned to a processor is bounded above by

$$\frac{nnz(\mathcal{X})(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3)}{P}.$$

6.3 deCartHP: Direct Extension of CBHP

As CartHP, deCartHP contains M hypergraph partitioning phases for an M-mode tensor. In the mth phase, each mode-m slice is represented by a vertex, whereas each slice along the other modes is represented by a net. The main difference of deCartHP from CartHP is using a single net for each slice, disregarding the subslices in the ones that have been divided in the previous partitioning phases.

Algorithm 12 $\phi 2(\mathcal{X}, R, \epsilon_2)$ of deCartHP

1: $\mathcal{V}^B \leftarrow \{v_1^B, \ldots, v_J^B\}$
2: **for each** lateral subslice $\mathcal{X}(:, j, :)_{q,:,:}$ **do**
3:     $w_q(v_j^B) \leftarrow nnz(\mathcal{X}(:, j, :)_{q,:,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^C \leftarrow \emptyset$
5: **for each** horizontal slice $\mathcal{X}(i, :, :)$ **do**
6:     $\mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_i^A\}$ with $Pins(n_i^A) = \emptyset$
7:     **for each** nonzero fiber $\mathcal{X}(i, j, :)$ **do**
8:         $Pins(n_i^A) \leftarrow Pins(n_i^A) \cup \{v_j^B\}$
9: **for each** frontal slice $\mathcal{X}(:, :, k)$ **do**
10:     $\mathcal{N}^C \leftarrow \mathcal{N}^C \cup \{n_k^C\}$ with $Pins(n_k^C) = \emptyset$
11:     **for each** nonzero fiber $\mathcal{X}(i, :, k)$ **do**
12:         $Pins(n_k^C) \leftarrow Pins(n_k^C) \cup \{v_j^B\}$
13: $\mathcal{H}^B \leftarrow (\mathcal{V}^B, \mathcal{N}^A \cup \mathcal{N}^C)$
14: $\Pi^B = \{\mathcal{V}_1^B, \ldots, \mathcal{V}_R^B\} \leftarrow$ MC-HP$(\mathcal{H}^B, R, \epsilon_2)$
15: **for** $r \leftarrow 1$ **to** R **do**
16:     **for each** $v_j^B \in \mathcal{V}_r^B$ **do**
17:         Assign slice $\mathcal{X}(:, j, :)$ to chunk $\mathcal{X}_{:,r,:}$

The rest of the discussions is held for three-mode tensors for simplicity. For deCartHP and CartHP, the main layout (Algorithm 8) and phase $\phi 1$ (Algorithm 9) are exactly the same. However, phases $\phi 2$ and $\phi 3$ of deCartHP are different than those of CartHP in terms of the nets of $\mathcal{H}^B$ and $\mathcal{H}^C$. Algorithms 12 and 13 respectively display $\phi 2$ and $\phi 3$ of deCartHP.

In $\phi 2$, the difference between deCartHP and CartHP is the representation of the frontal slices by the nets in $\mathcal{N}^C$. In $\mathcal{H}^B$ of deCartHP, each frontal slice $\mathcal{X}(:, :, k)$ is represented by a single net $n_k^C \in \mathcal{N}^C$ instead of multiple nets. Then, $\mathcal{N}^C = \{n_1^C, \ldots, n_K^C\}$ (lines 9-10). Net n_k^C connects vertex v_j^B if $\mathcal{X}(:, j, k)$ is a nonzero fiber (lines 11-12).

In $\phi 3$, the difference between deCartHP and CartHP is the representation of the horizontal and lateral slices by the nets in $\mathcal{N}^A$ and $\mathcal{N}^B$, respectively. In $\mathcal{H}^C$ of deCartHP, each horizontal slice $\mathcal{X}(i, :, :)$ is represented by a single net $n_i^A \in \mathcal{N}^A$ instead of multiple nets. Similarly, each lateral slice $\mathcal{X}(:, j, :)$ is represented by a single net $n_j^B \in \mathcal{N}^B$ instead of multiple nets. Then, $\mathcal{N}^A = \{n_1^A, \ldots, n_I^A\}$ (lines

Algorithm 13 $\phi 3(\mathcal{X}, S, \epsilon_3)$ of deCartHP

1: $\mathcal{V}^C \leftarrow \{v_1^C, \ldots, v_K^C\}$
2: **for each** frontal subslice $\mathcal{X}(:,:,k)_{q,r,:}$ **do**
3: $\quad w_{q,r}(v_k^C) \leftarrow nnz(\mathcal{X}(:,:,k)_{q,r,:})$
4: $\mathcal{N}^A \leftarrow \mathcal{N}^B \leftarrow \emptyset$
5: **for each** horizontal slice $\mathcal{X}(i,:,:)$ **do**
6: $\quad \mathcal{N}^A \leftarrow \mathcal{N}^A \cup \{n_i^A\}$ with $Pins(n_i^A) = \emptyset$
7: $\quad$ **for each** nonzero fiber $\mathcal{X}(i,:,k)$ **do**
8: $\qquad Pins(n_i^A) \leftarrow Pins(n_i^A) \cup \{v_k^C\}$
9: **for each** lateral slice $\mathcal{X}(:,j,:)$ **do**
10: $\quad \mathcal{N}^B \leftarrow \mathcal{N}^B \cup \{n_j^C\}$ with $Pins(n_j^C) = \emptyset$
11: $\quad$ **for each** nonzero fiber $\mathcal{X}(:,j,k)$ **do**
12: $\qquad Pins(n_j^B) \leftarrow Pins(n_j^B) \cup \{v_k^C\}$
13: $\mathcal{H}^C \leftarrow (\mathcal{V}^C, \mathcal{N}^A \cup \mathcal{N}^B)$
14: $\Pi^C = \{\mathcal{V}_1^C, \ldots, \mathcal{V}_S^C\} \leftarrow$ MC-HP$(\mathcal{H}^C, S, \epsilon_3)$
15: **for** $s \leftarrow 1$ **to** S **do**
16: $\quad$ **for each** $v_k^C \in \mathcal{V}_s^C$ **do**
17: $\qquad$ Assign slice $\mathcal{X}(:,:,k)$ to chunk $\mathcal{X}_{:,:,s}$

5-6) and $\mathcal{N}^B = \{n_1^B, \ldots, n_J^B\}$ (lines 9-10). Net n_i^A connects vertex v_k^C if $\mathcal{X}(i,:,k)$ is a nonzero fiber (lines 7-8). Similarly, net n_j^B connects vertex v_k^C if $\mathcal{X}(:,j,k)$ is a nonzero fiber (lines 11-12).

The weights assigned to vertices in deCartHP are the same as those in CartHP, so is the discussion held for the maximum imbalance ratios.

Figure 6.4 illustrates $\phi 2$ and $\phi 3$ of deCartHP on the same tensor $\mathcal{X}$ given in Figure 6.3. We omit to show $\phi 1$ and the resulting cartesian partition as they are exactly the same as those in Figure 6.3. Recall that the total communication volume of that cartesian partition is $12F$. Also recall that the cutsize of Π^A is $4F$ in $\phi 1$ of Figure 6.3. As seen in Figure 6.4, the cutsizes of Π^B and Π^C in deCartHP are $3F$ and $7F$, respectively. Thus, the total cutsize in deCartHP, which is $14F$, does not correspond to the total communication volume of the resulting cartesian partition, which is $12F$. As exemplified by this figure, deCartHP is deficient since minimizing the total cutsize in deCartHP does not correspond to minimizing the total communication volume. The reason for this deficiency is explained in the

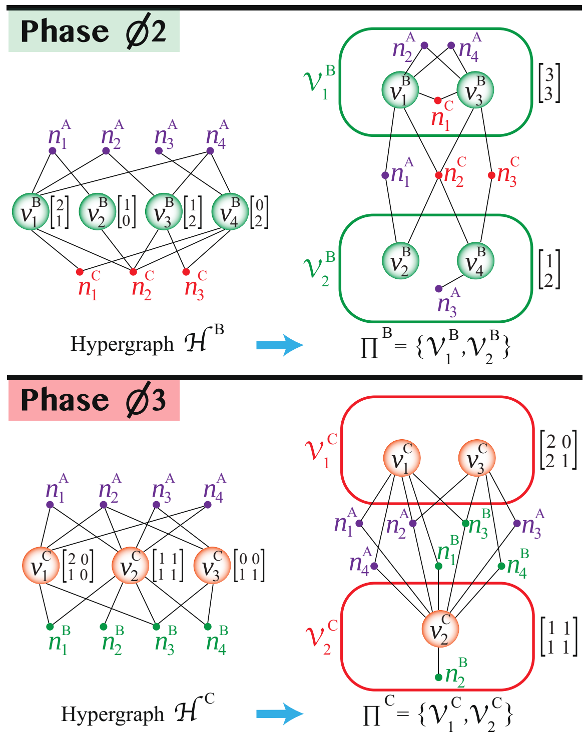


Figure 6.4: Phases $\phi2$ and $\phi3$ of deCartHP for the example given in Figure 6.3.
following section.

6.3.1 Deficiency of deCartHP

Suppose we have a 3D cartesian partition of $\mathcal{X}$ obtained by deCartHP and consider a horizontal slice $\mathcal{X}(i,:,:)$. In $\phi2$, R lateral chunks obtained by Π^B divide $\mathcal{X}(i,:,;)$ into R subslices along mode 2. $\lambda(n_i^A)$ in Π^B amounts to the number of nonzero subslices among these R subslices, which is denoted by $|Z^B(i,:,:)|$. Then, the cutsize incurred by net n_i^A in $\phi2$ is

$$cs_i^A(\phi2) = (\lambda(n_i^A) - 1)c(n_i^A) = (|Z^B(i,:,:)| - 1)F.$$

In $\phi3$, S frontal chunks obtained by Π^C divide $\mathcal{X}(i,:,;)$ into S subslices along mode 3. Let $Z^C(i,:,:)$ denote the set of mode 3 indices of the nonzero subslices among these S subslices, that is,

$$Z^C(i,:,:) = \{s \mid \mathcal{X}(i,:,:) \text{ is a nonzero subslice}\}.$$

$\lambda(n_i^A)$ in Π^C amounts to the number of nonzero subslices among these S subslices, which is denoted by $|Z^C(i,:,:)|$. Then, the cutsize incurred by net n_i^A in $\phi 3$ is

$$cs_i^A(\phi 3) = (\lambda(n_i^A) - 1)c(n_i^A) = (|Z^C(i,:,:)| - 1)F.$$

Therefore, the total cutsize cs_i^A incurred by the nets representing $\mathcal{X}(i,:,:)$ is

$$(|Z^B(i,:,:)| - 1)F + (|Z^C(i,:,:)| - 1)F. \tag{6.4}$$

However as explained in Section 6.1, the volume of communication regarding factor-matrix row $\hat{\mathbf{A}}(i,:)$ is $(|Z_i^A| - 1)F$, which is not necessarily equal to the expression given in (6.4). Thus, minimizing the total cutsize in deCartHP does not necessarily correspond to minimizing the total communication volume.

In Figure 6.4, consider slice $\mathcal{X}(1,:,:)$ and the nets that represent this slice. The total cutsize incurred by nets representing $\mathcal{X}(1,:,:)$ is $(|Z^B(1,:,:)|-1)F+(|Z^C(1,:,:)|-1)F=F+F=2F$. However, the communication volume regarding factor-matrix row $\hat{\mathbf{A}}(1,:)$ is $(|Z_1^A|-1)F=F$. Therefore, the total cutsize incurred by the nets representing $\mathcal{X}(1,:,:)$ does not correspond to the total volume regarding $\hat{\mathbf{A}}(1,:)$.

6.4 Experiments

We evaluate the performance of the proposed CartHP method against the baseline multi-dimensional cartesian partitioning method [48]. For obtaining balance on the number of tensor nonzeros, this method randomly permutes the slices at each mode before obtaining respective slice chunks. We refer to this baseline method as CartR, with “R” standing for “random”. The performance comparison is conducted in terms of partition statistics and parallel CPD-ALS runtimes for 12 tensors on $64, 128, 256, 512$ and 1024 processors. Finally, we discuss the amortization of the partitioning overhead introduced by CartHP in terms of CPD-ALS solutions.

6.4.1 Setting

For partitioning hypergraphs in CartHP (line 14 in Algorithms 9, 10 and 11), we use PaToH [49] (version 3.2) in speed mode with maximum allowable imbalance ratio set to 0.04, i.e., $\epsilon_m = 0.04$. In PaToH, we set the refinement algorithm to FM with tight balance. Since PaToH contains randomized algorithms, we ran CartHP five times for each instance and report the geometric average of the results.

For conducting the parallel CPD-ALS experiments, we implemented the medium-grain CPD-ALS algorithm in C using MPI for interprocess communication. The source code is compiled with Cray C compiler (version 2.5.9) using the optimization level three. For the fold and expand operations on factor-matrix rows, personalized all-to-all collective operations are used. For storing the subtensors in processors, an extension of the compressed row storage (CRS) scheme for tensors [114] is utilized. MTTKRP operation is performed in a fiber-centric manner to reduce the FLOP counts, as described in [114]. For the rest of the computations, efficient CBLAS routines provided by Intel MKL library (version 2017) are used whenever needed. Our parallel implementation is orthogonal to the data partitioning method, hence it can take any medium-grain partition as input. For a fair comparison, we use the same parallel implementation for evaluating the partitions obtained by CartR. In our experiments, we set the number of components in CPD-ALS to 16, i.e., $F = 16$. For each instance, the runtime of one CPD-ALS iteration is reported by taking the average of the total runtime of 1000 iterations.

We conducted our parallel experiments on a Cray XC40 cluster, namely Hazel Hen, based in the High Performance Computing Center Stuttgart (HLRS). A node of this cluster 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 CRAY Aries, which is a high speed network with Dragonfly topology. The peak performance is up to 7.42 Petaflops (quadrillion floating point operations per second).

Table 6.1: Properties of the test tensors.

name	I	J	K	L	nnz
Facebook	42.4K	40.0K	1.5K	–	738.1K
NELL-b	2.4M	428	344.6K	–	3.0M
Brightkite	51.4K	942	773.0K	–	2.7M
Finefoods	67.1K	11.8K	82.3K	–	5.6M
Gowalla	107.1K	597	1.3M	–	6.3M
MovieAmazon	87.9K	4.4K	226.5K	–	15.0M
NELL-c	5.1M	435	716.3K	–	96.7M
Netflix	17.8K	480.2K	2.2K	–	100.5M
Yelp	686.6K	85.5K	773.3K	–	185.6M
MovieLens	7.8K	19.5K	38.6K	3.4K	465.6K
Flickr	319.7K	28.2M	1.6M	730	112.9M
Delicious	532.9K	17.3M	2.5M	1.4K	140.1M

6.4.2 Dataset

In our experiments, we use 12 sparse tensors whose properties are given in Table 6.1. All of these tensors are obtained from the datasets arising in real-world applications. First nine of them have three modes, whereas the remaining three have four modes. Columns 2–5 and 6 respectively display the dimensions and the number of nonzeros in the respective tensor.

`Facebook` consists of the wall-posting information in the form of owner-poster-date triplets from the Facebook New Orleans networks[115]. `NELL-b` and `NELL-c` consist of the beliefs in the form of entity-relation-entity triplets discovered by the Never Ending Language Learning (NELL) project [116]. `NELL-b` contains the relations that NELL believes to be true, whereas `NELL-c` contains only the candidate beliefs. `Brightkite` and `Gowalla` consist of checkin information in the form of user-date-location triplets obtained from location-based social networks [117]. `Finefoods` and `MovieAmazon` consist of user-product-word triplets obtained from food and movie reviews in Amazon, respectively [118]. `Netflix` consists of user-item-time triplets obtained from the ratings in Netflix Prize competition [119]. Similar to `Finefoods`, `Yelp` consists of user-business-word triplets obtained from business reviews in Yelp academic dataset[2]. `MovieLens` consists

[2]https://www.yelp.com/dataset_challenge/dataset

Table 6.2: Average results obtained by CartHP normalized with respect to those obtained by CartR.

number of procs	imb	number of messages		comm volume		parallel runtime	
		max	avg	max	avg	comm	total
64	1.01	0.97	0.93	0.61	0.42	0.50	0.82
128	1.01	0.97	0.93	0.60	0.45	0.56	0.78
256	1.05	0.97	0.91	0.60	0.49	0.59	0.74
512	1.05	0.98	0.90	0.53	0.51	0.61	0.72
1024	1.05	0.97	0.85	0.53	0.53	0.61	0.72
average	**1.03**	**0.97**	**0.90**	**0.57**	**0.48**	**0.57**	**0.76**

of user-movie-tag-time quadruplets obtained from free-text taggings in MovieLens 20M dataset [120]. `Flickr` and `Delicious` consist of user-resource-tag-time quadruplets which were first crawled by Görlitz et al. [121] respectively from flickr.com and delicious.com.

6.4.3 Parallel CPD-ALS Results

Table 6.2 presents the average results obtained by CartHP normalized with respect to those obtained by CartR. Each row displays the geometric average of the results on 12 tensors for the respective number of processors. Column “imb” denotes load imbalance, which we compute as the ratio of the maximum to the average number of nonzeros assigned to a processor. Columns under ”number of messages” and ”comm volume” denote the number of messages sent and received by a processor regarding the expand and fold steps through all phases and the volume of data communicated along these messages, respectively. Under both, ”max” and ”avg” denote the maximum and average amount of the corresponding metric over all processors, respectively. Under ”parallel runtime”, columns ”comm” and ”total” respectively denote the communication time and total runtime of a single iteration in medium-grain-parallel CPD-ALS.

As seen in Table 6.2, CartHP drastically reduces average communication volume compared to CartR. Note that the reduction in average communication volume also refers to the reduction in total communication volume. CartHP reduces average (total) volume by 58%, 55%, 51%, 49% and 47% for 64, 128, 256, 512 and 1024 processors, respectively. These improvements are expected since CartHP minimizes this metric while CartR only provides a loose upper bound on it. The reduction in average volume leads to a similar reduction in maximum volume, by 39%, 40%, 40%, 47% and 47% for 64, 128, 256, 512 and 1024 processors, respectively. The reduction in average volume also leads to a significant reduction in average (total) number of messages. CartHP reduces average number of messages by 7%, 7%, 9%, 10% and 15% for 64, 128, 256, 512 and 1024 processors, respectively. The reduction in average number of messages leads to a slight reduction of 2-3% in maximum number of messages.

The drastic reductions in communication cost metrics lead to a drastic reduction in the communication time of CPD-ALS by 50%, 44%, 41%, 39% and 39% for 64, 128, 256, 512 and 1024 processors, respectively. Although CartHP causes an increase in load imbalance by at most 5% on the average, the reduction obtained in communication time conceals this increase and leads to a significant reduction in total CPD-ALS runtime. CartHP reduces total runtime by 28%, 32%, 36%, 38% and 38% for 64, 128, 256, 512 and 1024 processors, respectively.

Table 6.3 presents the detailed results obtained by CartR and CartHP on 512 processors for each tensor. The values given for maximum and average communication volumes are in terms of words. For each tensor, the best result attained for each metric is given in boldface.

Table 6.3: Partition statistics and parallel runtime results obtained by CartR and CartHP for one CPD-ALS iteration on 512 processors.

	CartR							CartHP						
		number of messages		comm volume		parallel runtime (ms)			number of messages		comm volume		parallel runtime (ms)	
tensor	imb	max	avg	max	avg	comm	total	imb	max	avg	max	avg	comm	total
`Facebook`	1.32	2,162	1,956	114K	83K	2.7	3.4	**1.01**	**2,043**	**1,901**	**67K**	**58K**	**1.9**	**2.8**
`NELL-b`	1.06	1,400	534	158K	75K	4.4	7.5	**1.01**	**1,262**	**224**	**38K**	**11K**	**2.1**	**4.5**
`Brightkite`	**1.73**	2,323	2,306	231K	142K	5.1	8.8	4.25	**2,300**	**2,155**	**85K**	**64K**	**3.3**	**6.0**
`Finefoods`	1.08	**1,259**	1,225	356K	257K	7.4	11.1	**1.05**	1,263	**1,191**	**308K**	**203K**	**5.1**	**9.4**
`Gowalla`	1.08	**2,136**	1,866	687K	443K	7.6	13.2	**1.01**	2,182	**1,757**	**186K**	**133K**	**4.0**	**7.0**
`MovieAmazon`	**1.09**	**2,209**	**2,154**	**607K**	474K	**8.3**	**13.9**	1.10	2,228	2,209	1.1M	**423K**	8.5	16.3
`NELL-c`	**1.01**	1,941	1,504	2.5M	1.4M	34.5	72.6	1.07	**1,845**	**1,254**	**943K**	**491K**	**15.4**	**44.5**
`Netflix`	**1.01**	**2,564**	**2,562**	**594K**	551K	9.9	**35.5**	1.14	**2,564**	2,564	729K	**471K**	**9.3**	35.7
`Yelp`	**1.06**	**1,267**	**1,267**	**4.1M**	3.3M	62.5	126.7	1.07	1,268	1,268	5.7M	**2.3M**	**47.9**	**113.4**
`MovieLens`	1.30	2,464	2,043	198K	85K	2.9	4.3	**1.08**	**2,219**	**1,969**	**77K**	**65K**	**2.4**	**3.9**
`Flickr`	**1.01**	**4,603**	**4,595**	17.7M	10.6M	327.0	505.2	1.14	4,608	4,597	**4.0M**	**3.4M**	**108.0**	**216.3**
`Delicious`	1.06	**4,367**	**4,367**	24.0M	11.3M	398.2	649.7	**1.05**	4,368	4,368	**8.8M**	**6.1M**	**171.6**	**355.9**

As seen in Table 6.3, CartHP attains a better result in average communication volume for all tensors and in maximum communication volume for 9 out of 12 tensors. In communication time and total CPD-ALS runtime, it achieves a better result for 11 and 10 tensors, respectively. For the rest of the metrics, CartHP and CartR have comparable performances since each achieves a better result for half of the tensors. The highest reduction rates in total runtime are observed for `Gowalla`, `Flickr` and `Delicious`. This can be explained by the drastic amounts of decrease achieved in both maximum volume and total volume for these tensors. CartHP performs comparable to CartR for `Netflix` since the reduction in the communication time and the increase in the imbalance compensate each other. For `MovieAmazon`, CartHP performs worse than CartR due to the increase in the communication time stemming from the increase in maximum volume despite the decrease in total volume. Note that a similar increase is also observed for `Netflix`, but it does not degrade the communication time much due to a higher decrease in total volume.

Figure 6.5 displays the strong scaling curves for all tensors in terms of total CPD-ALS runtime. For 9 out of 12 tensors, CartHP achieves better CPD-ALS scalability compared to CartR. This is because CartHP obtains drastic reductions in both maximum and average communication volume metrics for these tensors. CartHP performs comparable to CartR for `Netflix` and `Yelp` and slightly worse than CartR for `MovieAmazon` since CartHP increases maximum volume while decreasing average volume for these tensors on all processor counts. For `Facebook` and `MovieLens`, although CartHP performs better than CartR, both methods display poor scalability for these tensor since they are small.

Table 6.4 presents the results obtained by CartHP normalized with respect to those obtained by CartR for each tensor and number of processors. Column "imb" denotes load imbalance, which we compute as the ratio of the maximum to the average number of nonzeros assigned to a processor. Columns under "number of messages" and "comm volume" denote the number of messages sent and received by a processor regarding the expand and fold steps through all phases and the volume of data communicated along these messages, respectively. Under both, "max" and "avg" denote the maximum and average amount of the

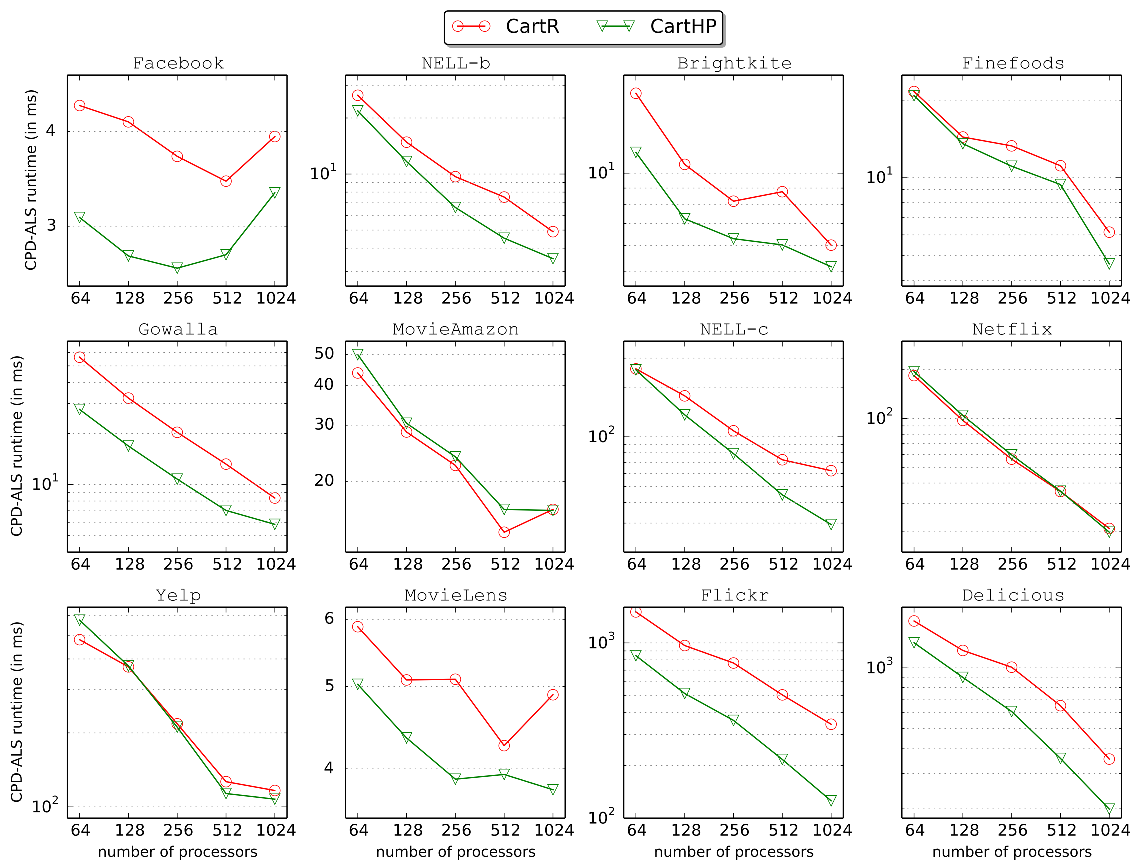


Figure 6.5: Strong scaling curves for medium-grain-parallel CPD-ALS obtained by CartR and CartHP.

Table 6.4: Detailed results obtained by CartHP normalized with respect to those obtained by CartR.

number of procs	tensor	imb	number of messages		comm volume		parallel runtime		tensor	imb	number of messages		comm volume		parallel runtime	
			max	avg	max	avg	comm	total			max	avg	max	avg	comm	total
64	Facebook	0.92	1.00	1.00	0.56	0.54	0.53	0.71	NELL-c	1.05	1.00	1.00	0.43	0.31	0.34	0.99
128		0.92	1.00	1.00	0.57	0.60	0.60	0.67		1.05	1.00	1.00	0.37	0.33	0.31	0.77
256		0.85	1.00	1.00	0.60	0.65	0.70	0.71		1.05	1.00	0.97	0.38	0.34	0.36	0.73
512		0.77	0.94	0.97	0.59	0.69	0.69	0.80		1.06	0.95	0.83	0.37	0.36	0.45	0.61
1024		0.73	0.83	0.86	0.57	0.72	0.77	0.84		1.03	0.96	0.79	0.30	0.37	0.32	0.47
64	NELL-b	0.99	0.72	0.44	0.25	0.05	0.15	0.82	Netflix	1.12	0.99	0.99	1.30	0.89	0.89	1.06
128		0.99	0.70	0.40	0.27	0.08	0.25	0.79		1.12	1.00	1.00	1.24	0.85	1.05	1.07
256		0.98	0.71	0.31	0.28	0.14	0.37	0.68		1.11	1.00	1.00	1.27	0.87	0.96	1.06
512		0.95	0.90	0.42	0.24	0.15	0.47	0.60		1.14	1.00	1.00	1.23	0.85	0.94	1.00
1024		0.93	0.96	0.37	0.35	0.20	0.70	0.72		1.11	1.00	0.97	1.24	0.83	0.87	0.95
64	Brightkite	0.83	0.99	0.99	0.24	0.26	0.28	0.66	Yelp	1.01	1.00	1.00	1.89	0.80	0.89	1.20
128		1.05	1.00	1.00	0.32	0.34	0.56	0.68		0.97	1.00	1.00	1.93	0.81	0.86	1.01
256		1.95	1.00	0.99	0.32	0.38	0.55	0.77		1.01	1.00	1.00	1.89	0.76	0.83	0.97
512		2.46	0.99	0.93	0.37	0.45	0.64	0.69		1.01	1.00	1.00	1.40	0.71	0.77	0.89
1024		2.58	0.92	0.72	0.42	0.49	0.65	0.86		1.02	1.00	1.00	1.33	0.73	0.76	0.92
64	Finefood	1.00	1.00	1.00	1.21	0.83	0.89	0.96	MovieLens	0.94	1.00	1.00	0.74	0.76	0.77	0.85
128		0.98	1.00	1.00	0.97	0.79	0.80	0.94		0.91	1.00	1.00	0.68	0.76	0.76	0.85
256		0.95	1.00	1.00	0.88	0.81	0.81	0.83		0.91	1.00	1.01	0.65	0.77	0.72	0.76
512		0.97	1.00	0.97	0.87	0.79	0.70	0.85		0.83	0.90	0.96	0.39	0.77	0.80	0.93
1024		0.95	1.00	0.92	0.76	0.78	0.79	0.75		0.94	0.94	0.97	0.51	0.77	0.55	0.77
64	Gowalla	1.01	0.99	0.99	0.23	0.18	0.21	0.49	Flickr	1.13	1.00	1.00	0.33	0.36	0.49	0.56
128		0.98	1.00	1.00	0.27	0.23	0.33	0.52		1.12	1.00	1.00	0.25	0.32	0.46	0.53
256		0.96	1.00	1.00	0.24	0.25	0.44	0.53		1.15	1.00	1.00	0.30	0.34	0.38	0.47
512		0.93	1.02	0.94	0.27	0.30	0.52	0.53		1.13	1.00	1.00	0.22	0.32	0.33	0.43
1024		0.91	1.01	0.93	0.29	0.34	0.66	0.70		1.06	0.99	0.92	0.20	0.32	0.31	0.37
64	MovieAmazon	1.00	1.00	1.00	1.91	0.92	1.01	1.14	Delicious	1.09	1.00	1.00	0.65	0.56	0.58	0.78
128		1.00	1.00	1.00	1.98	0.90	0.99	1.07		1.06	1.00	1.00	0.53	0.55	0.50	0.74
256		1.03	1.00	1.00	1.99	0.91	0.95	1.06		1.00	1.00	1.00	0.50	0.56	0.48	0.60
512		1.01	1.01	1.03	1.79	0.89	1.03	1.18		0.99	1.00	1.00	0.37	0.54	0.43	0.55
1024		1.08	1.04	1.04	1.54	0.90	0.78	0.99		0.99	1.00	0.99	0.38	0.58	0.47	0.56

corresponding metric over all processors, respectively. Under ”parallel runtime”, columns ”comm” and ”total” respectively denote the communication time and total runtime of a single iteration in medium-grain-parallel CPD-ALS.

As seen in the table, CartHP significantly reduces average (total) communication volume compared to CartR in all partitioning instances. This is expected since CartHP directly minimizes the total communication volume while CartR only provides an upper bound on it. The reduction in average communication volume results in a reduction in communication time of parallel CPD-ALS algorithm in all partitioning instances except three (`MovieAmazon` on 64 and 512 processors `Netflix` on 128 processors). CartHP outperforms CartR for all considered processor counts in terms of parallel CPD-ALS runtime for all tensors

Table 6.5: Comparison of partitioning overhead of CartHP against factorization in terms of sequential runtime.

tensor	CartHP time (s)	CartHP/factorization $F = 16$	CartHP/factorization $F = 64$
`Facebook`	5.8	4.68	0.82
`NELL-b`	9.8	0.53	0.10
`Brightkite`	9.2	1.73	0.18
`Finefoods`	22.3	2.32	0.32
`Gowalla`	31.5	3.93	0.47
`MovieAmazon`	35.0	1.17	0.12
`NELL-c`	62.3	0.50	0.08
`Netflix`	36.2	0.39	0.08
`Yelp`	380.6	6.28	1.10
`MovieLens`	4.6	9.22	1.38
`Flickr`	569.6	7.97	1.69
`Delicious`	1693.0	23.10	5.06
average	-	**2.60**	**0.41**

except for `MovieAmazon`, `Netflix` and `Yelp`. For these three tensors, observe that CartHP starts to perform better compared to CartR with increasing number of processors. For example on `Yelp`, the paralell runtime obtained by CartHP is 20% higher than that obtained by CartR on 64 processors, whereas it is 8% lower on 1024 processors.

6.4.4 Partitioning Overhead and Amortization

Table 6.5 reports the partitioning time of CartHP in seconds as well as the ratio of this partitioning time to the factorization time for each tensor. Here, each factorization involves a number of CPD-ALS iterations required to converge with tolerance 10^{-5} (as computed in [114]), where the number of iterations typically increases with increasing F. Both partitioning and factorization are performed in a sequential setting. As seen in the table, for `Netflix`, partitioning takes 0.39 and 0.08 factorizations for $F = 16$ and $F = 64$, respectively. On average, it takes 2.60 and 0.41 factorizations for $F = 16$ and $F = 64$, respectively.

Table 6.6 displays the average number of CPD solutions that amortize the

Table 6.6: Average number of CPD solutions that amortize the sequential partitioning time of CartHP.

$P=64$	$P=128$	$P=256$	$P=512$	$P=1024$	**avg**
3.39	3.91	4.92	8.02	14.18	**5.94**

sequential partitioning time of CartHP for each processor count, i.e., P value. Here, each CPD solution refers to running the parallel CPD-ALS algorithm for computing a factorization for ten different F values [122] starting from three different sets of initial factor matrices [67]. For each F value and initial factor matrix set, a factorization is assumed to require 25 iterations, so, each CPD solution is assumed to involve $10 \times 3 \times 25 = 750$ iterations. As seen in the table, on the average, the partitioning time of CartHP amortizes in only 3.39, 3.91, 4.92, 8.02, and 14.18 CPD solutions for 64, 128, 256, 512, and 1024 processors, respectively, where the overall average is computed as 5.94 CPD solutions.

6.5 Summary

We investigated the utilization of the sparsity pattern of a given tensor for minimizing the total communication volume in medium-grain CPD-ALS algorithm which adopts multi-dimensional cartesian tensor partitioning. We proposed a novel hypergraph-partitioning model that correctly encapsulates the total communication volume of medium-grain-parallel CPD-ALS. We demonstrated the effectiveness of the proposed model by conducting experiments on 12 tensors for up to 1024 processors. Our model drastically reduces the communication volume and the communication time of medium-grain-parallel CPD-ALS, hence the total parallel runtime.

Chapter 7

Conclusion

In this thesis, we presented novel hypergraph partitioning models and reordering methods for improving the performance of distributed-memory parallel sparse matrix and tensor computations. We proposed distributed-memory parallel Gauss-Seidel (dmpGS) and incomplete ILU (dmpILU) algorithms by implementing a parallel sparse triangular solver (stSPIKE) based on the SPIKE algorithm. By this way, the triangular systems in both Gauss-Seidel and ILU are parallelized in exchange for solving a much smaller reduced triangular system at each iteration. For reducing the size of these reduced systems in both dmpGS and dmpILU, we proposed novel hypergraph partitioning models that introduces new net anchoring and splitting schemes using the recursive bipartitioning scheme. For reducing the nonzero counts of the reduced systems, we proposed successful row reordering methods within the row blocks obtained by the hypergraph partitioning models. Furthermore, we proposed a novel hypergraph partitioning model for minimizing the communication volume and hence improve the parallel scalability of the medium-grain sparse tensor decomposition. We provided the results of extensive experiments validating the effectiveness of the proposed partitioning and reordering models against the state-of-the-art algorithms. We conclude that the hypergraph partitioning and reordering models can be used as powerful tools for improving the speedup and scalability of parallel sparse matrix and tensor computations.

As a future work, we will consider the parallel solution of the reduced systems in dmpGS and dmpILU to further alleviate the sequential bottleneck. We will also consider an in-block row reordering which takes the nonzeros of the diagonal blocks into account for further reducing the nonzero count in the reduced system. We are aiming to develop an ILU-based preconditioner which allows fill-in. We also plan to test the performance of the proposed preconditoner on iterative Krylov subspace methods. Finally, the future work will include extending the dmpGS and dmpILU algorithms for multiple right-hand-sides as it is also very common in modern applications. Using multiple right-hand-side vectors is expected to further enhance the performance of these algorithms since it enables using higher level BLAS subroutines compared to the single right-hand-side case. Moreover, the parallel solution time per right-hand-side vector will decrease since the parallel factorization is done only once.

Bibliography

[1] T. A. Davis, *Direct methods for sparse linear systems*, pp. 83–95. SIAM, 2006.

[2] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3, pp. 606–616. JHU press, 2013.

[3] M. W. Berry, "Large-scale sparse singular value computations," *The International Journal of Supercomputing Applications*, vol. 6, no. 1, pp. 13–49, 1992.

[4] B. Bylina and J. Bylina, "Analysis and comparison of reordering for two factorization methods (LU and WZ) for sparse matrices," in *International Conference on Computational Science*, pp. 983–992, Springer, 2008.

[5] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM Journal on Matrix Analysis and Applications*, vol. 23, no. 1, pp. 15–41, 2001.

[6] X. S. Li and J. W. Demmel, "SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 2, pp. 110–140, 2003.

[7] W. Liu, A. Li, J. Hogg, I. S. Duff, and B. Vinter, "A synchronization-free algorithm for parallel sparse triangular solves," in *European Conference on Parallel Processing*, pp. 617–630, Springer, 2016.

[8] Y. Saad, *Iterative methods for sparse linear systems.* SIAM, 2003.

[9] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the solution of linear systems: building blocks for iterative methods.* SIAM, 1994.

[10] Y. Saad, "ILUT: A dual threshold incomplete LU factorization," *Numerical linear algebra with applications*, vol. 1, no. 4, pp. 387–402, 1994.

[11] M. Benzi and M. Tuma, "A sparse approximate inverse preconditioner for nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, vol. 19, no. 3, pp. 968–994, 1998.

[12] Y. Notay, "An aggregation-based algebraic multigrid method," *Electronic transactions on numerical analysis*, vol. 37, no. 6, pp. 123–146, 2010.

[13] J. W. Ruge and K. Stüben, "Algebraic multigrid," in *Multigrid methods*, pp. 73–130, SIAM, 1987.

[14] M. Benzi, "Preconditioning techniques for large linear systems: a survey," *Journal of Computational Physics*, vol. 182, no. 2, pp. 418–477, 2002.

[15] M. Bernaschi, P. D'Ambra, and D. Pasquini, "AMG based on compatible weighted matching for GPUs," *Parallel Computing*, vol. 92, 2020.

[16] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics.* Numerical Mathematics and Scientific Computation, 2014.

[17] W. Pazner, "Efficient low-order refined preconditioners for high-order matrix-free continuous and discontinuous Galerkin methods," *SIAM Journal on Scientific Computing*, vol. 42, no. 5, pp. A3055–A3083, 2020.

[18] K. Watanabe, H. Igarashi, and T. Honma, "Comparison of geometric and algebraic multigrid methods in edge-based finite-element analysis," *IEEE transactions on magnetics*, vol. 41, no. 5, pp. 1672–1675, 2005.

[19] V. E. Henson and U. M. Yang, "BoomerAMG: A parallel algebraic multigrid solver and preconditioner," *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.

[20] R. Webster, "An algebraic multigrid solver for Navier-Stokes problems," *International Journal for Numerical Methods in Fluids*, vol. 18, no. 8, pp. 761–780, 1994.

[21] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, "Parallel multigrid smoothing: polynomial versus Gauss–Seidel," *Journal of Computational Physics*, vol. 188, no. 2, pp. 593–610, 2003.

[22] J. Boyle, M. Mihajlovic, and J. Scott, "HSL MI20: an efficient AMG preconditioner," tech. rep., Citeseer, 2007.

[23] P. Van, M. Brezina, J. Mandel, *et al.*, "Convergence of algebraic multigrid based on smoothed aggregation," *Numerische Mathematik*, vol. 88, no. 3, pp. 559–579, 2001.

[24] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala, "ML 5.0 smoothed aggregation user's guide," Tech. Rep. SAND2006-2649, Sandia National Laboratories, 2006.

[25] A. H. Sameh and R. P. Brent, "Solving triangular systems on a parallel computer," *SIAM Journal on Numerical Analysis*, vol. 14, no. 6, pp. 1101–1113, 1977.

[26] S.-C. Chen, D. J. Kuck, and A. H. Sameh, "Practical parallel band triangular system solvers," *ACM Transactions on Mathematical Software (TOMS)*, vol. 4, no. 3, pp. 270–277, 1978.

[27] J. J. Dongarra and A. H. Sameh, "On some parallel banded system solvers," *Parallel Computing*, vol. 1, no. 3, pp. 223–235, 1984.

[28] E. Polizzi and A. Sameh, "SPIKE: A parallel environment for solving banded linear systems," *Computers & Fluids*, vol. 36, no. 1, pp. 113–120, 2007. Challenges and Advances in Flow Simulation and Modeling.

[29] E. Polizzi and A. H. Sameh, "A parallel hybrid banded system solver: the SPIKE algorithm," *Parallel Computing*, vol. 32, no. 2, pp. 177–194, 2006. Parallel Matrix Algorithms and Applications (PMAA'04).

[30] B. S. Spring, E. Polizzi, and A. H. Sameh, "A feature-complete SPIKE dense banded solver," *ACM Transactions on Mathematical Software (TOMS)*, vol. 46, no. 4, pp. 1–35, 2020.

[31] M. Manguoglu, A. H. Sameh, and O. Schenk, "PSPIKE: A parallel hybrid sparse linear system solver," in *European Conference on Parallel Processing*, pp. 797–808, Springer, 2009.

[32] O. Schenk, M. Manguoglu, A. Sameh, M. Christen, and M. Sathe, "Parallel scalable PDE-constrained optimization: antenna identification in hyperthermia cancer treatment planning," *Computer Science-Research and Development*, vol. 23, no. 3-4, pp. 177–183, 2009.

[33] E. S. Bolukbasi and M. Manguoglu, "A multithreaded recursive and nonrecursive parallel sparse direct solver," in *Advances in Computational Fluid-Structure Interaction and Flow Simulation*, pp. 283–292, Springer, 2016.

[34] M. Manguoglu, "A domain-decomposing parallel sparse linear system solver," *Journal of Computational and Applied Mathematics*, vol. 236, no. 3, pp. 319–325, 2011.

[35] M. Manguoglu, "Parallel solution of sparse linear systems," in *High-Performance Scientific Computing*, pp. 171–184, Springer, 2012.

[36] I. E. Venetis, A. Kouris, A. Sobczyk, E. Gallopoulos, and A. H. Sameh, "A direct tridiagonal solver based on Givens rotations for GPU architectures," *Parallel Computing*, vol. 49, pp. 101–116, 2015.

[37] İ. Çuğu and M. Manguoğlu, "A parallel multithreaded sparse triangular linear system solver," *Computers & Mathematics with Applications*, vol. 80, no. 2, pp. 371–385, 2020.

[38] U. M. Yang, "Parallel algebraic multigrid methods-high performance preconditioners," in *Numerical solution of partial differential equations on parallel computers*, pp. 209–236, Springer, 2006.

[39] J. I. Aliaga, M. Barreda, G. Flegar, M. Bollhöfer, and E. S. Quintana-Ortí, "Communication in task-parallel ilu-preconditioned cg solvers using mpi+ ompss," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 21, p. e4280, 2017.

[40] M. Benzi, W. Joubert, and G. Mateescu, "Numerical experiments with parallel orderings for ilu preconditioners," *Electronic Transactions on Numerical Analysis*, vol. 8, pp. 88–114, 1999.

[41] L. C. Dutto, "The effect of ordering on preconditioned gmres algorithm, for solving the compressible navier-stokes equations," *International Journal for Numerical Methods in Engineering*, vol. 36, no. 3, pp. 457–497, 1993.

[42] J. A. Meijerink and H. A. Van Der Vorst, "An iterative solution method for linear systems of which the coefficient matrix is a symmetric m-matrix," *Mathematics of computation*, vol. 31, no. 137, pp. 148–162, 1977.

[43] Y. Chen, X. Tian, H. Liu, Z. Chen, B. Yang, W. Liao, P. Zhang, R. He, and M. Yang, "Parallel ilu preconditioners in gpu computation," *Soft Computing*, vol. 22, no. 24, pp. 8187–8205, 2018.

[44] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[45] R. A. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an" explanatory" multi-modal factor analysis," *UCLA Working Papers in Phonetics*, pp. 1–84, 1970.

[46] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[47] N. K. M. Faber, R. Bro, and P. K. Hopke, “Recent developments in candecomp/parafac algorithms: a critical review,” *Chemometrics and Intelligent Laboratory Systems*, vol. 65, no. 1, pp. 119 – 137, 2003.

[48] S. Smith and G. Karypis, “A medium-grained algorithm for distributed sparse tensor factorization,” *30th IEEE International Parallel & Distributed Processing Symposium*, 2016.

[49] U. V. Çatalyürek and C. Aykanat, “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 10, pp. 673–693, Jul 1999.

[50] 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.

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

[52] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to parallel computing: design and analysis of algorithms*, vol. 400. Benjamin/Cummings Redwood City, 1994.

[53] B. Hendrickson, R. Leland, and S. Plimpton, “An efficient parallel algorithm for matrix-vector multiplication,” *International Journal of High Speed Computing*, vol. 07, no. 01, pp. 73–88, 1995.

[54] U. V. Catalyurek and C. Aykanat, “A hypergraph-partitioning approach for coarse-grain decomposition,” in *Supercomputing, ACM/IEEE 2001 Conference*, pp. 42–42, Nov 2001.

[55] U. V. Catalyurek and C. Aykanat, “Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication,” *IEEE Transactions on parallel and distributed systems*, vol. 10, no. 7, pp. 673–693, 1999.

[56] G. Ballard, A. Druinsky, N. Knight, and O. Schwartz, “Hypergraph partitioning for sparse matrix-matrix multiplication,” *ACM Transactions on Parallel Computing (TOPC)*, vol. 3, no. 3, pp. 1–34, 2016.

[57] R. H. Bisseling, *Parallel Scientific Computation: A Structured Approach Using BSP*. Oxford University Press, USA, 2020.

[58] Ü. V. Çatalyürek and C. Aykanat, "PaToH (Partitioning Tool for Hypergraphs)," in *Encyclopedia of Parallel Computing*, pp. 1479–1487, Springer, 2011.

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

[60] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, pp. 10–pp, IEEE, 2006.

[61] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[62] T. Torun, "Locality aware reordering for sparse triangular solve," Master's thesis, Bilkent university, 2014.

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

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

[65] R. Bagnara, "A unified proof for the convergence of Jacobi and Gauss–Seidel methods," *SIAM review*, vol. 37, no. 1, pp. 93–97, 1995.

[66] G. H. Golub and C. F. Van Loan, "Matrix computations. johns hopkins studies in the mathematical sciences," 1996.

[67] R. A. Harshman and M. E. Lundy, *Research Methods for Multi-Mode Data Analysis*, ch. The PARAFAC model for three-way factor analysis and multidimensional scaling. New York: Praeger, 1984.

[68] K. S. Aggour, A. Gittens, and B. Yener, "Adaptive sketching for fast and convergent canonical polyadic decomposition," in *International Conference on Machine Learning. PMLR*, 2020.

[69] P. Amodio and F. Mazzia, "A parallel Gauss–Seidel method for block tridiagonal linear systems," *SIAM Journal on Scientific Computing*, vol. 16, no. 6, pp. 1451–1461, 1995.

[70] D. P. Koester, S. Ranka, and G. C. Fox, "A parallel Gauss-Seidel algorithm for sparse power system matrices," in *Supercomputing'94: Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*, pp. 184–193, IEEE, 1994.

[71] Y. Shang, "A distributed memory parallel Gauss–Seidel algorithm for linear algebraic systems," *Computers & Mathematics with Applications*, vol. 57, no. 8, pp. 1369–1376, 2009.

[72] K. S. Kang, "Scalable implementation of the parallel multigrid method on massively parallel computers," *Computers & Mathematics with Applications*, vol. 70, no. 11, pp. 2701–2708, 2015.

[73] R. Tavakoli and P. Davami, "A new parallel Gauss–Seidel method based on alternating group explicit method and domain decomposition method," *Applied mathematics and computation*, vol. 188, no. 1, pp. 713–719, 2007.

[74] J. Zhang, "Acceleration of five-point red-black Gauss-Seidel in multigrid for Poisson equation," *Applied Mathematics and Computation*, vol. 80, no. 1, pp. 73–93, 1996.

[75] L. M. Adams and H. F. Jordan, "Is SOR color-blind?," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 2, pp. 490–506, 1986.

[76] G. C. Fox, *Solving problems on concurrent processors.* Old Tappan, NJ; Prentice Hall Inc., 1988.

[77] M. Kawai, T. Iwashita, H. Nakashima, and O. Marques, "Parallel smoother based on block red-black ordering for multigrid Poisson solver," in *International Conference on High Performance Computing for Computational Science*, pp. 292–299, Springer, 2012.

[78] G. Golub and J. M. Ortega, *Scientific computing: an introduction with parallel computing.* Academic Press Professional, Inc., 1993.

[79] D. P. O'Leary, "Ordering schemes for parallel processing of certain mesh problems," *SIAM Journal on Scientific and Statistical Computing*, vol. 5, no. 3, pp. 620–632, 1984.

[80] M. F. Adams, "A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers," in *SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, 2001.

[81] A. Ahmadi, F. Manganiello, A. Khademi, and M. C. Smith, "A parallel Jacobi-embedded Gauss-Seidel method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 6, pp. 1452–1464, 2021.

[82] M. T. Jones and P. E. Plassmann, "Scalable iterative solution of sparse linear systems," *Parallel Computing*, vol. 20, no. 5, pp. 753–773, 1994.

[83] I. S. Duff and G. A. Meurant, "The effect of ordering on preconditioned conjugate gradients," *BIT Numerical Mathematics*, vol. 29, no. 4, pp. 635–657, 1989.

[84] B. F. Smith, P. E. Bjorstad, W. D. Gropp, and J. E. Pasciak, "Domain decomposition: Parallel multilevel methods for elliptic partial differential equations," *SIAM Review*, vol. 40, no. 1, pp. 169–170, 1998.

[85] S. T. Mukhambetzhanov, D. V. Lebedev, N. M. Kassymbek, T. S. Imankulov, B. Matkerim, and D. Z. Akhmed-Zaki, "GMRES based numerical simulation and parallel implementation of multicomponent multiphase flow in porous media," *Cogent Engineering*, vol. 7, no. 1, p. 1785189, 2020.

[86] C. M. Andersen and R. Bro, “Practical aspects of parafac modeling of fluorescence excitation-emission data,” *Journal of Chemometrics*, vol. 17, no. 4, pp. 200–215, 2003.

[87] N. D. Sidiropoulos, R. Bro, and G. B. Giannakis, “Parallel factor analysis in sensor array processing,” *IEEE Transactions on Signal Processing*, vol. 48, pp. 2377–2388, Aug 2000.

[88] A. H. Andersen and W. S. Rayens, “Structure-seeking multilinear methods for the analysis of fmri data,” *NeuroImage*, vol. 22, no. 2, pp. 728 – 739, 2004.

[89] E. Martinez-Montes, P. A. Valdes-Sosa, F. Miwakeichi, R. I. Goldman, and M. S. Cohen, “Concurrent EEG/fMRI analysis by multiway Partial Least Squares,” *NeuroImage*, vol. 22, no. 3, pp. 1023 – 1034, 2004.

[90] A. Shashua and A. Levin, “Linear image coding for regression and classification using the tensor-rank principle,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–42–I–49 vol.1, 2001.

[91] E. Acar, S. A. Çamtepe, M. S. Krishnamoorthy, and B. Yener, *Modeling and Multiway Analysis of Chatroom Tensors*, pp. 256–268. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[92] B. W. Bader, M. W. Berry, and M. Browne, *Discussion Tracking in Enron Email Using PARAFAC*, pp. 147–163. London: Springer London, 2008.

[93] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell, “Toward an architecture for never-ending language learning,” in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[94] Y. Shi, A. Karatzoglou, L. Baltrunas, M. Larson, A. Hanjalic, and N. Oliver, “Tfmap: Optimizing map for top-n context-aware recommendation,” in *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’12, (New York, NY, USA), pp. 155–164, ACM, 2012.

[95] J. H. Choi and S. Vishwanathan, “Dfacto: Distributed factorization of tensors,” in *Advances in Neural Information Processing Systems 27*, pp. 1296–1304, Curran Associates, Inc., 2014.

[96] B. W. Bader and T. G. Kolda, “Efficient MATLAB computations with sparse and factored tensors,” *SIAM Journal on Scientific Computing*, vol. 30, pp. 205–231, December 2007.

[97] U. Kang, E. Papalexakis, A. Harpale, and C. Faloutsos, “Gigatensor: Scaling tensor analysis up by 100 times - algorithms and discoveries,” in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’12, (New York, NY, USA), pp. 316–324, ACM, 2012.

[98] O. Kaya and B. Uçar, “Scalable sparse tensor decompositions in distributed memory systems,” in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC ’15, (New York, NY, USA), pp. 77:1–77:11, ACM, 2015.

[99] W. Austin, G. Ballard, and T. G. Kolda, “Parallel tensor compression for large-scale scientific data,” in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 912–922, May 2016.

[100] S. Acer, E. Kayaaslan, and C. Aykanat, “A hypergraph partitioning model for profile minimization,” *SIAM Journal on Scientific Computing*, vol. 41, no. 1, pp. A83–A108, 2019.

[101] S. Acer, E. Kayaaslan, and C. Aykanat, “A hypergraph partitioning model for profile minimization,” *SIAM Journal on Scientific Computing*, vol. 41, no. 1, pp. A83–A108, 2019.

[102] L. O. Mafteiu-Scai, “The bandwidths of a matrix. a survey of algorithms,” *Annals of West University of Timisoara-Mathematics*, vol. 52, no. 2, pp. 183–223, 2014.

[103] J. Díaz, J. Petit, and M. Serna, “A survey of graph layout problems,” *ACM Computing Surveys (CSUR)*, vol. 34, no. 3, pp. 313–356, 2002.

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

[105] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[106] I. S. Duff and J. Koster, "On algorithms for permuting large entries to the diagonal of a sparse matrix," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 4, pp. 973–996, 2001.

[107] G. Karypis and V. Kumar, "METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 5.1." `http://www.cs.umn.edu/~metis`, 2013.

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

[109] UHEM, "National Center for High Performance Computing." `http://www.uhem.itu.edu.tr`, 2021.

[110] Intel, "Intel Math Kernel Library (MKL)." `https://software.intel.com/en-us/mkl`, 2019.

[111] S. Acer, T. Torun, and C. Aykanat, "Improving medium-grain partitioning for scalable sparse tensor decomposition," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2814–2825, 2018.

[112] E. Horowitz and S. Sahni, *Fundamentals of computer algorithms*. Computer Science Press, 1978.

[113] C. Aykanat, B. B. Cambazoglu, and B. Uçar, "Multi-level direct k-way hypergraph partitioning with multiple constraints and fixed vertices," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 609–625, 2008.

[114] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "Splatt: Efficient and parallel sparse tensor-matrix multiplication," in *2015 IEEE International Parallel and Distributed Processing Symposium*, pp. 61–70, May 2015.

[115] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[116] A. Carlson, J. Betteridge, B. Kisiel, and B. Settles, "Toward an architecture for never-ending language learning.," in *AAAI*, vol. 5, p. 3, 2010.

[117] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, (New York, NY, USA), pp. 1082–1090, ACM, 2011.

[118] J. J. McAuley and J. Leskovec, "From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews," in *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, (New York, NY, USA), pp. 897–908, ACM, 2013.

[119] J. Bennett, S. Lanning, and N. Netflix, "The netflix prize," in *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[120] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, pp. 19:1–19:19, Dec. 2015.

[121] O. Görlitz, S. Sizov, and S. Staab, "Pints: Peer-to-peer infrastructure for tagging systems," in *Proceedings of the 7th International Conference on Peer-to-peer Systems*, IPTPS'08, (Berkeley, CA, USA), pp. 19–19, USENIX Association, 2008.

[122] N. Zheng, Q. Li, S. Liao, and L. Zhang, "Flickr group recommendation based on tensor decomposition," in *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, (New York, NY, USA), pp. 737–738, ACM, 2010.