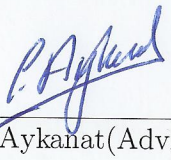
GRAPH/HYPERGRAPH PARTITIONING MODELS FOR SIMULTANEOUS LOAD BALANCING ON COMPUTATION AND DATA

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

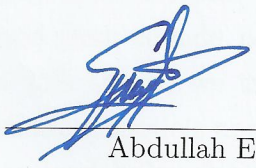
By
Mestan Fırat Çeliktuğ
December 2018

Graph/Hypergraph Partitioning Models for Simultaneous Load Balancing on Computation and Data
By Mestan Fırat Çeliktuğ
December 2018

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



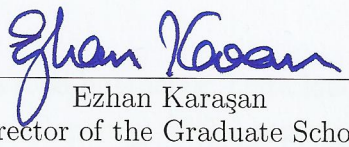
Cevdet Aykanat(Advisor)



Abdullah Ercüment Çiçek

Uraz Yavanoğlu

Approved for the Graduate School of Engineering and Science:



Ezhan Karaşan
Director of the Graduate School

ABSTRACT

GRAPH/HYPERGRAPH PARTITIONING MODELS FOR SIMULTANEOUS LOAD BALANCING ON COMPUTATION AND DATA

Mestan Fırat Çeliktuğ
M.S. in Computer Engineering
Advisor: Cevdet Aykanat
December 2018

In the literature, several successful partitioning models and methods have been proposed and used for computational load balancing of irregularly sparse applications on distributed-memory architectures. However, the literature lacks partitioning models and methods that encode both computational and data load balancing of processors. In this thesis, we try to close this gap by proposing graph and hypergraph partitioning models and methods that simultaneously encode computational and data load balancing of processors. The validity of the proposed models and methods are tested on two widely-used irregularly sparse applications: parallel mesh simulations and parallel sparse matrix sparse matrix multiplication.

Keywords: Computation load balancing, Data load balancing, Simultaneous computation and data load balancing, Multi-constraint graph partitioning , multi-constraint hypergraph partitioning.

ÖZET

Eş ZAMANLI HESAPLAMA VE VERI YÜKÜ DENGELEME IÇIN ÇIZGE/HIPERÇIZGE BÖLÜMLEME MODELLERI

Mestan Fırat Çeliktuğ
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Cevdet Aykanat
Aralık 2018

Literatürde, dağıtık hafıza mimarilerinde düzensiz surette seyrek uygulamaların hesaplama yükünü dengelemek için başarılı bölümleme modelleri ve metotları önerilmiştir ve kullanılmıştır. Ancak, literatür işlemcilerin hem hesaplama hem veri yükünü dengelemeyi amaçlayan bölümleme modelleri ile metotlarından yoksun bulunmaktadır. Bu tezde, işlemcilerin hesaplama ve veri yükünü eşzamanlı olarak dengelemeyi amaçlayan çizge/hiperçizge modelleri ve metotları önererek bu boşluğu kapamaya çalışıyoruz. Önerilen modellerin ve metotların geçerliliği, iki genişçe kullanılan düzensiz şekilde seyrek uygulamada, paralel çözüm ağı (mesh) simülasyonlarında ve paralel seyrek matris seyrek matris çarpımında, test edildi.

Anahtar sözcükler: Hesaplama yükü dengeleme, Veri yükü dengeleme, Eşzamanlı veri ve hesaplama yükü dengeleme, Çok-kısıtlı çizge bölümleme, Çok-kısıtlı hiperçizge bölümleme.

Acknowledgement

I'm thankful to my supervisor Prof. Dr. Cevdet Aykanat since he supported me as M.Sc. student and researcher throughout my M.Sc. period.

I thank Dr. Seher Acer worked as Postdoctoral Fellow in our research group for her wide help and wide support.

I thank Dr. Reha Oğuz Selvitopi worked as Postdoctoral Fellow in our research group for providing us dataset for C=AB kind row-row parallel SPGEMM applications.

I thank to Asst. Prof. Dr. Abdullah Ercüment Çiçek, Asst. Prof. Dr. Uraz Yavanoğlu for reading and commenting on my thesis.

I thank Prof. Dr. Şeref Sağıroğlu, chairman of Gazi University, Computer Engineering Department where I'm actually working as Research Staff for his comprehensiveness, motivational support for completion of the thesis.

I thank my close friends who have supported, encouraged me intellectually and emotionally in terms of completion of the M.Sc. program.

I thank M.Sc. Mehmet Başaran, Ph.D. candidate in our group, for his sincerity, help.

I thank my family for their patience, material, intellectual and moral support throughout my M.Sc. period.

I thank my father who has died on 2009, he always sincerely supported, encouraged me in my education life throughout time we spent together.

Finally, I would like to thank TUBITAK BIDEB scholarship for master students, TUBITAK for supporting me financially during my master program for certain period.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graph/hypergraph partitiong is utilized to partition workload for efficient parallelization of several irregularly sparse applications. In literature, there are successful proposed partitioning models and methods used for computational load balancing of irregularly sparse applications on distributed-memory architectures (e.g., Distributed mesh simulations, distributed sparse-matrix-sparse-matrix generalized multiplication (SPGEMM), distributed sparse-matrix-vector multiplication (SpMV) etc.).

Memory management is also an important issue on distributed memory architectures environments for several irregularly sparse applications. Therefore, memory footprint, which refers to total amount of data required for the computations associated with a processor, is also an important issue while handling given computation duties. So, simultaneous computation load and data size balancing is an important objective for the partitioning of the graphs/hypergraphs representing the sparse applications.

In literature, simultaneous computation and data load balancing on given application set is known in scope of multi-constraint graph/hypergraph partitioning problem as two-constraint graph/hypergraph partitioning problem, i.e., Computation and data load constraints to be abided. In [2, 3], this problem is addressed

in certain form for distributed parallel mesh simulations under given memory constraints. However, partitioning models and methods that encode both computational and data load balancing of processors is lacking in literature. In this thesis, we tried to contribute on this need by proposing graph/hypergraph partitioning models and methods that encode simultaneous computation and data load balancing for processors. Empirical validity of the proposed models and methods are confirmed by partitioning experiments performed on a wide range of irregularly sparse matrices, i.e., on dataset certainly related to distributed mesh simulations, dataset of C = AB kind row-row parallel SPGEMM application.

Rest of this thesis is organized as in following: Related work, problem definition are given in Chapter 2. The proposed graph/hypergraph partitioning models for simultaneous computation and data load balancing are described in Chapter 3. Experimental details, experimental results are presented and discussed in Chapter 4. Finally, this thesis is concluded in Chapter 5.

Chapter 2

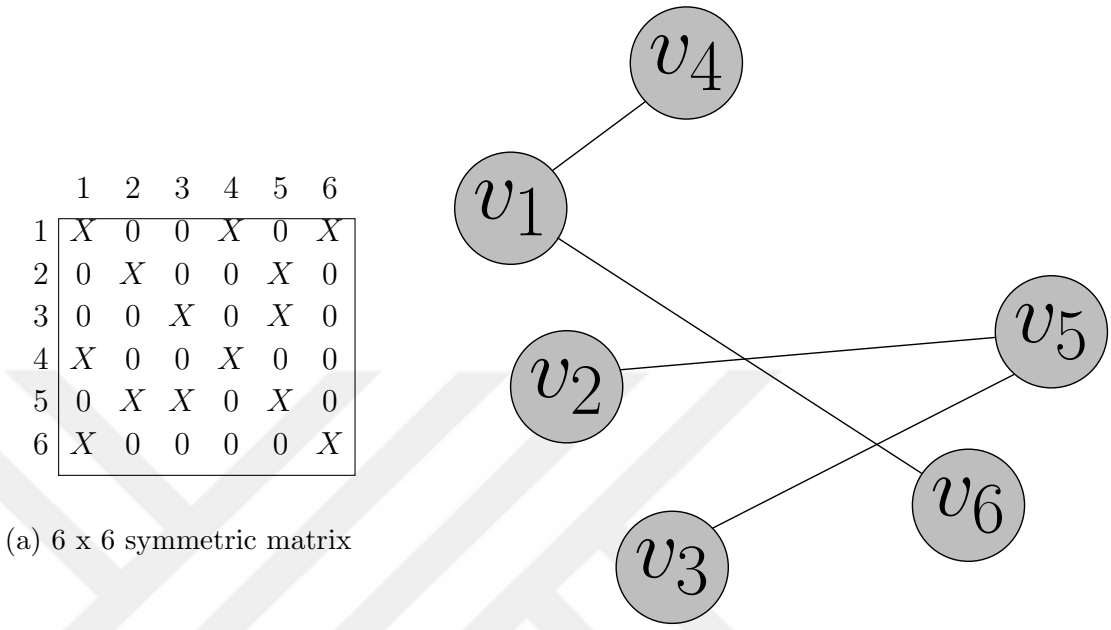
Related Work and Problem Definition

Various high performance computing applications such as mesh computations based on finite-element methods (FEMs) and finite-volume element methods (VEMs), sparse matrix sparse matrix generalized multiply (SPGEMM) and many other applications (e.g. Sparse Matrix Vector Multiply (SpMV) etc.) could be distributed to processors for parallellization [2, 3, 4]. In the thesis, we focus on partitioning of mesh computations and the row-row parallel SPGEMM [4]. We aim to construct balanced partitioning for both data and computation load according to predetermined imbalance ratio while minimizing the cutsize. It's important to note that mesh computation has a wide range of application fields. Also, SPGEMM has various different application fields such as graph operations [5, 6, 7, 8, 9, 10, 11, 12], linear programming [13, 14, 15], molecular dynamics [16, 17, 18, 19, 20, 21, 22, 23, 24], recommender systems [25] as noted in [26] in detail. There are various partitioning schemes for both applications in the literature. Here, we investigate specifically mesh partitioning under memory constraints, multi-constraint (or multi-criteria) graph partitioning for multi-physics simulation.

In literature, mesh partitioning under memory constraints problem is studied in two studies [2, 3]. In physical applications, data can be represented by mesh $M = (C_m, N_m)$ [2, 3]. C_m signifies the set of mesh cells whereas N_m signifies the per-edge or per-face neighborhood relationship. The mesh M computations are completed in the cell level (e.g., within triangles and quadrilaterals in two dimensional-mesh, or within tetrahedrons and hexahedrons in three dimensional-mesh) [2]. Each mesh $cell_i \in C_m$ represents a computation, also each mesh $cell_i \in C_m$ needs data and/or is given as data to its neighbor mesh cells ($\forall cell_j \in N(i)$) originating from the geometrical mesh topology [2]. It's interesting to note that there is a one-to-one correspondence in terms of application model with [27, 28, 29] in this regard. There are tasks (or computations) to be handled and there are files (or data) that would be given for respective tasks [27, 28, 29]. In distributed memory parallel mesh computations, there are multiple processors running concurrently. Each parallel processor contains several cells in a mesh partitioning, so each processor could need other cells, i.e., ghost cells, from other parts (or other processors) as said. The ghost cells are reported to affect dramatically memory footprint of a processor [2]. Each processor's memory footprint is tried to be bounded according to pre-determined memory bounds. In this configuration, common mesh M partitioning problem formulation in [2, 3] is to have a partitioning of M such that maximum computation among processors is minimized and memory footprint of each processor stays in given memory bounds for each processor. In the formulation, M is represented by a bipartite graph model in order to differ memory costs and computation costs [2, 3]. So, the problem becomes a certain type of graph partitioning problem. We use the representation in Section 3.4, 3.5. Multi-level partitioning tools have been developped such as Chaco [30], METIS [31, 32], ParMETIS [33, 34], Party [35], Scotch [36, 37], Pt-Scotch [38, 39] , Jostle [40, 41], DRAMA [42], PaToH [1, 43], MLPart [44], Mondriaan [45] hMetis [46, 47] Parkway [48, 49]. Two issues seem problematic in [3]. It's impractical to have a partitioning in which each processor's given memory bound is controlled. Since, it's hard to respect each processor's memory bound. Rather, it's practical, better to minimize the maximum data load as done in computation constraint. The partitioner's empirical validity is tested in very narrow instance space consisting of 3 instances in [3].

Multi-constraint (or multi-criteria) graph partitioning was studied widely. Nevertheless, the partitioning for multi-physics simulation is specially addressed in [50, 51]. They propose two new multi-constraint (or multi-criteria) graph partitioning tools such as Crack and new release of Scotch that could deal with multi-constraint graph partitioning [50, 51] (an uncommon use of 'multi-criteria' expression in lieu of generally utilized 'multi-constraint' expression in literature [52, 53, 54, 55, 56]). There are three-constraint partitioning experiments for accelerating multiphysics Particle-In-Cell(PIC) simulation experiments in which they try to balance computational and memory (or data) weight costs, at the same time one additional unit weight cost [50]. This is very close to the problem studied in the thesis. Nevertheless, as there is no guarantee of data needed by each processor would not create a memory overflow, the proposed partitioning doesn't correspond one-to-one [2, 3]. We used generated computation and data weighting scheme of [50] in mesh weighting (Section 4.2.1). Again, instance space is narrow to test empirical validity.

In distributed $C = AB$ kind row-row parallel SPGEMM, v_x represents each the pre-multiply of row x of A with matrix B such as $c_{x,*} = a_{x,*}B$ computation, and requires respective rows of B as data to complete its computation [4, 26]. Yet to our knowledge, there is no study in which memory footprint is tried to be balanced with synchronous effort of balancing maximum computation load for the C = AB kind SPGEMM computation as opposed to the mesh partitioning schemes presented above [2, 3, 50, 51]. Differing properties of the addressed applications in the thesis are twofold, such as structural patterns of the matrices representing the computations, and weighting schemes arising from computation natures (Section 4.2.1, Section 4.2.2). For both applications, we try to construct a balance on maximum computation and data load by modelling the problem as graph/hypergraph partitioning problem, different from [2, 3] similar to [50, 51], with new ideas such as split data vertex replication and inverse data weight distribution.



(a) 6 x 6 symmetric matrix

(b) Undirected graph representation of the matrix

Figure 2.1: 6 x 6 symmetric matrix, its undirected graph representation

2.1 Problem Definition

In this section, we describe graph/hypergraph partitioning problems with their instances we studied, i.e., two-constraint graph partitioning (Section 3.4, 3.5), single-constraint hypergraph partitioning (Section 3.1) and two-constraint hypergraph partitioning (Section 3.2, 3.3, 3.6, 3.7).

2.1.1 Graph Partitioning Problem

A graph $G = (\mathcal{V}, \mathcal{E})$ is a data structure which is composed of a set $\mathcal{V}$ of vertices and a set $\mathcal{E}$ of edges. An example of undirected graph drawing which representing a matrix is illustrated in Fig. 2.1. In Fig. 2.1, vertex $v_i \in \mathcal{V}$ is represented with gray circles, and undirected edge $e_u \in \mathcal{E}$ is represented with with line segment.

Each edge e_{ij} connects a pair of distinct vertices v_i and v_j. A cost c_i is assigned to each edge e_{ij}. Each vertex has its adjaceny list, $Adj(v_i)$ denotes the set of its

neighbors, that is,

$$Adj(v_i) = \{v_j : e_{ij} \in \mathcal{E}\} \tag{2.1}$$

Multiple weights $w^1(v_i), ..., w^C(v_i)$ are assigned to each vertex v_i, where $w^c(v_i)$ referring the cth weight assigned to $w^c(v_i)$.

$\Pi(G) = \{P_1, ..., P_K\}$ is called a K-way vertex partition of G if parts are mutually disjoint and mutually exhaustive.

In $\Pi(G)$, an edge e_{ij} is defined as cut (external) if vertices v_i and v_j are located in different parts, and defined as uncut (internal) otherwise. A vertex v_i in $\Pi(G)$ is defined as a boundary vertex if it is connected by at least one cut edge.

The cutsize of $\Pi(G)$ is:

$$\sum_{e_{ij} \in \mathcal{E}_{ec}} c_(n_j), \tag{2.2}$$

where $\mathcal{E}_{ec}$ is the set of cut edges.

$W^c(P_k)$ of part P_k is defined as the sum of the cth weights of the vertices in P_k.

A partition $\Pi(G)$ is defined as balanced if

$$W^c(P_k) \leq W^c_{avg}(1 + \epsilon^c) \; k \in \{1, 2, ..., K\} \text{ and } c \in \{1, 2, ..., C\}, \tag{2.3}$$

where $W^c_{avg} = (\sum_{i=1}^{K} W^c(P_i))/K$, and ϵ^c is the predetermined imbalance ratio for the cth weight.

The K-way multi-constraint graph partitioning problem [54, 56] is then defined as finding a K-way vertex partition such that the cutsize is minimized while the balance constraint (2.3) is maintained. Graph partitioning problems are known to be NP-hard [57, 58]

In our proposed bipartite graph partitioning models (Section 3.4, Section 3.5), we have $C = 2$, i.e., first constraint for computation load and second for data requirement. We use well known, widely used, the state-of-the-art graph/mesh partitioner METIS [31, 32] for achieving a partition $\Pi(G)$ in the models (Section 3.4, Section 3.5).

2.1.2 Hypergraph Partitioning Problem

A hypergraph $H = (\mathcal{V}, \mathcal{N})$ is a generalization of the concept of a graph and is composed of a set $\mathcal{V}$ of vertices and a set $\mathcal{N}$ of nets, i.e., $\forall n_j \in \mathcal{N}$, $n_j \subset \mathcal{V}$ and known also as hyperedge. An example of a hypergraph representing the neighboorhood structure of 2D mesh of 6 cells is illustrated in Fig. 2.2. Vertex $\forall v_i \in \mathcal{V}$ is represented with gray circles, $\forall n_j \in \mathcal{N}$ is represented with black point and line segments (See Fig. 2.2).

Each net $n_j \in \mathcal{N}$ connects a subset of vertices defined as $Pins(n_j)$. A cost $c(n_j)$ is assigned to each net n_j. $Nets(v_i)$ defines the set of nets that connect v_i.

Multiple weights $w^1(v_i), ..., w^C(v_i)$ are assigned to each vertex v_i, where $w^c(v_i)$ referring to the cth weight assigned to vertex v_i.

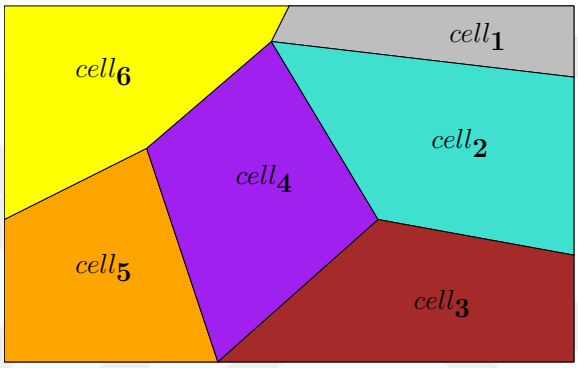
$\Pi(H) = \{P_1, ..., P_K\}$ is called a K-way vertex partition of H if parts are mutually disjoint and mutually exhaustive.

In $\Pi(H)$, the connectivity set $\bigwedge(n_j)$ of net n_j is composed of the parts which are connected by the respective net, that is,

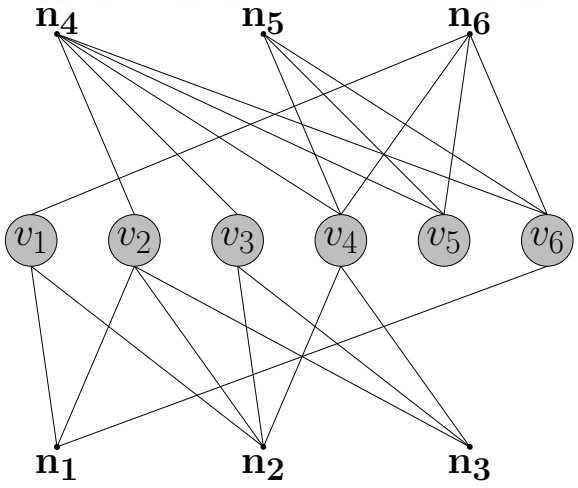
$$\bigwedge(n_j) = \{P_k : Pins(n_j) \cap P_k \neq \emptyset\}. \tag{2.4}$$

The number of parts connected by n_j is defined as:

$$\diagup(n_j) = |\bigwedge(n_j)|. \tag{2.5}$$



(a) Representation of 2D mesh of 6-cells



(b) Hypergraph representation of the mesh

Figure 2.2: The mesh, its hypergraph representation

A net n_j is defined as cut (external) if it connects more than one part, i.e., $\curlywedge(n_j) > 1$, and is defined uncut (internal) otherwise.

The cutsize of $\Pi(H)$ is defined as:

$$\sum_{n_j \in N} c(n_j)(\curlywedge(n_j) - 1). \tag{2.6}$$

A vertex v_i in $\Pi(H)$ is defined as a boundary vertex if it is connected by at least one cut net.

Definition of the weight $W^c(P_k)$ of part P_k is defined as the sum of the cth weights of the vertices in P_k.

A partition $\Pi(H)$ is said to be balanced if

$$W^c(P_k) \leq W^c_{avg}(1 + \epsilon^c) \; k \in \{1, ..., K\} \text{ and } c \in \{1, ..., C\}, \tag{2.7}$$

where $W^c_{avg} = (\sum_{i=1}^{K} W^c(P_i))/K$, and ϵ^c is the predetermined imbalance ratio for the cth weight.

The K-way multi-constraint hypergraph partitioning problem [55, 52, 53] is then defined as finding a K-way vertex partition such that the cutsize is minimized while the balance constraint (2.7) is maintained. The hypergraph partitioning problem is known to be NP-hard [57, 58].

We note that for $C = 1$, problem is reduced to well-studied standard hypergraph partitioning problem. This is our baseline model, i.e., only constraint for computation load (Section 3.1). In our proposed hypergraph partitioning models (Section 3.2, Section 3.3, Section 3.6, Section 3.7), we have $C = 2$, where the first constraint for computation load and second for data size. We use the well known, state-of-the-art hypergraph partitioner PATOH [1, 43] for acquiring a partition $\Pi(H)$ in the models (Section 3.2, Section 3.3, Section 3.6, Section 3.7).

Chapter 3

Proposed Graph/Hypergraph Partitioning Models for Simultaneous Computation Load and Data Size Balance

The total amount of data usage of processors is not constant while the total amount of computation of processors is constant and the data amount depends on the quality of the task partitioning. Therefore, a naive balancing on the data weights of the parts will not encode the minimization of the data size of the maximally loaded processor. For instance, a very tight balance on the data sizes of the processors might yield a very high max processor data size if the underlying partition produces a high amount of data replication. The partitioning objective of minimizing the cutsize in graph/hypergraph partitioning problems encode the minimization of the total amount of data replication via clustering the tasks that require the same data elements to the same parts under the given balancing constraints. In this regard, the main contributions to the literature are split data vertex replication and inverse data weight distribution. Split data vertex replication here has two implications, one for graph partitioning and another for hypergraph partitioning. In both cases, recursive bipartitioning (RB) framework

is constructed by using as-is a state-of-art partitioning tool, i.e., METIS/PATOH, for two-way partitioning. In these RB frameworks, two sub-graph/hypergraphs are formed from previously acquired partitions. In these sub-graphs/hypergraphs, we adopt the replication of boundary data vertices in the bipartition. Replicated data vertices are evaluated as other "normal" vertices. Their edges/pins are put to new sub-graph/hypergraphs. When we obtained a K-way partition by the RB frameworks, we calculate computation load and data size of processors by using initial graph/hypergraph adjacency information. Inverse data weight distribution denotes distribution of net weights onto computation vertices' data weights. The distribution is handled either before K-way direct partitioning or before/during RB of the hypergraph. In the RB framework, PATOH is used as-is for two-way partitioning as above. When two new sub-hypergraphs are built in each step of the RB framework, the distribution is made from scratch for each sub-hypergraph. After acquisition of a K-way partition, computation load and data size of processors are calculated as above, with a general method. Baseline model will be described first. Then, split data vertex replication-related models and inverse data weight distribution related models will be presented in order.

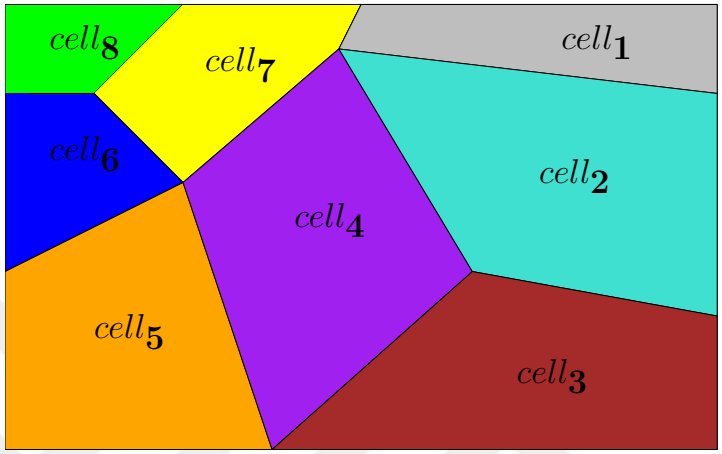
3.1 Baseline Model: Single-Constraint Hypergraph Partitioning Model

A mesh M which results from Finite Element Method (FEM) or Volume Element Method (VEM) or another application is demanded to be partitioned for quicker solution of a mesh. In this regard, data load and computation load is aimed to be balanced as aforemetioned. A mesh $M = (C_m, N_m)$, where C_m represents cell set and N_m represents neighborhood relation, could be modelled as hypergraph $H_M = (V_M, N_M)$ to model computation load and data requirement simultaneously. The hypergraph could vary structurally and in terms of number of constraints. In our hypergraph models, there are at most two constraints, i.e., one for computation constraint and another for data constraint. In this section, single-constraint hypergraph model $H_{Sc} = (V_{Sc}, N_{Sc})$ will be described in detail.

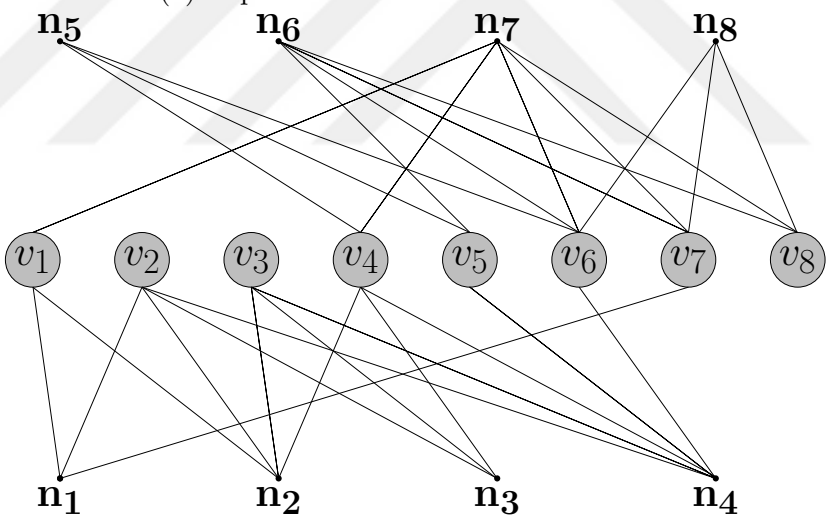
In the model, V_{Sc} represents computations and N_{Sc} represents data requirement. There is an exemplary mesh composed of eight (8) cells in Fig. 3.1a. Each cell is coloured by one different colour. Its correspondent single-constraint hypergraph H_{Sc} is shown in Fig. 3.1b.

Each cell in C_m expresses a single computation task. Each cell in C_m corresponds to a vertex V_{Sc}. $v_1 \in V_{Sc}$ corresponds to $cell_1 \in C_m$ as seen in Fig. 3.1. The same correspondence is valid between $v_2 \in V_{Sc}$ and $cell_2 \in C_m$ and so on. Conclusively, $v_i \in V_{Sc}$ corresponds to $cell_i \in C_m$. Neighborhood denotes data requirement of a cell. Since, each cell needs neighbor cells as data, i.e., cells sharing an edge, a face, a region with respective cell, for solution of a mesh. Each face, edge or a region that a cell shares with a neighbor cell is represented by a pin in its net (or hyperedge). In other words, net of $cell_i$ (i.e. n_i) connects all neighbor cells (or vertices) of $cell_i$ as well as $cell_i$. In Fig. 3.1a, $cell_2, cell_7$ are neighbor cells of $cell_1$, n_1 connects v_1, v_2, v_7 respectively. Utilized weighting scheme in Fig. 3.1 belongs to certain type of mesh computation described in Section 4.2.1. In $C = AB$ kind row-row parallel SPGEMM, vertex v_x represents $c_{x,*} = a_{x,*}B$ computation. Net n_i represents each row i of B, where n_i captures data dependency of $a_{x,}B$ computations to $b_{i,}$ and conclusively n_i connects vertex v_x such that $v_x \in rows(a_{*,i})$ [4]. The product's weighting has its own pattern described in Section 4.2.2.

We could use direct k-way hypergraph partitioning or recursive bipartitioning (RB) after having constructed the model H_{Sc}, we preferred RB as shown in Algorithm 1. We handle cut-net splitting as described in [1]. This single-constraint partitioning model tries to construct a balance on the computation loads of the processors. We couldn't build balancing on the data sizes of the processors as a baseline, since data partition is depending on task partition, but reverse dependency is invalid as described.



(a) Representation of 2D mesh of 8 cells



i	$w(v_i)$	$c(n_i)$
1	100	10
2	144	12
3	169	13
4	196	14
5	225	15
6	256	16
7	289	17
8	324	18

(b) Single-Constraint Hypergraph representation of the mesh

Figure 3.1: Mesh and its single-constraint hypergraph representation

Algorithm 1 Single-Constraint Hypergraph Partitioning Algorithm

Input : **Matrix M (representing a mesh or the B in the C = AB kind SPGEMM)**
Output: **Partition vector, Computed performance metrics (Section 4.4.1)**

1: H_{Sc} = ConstructSingleConsHypergraph(M);
2: *partVec* = PartitionRBSingleConsHypergraph(H_{Sc});
3: *perMetriCvec* = ComputePerformanceMetric(*partVec*);
4: **return** *partVec*, *perMetricVec*

3.2 Data-Vertex Based Direct K-way Hypergraph Partitioning Model

This model $H_{DVB_{Dr}} = (V_{DVB_{Dr}}, N_{DVB_{Dr})}$ is initial hypergraph to be used for applying split data vertex replication on hypergraphs. There are two main differences between the single-constraint hypergraph model H_{Sc} and the $H_{DVB_{Dr}}$.

$H_{DVB_{Dr}}$ has two constraints, the first for computation and the second for data, while H_{Sc} has single constraint, only for computation (See Fig. 3.2 and Fig. 3.1). In contrast to H_{Sc}, $H_{DVB_{Dr}}$ has vertex type differentiation. It has two types of vertices such as computation vertex (v_i^C) and data vertex (v_i^D) (See Fig. 3.2). As seen from Fig. 3.2 and Fig. 3.1, computation vertex v_i^C in $H_{DVB_{Dr}}$ corresponds to vertex v_i structurally where $i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. That is, both vertices represent same single computation work in the mesh (Fig. 3.1a). Naturally, their computation constraints are equal to each other, i.e., $w^1(v_i^C) = w^1(v_i)$. Nevertheless, v_i^C has additionally data constraint $w^2(v_i^C) = 0$ in the mesh application, positive data constraint in the SPGEMM application (Section 4.2.2). The data vertex directly correlates with formation of nets and respective net weight. As known, each net represents the data requirement of the vertices it connects in our hypergraph models. Naturally, characteristic of data vertex v_i^D is having data constraint equals to respective net weight $c(n_i)$, i.e., $w^2(v_i^D) = c(n_i)$, whereas its computation constraint is zero (See Fig. 3.2). As seen from Fig. 3.2, each net n_i connects one data vertex v_i^D having the same weight. It denotes data is two-fold represented in the $H_{DVB_{Dr}}$.

We use direct k-way hypergraph partitioning for the $H_{DVB_{Dr}}$. The partitioning

model aims at constructing a balance on the both computation load and data size of the processors. It is a two-constraint direct k-way hypergraph partitioning model. The partitioning procedure is presented as an overview in Algorithm 2.

Algorithm 2 Data Vertex Based Direct K-Way Hypergraph Partitioning Algorithm

Input : **Matrix M (representing Mesh cells or B in C = AB kind SPGEMM)**
Output: **Partition vector, Computed performance metrics (Section 4.4.1)**

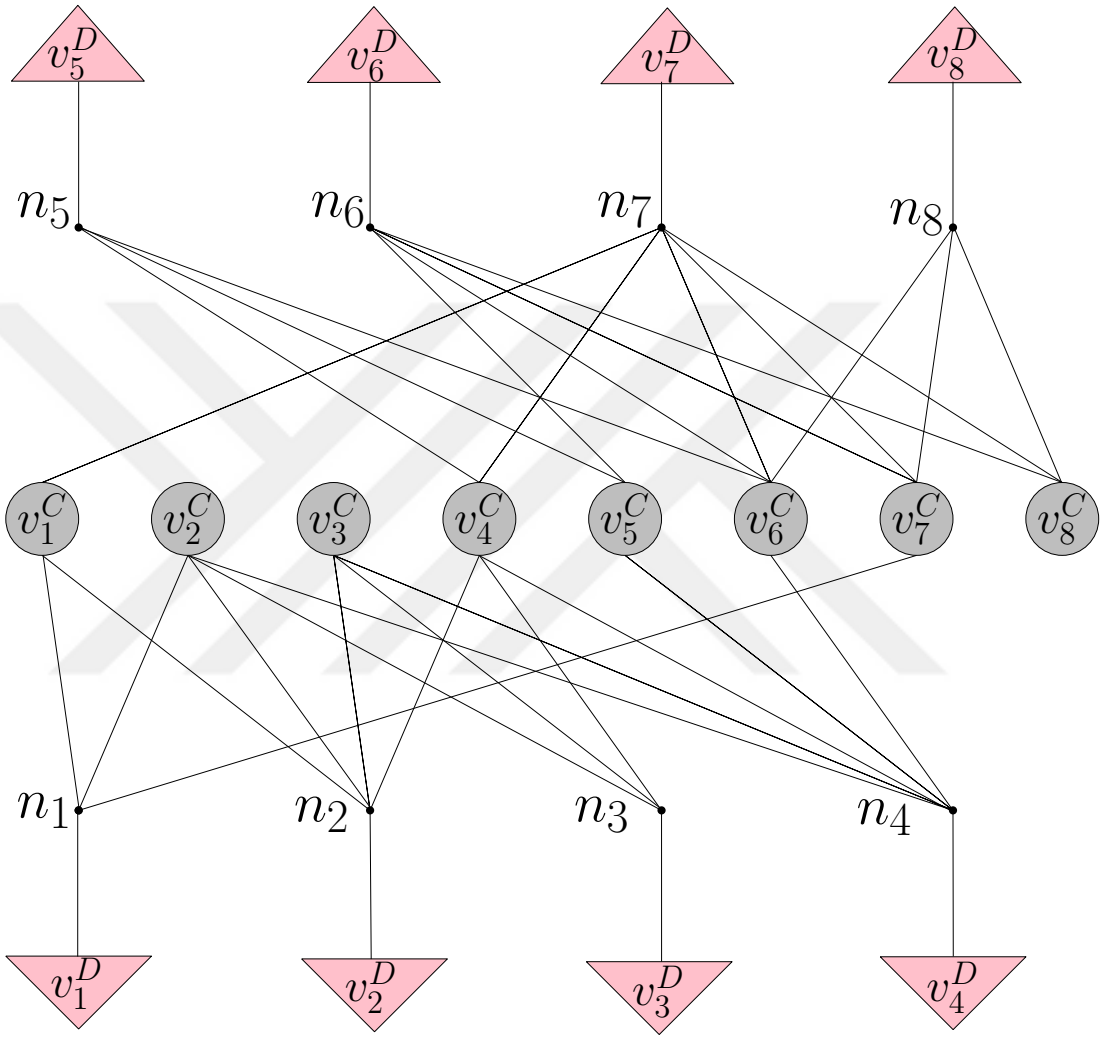
1: $H_{DVB_{Dr}}$ = ConstructDVBHypergraph(M);
2: *partVec* = PartitionDirectDVBHypergraph($H_{DVB_{Dr}}$);
3: *perMetriCvec* = ComputePerformanceMetric(*partVec*);
4: **return** *partVec*, *perMetriCvec*

3.3 Data-Vertex Replication Based Recursive Hypergraph Bipartitioning Model

$H_{DVB_{Dr}}$ is bipartitioned through the RB process to obtain sub-hypergraphs (Fig. 3.2). Intuitively, idea behind the RB framework is benefiting from certain useful information needed for improved partitioning objective while it may not be acquired by the direct k-way partitioning. In this regard, replication is utilized for improving the objective [59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72]. Main contribution of our RB framework is the split data vertex replication on hypergraphs to improve the objective.

3.3.1 Split Data Vertex Replication on Hypergraphs and the RB Framework

In this section, we describe split data vertex replication during RB of the hypergraphs and the RB framework in detail. We use PATOH as-is for two-way partitioning in each step of the framework. In two-way split data-vertex replicated partition $\Pi^{d_v R} = (V_L, V_R)$, a vertex (either computation or data) could belong to V_L, V_R or only a data vertex could belong to both of them. We use



$\mathbf{i}$	$\mathbf{w^1(v_i^C)}$	$\mathbf{w^2(v_i^C)}$	$\mathbf{w^1(v_i^D)}$	$\mathbf{w^2(v_i^D)}$	$\mathbf{c(n_i)}$
1	100	0	0	10	10
2	144	0	0	12	12
3	169	0	0	13	13
4	196	0	0	14	14
5	225	0	0	15	15
6	256	0	0	16	16
7	289	0	0	17	17
8	324	0	0	18	18

Figure 3.2: Data-vertex based hypergraph representation of the mesh

letters L or R to denote the parts of the bipartition. During the formation of two new sub-hypergraphs (H_L, H_R) in RB, critical operation is split data vertex replication. All other issues for the formation are handled according to [1, 43].

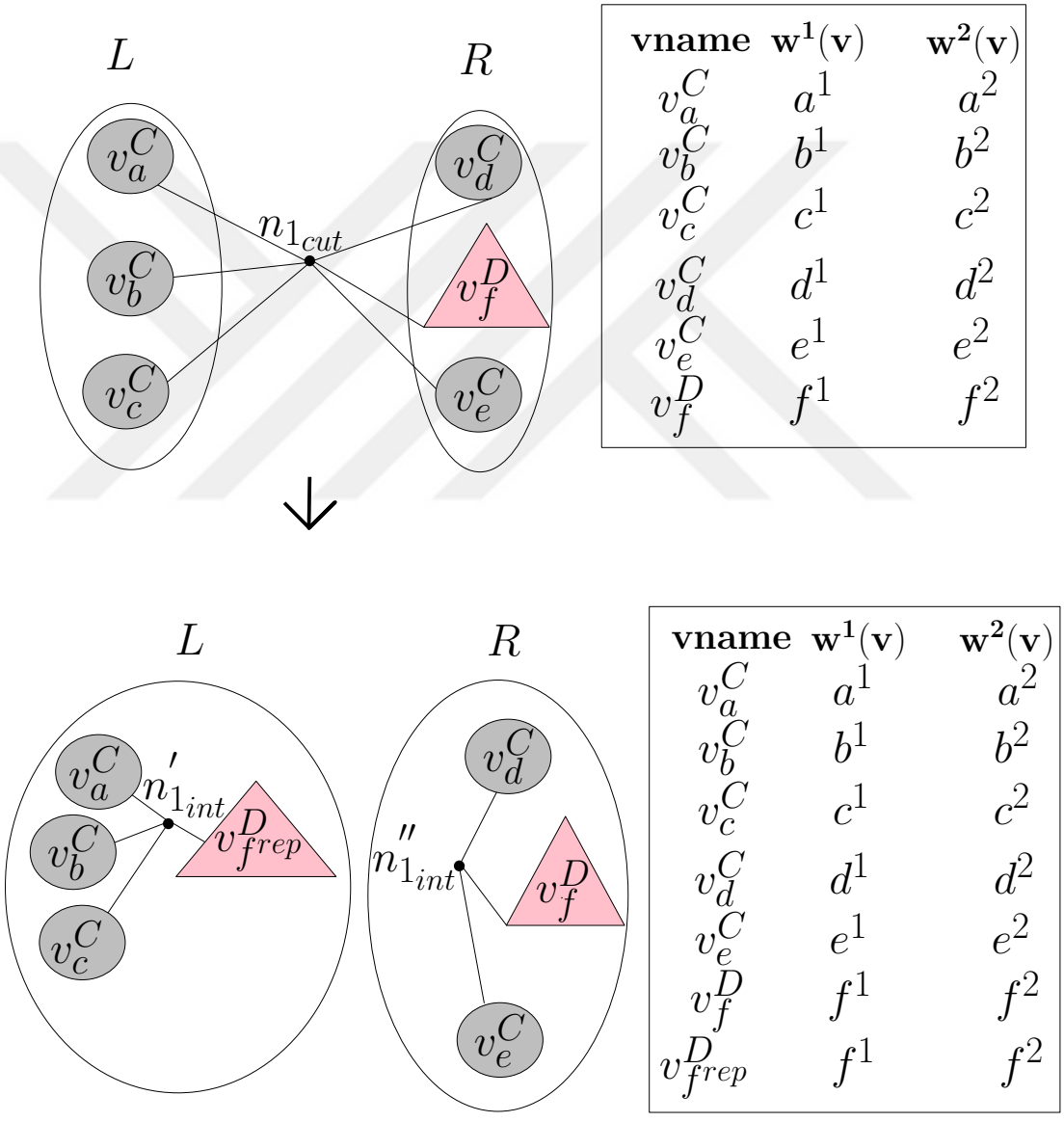
Exemplary split data vertex replication is described in Fig. 3.3 in absence of cut-net splitting anomaly. Data vertex v_f^D in cut-net $n_{1_{cut}}$ belongs to V_R (Fig. 3.3). If the cut-net is splitted like [1], $n_{1_{int}}^{''}$ will have v_f^D on V_R, but $n_{1_{int}}^{'}$ won't have any data vertex on V_L. However, our model requires replication for each cut-net. Therefore, v_f^D is replicated to $n_{1_{int}}^{'}$ as v_{frep}^D and v_{frep}^D is evaluated as other vertices in $n_{1_{int}}^{'}$, its pin is added $n_{1_{int}}^{'}$ as seen in Fig. 3.3. This split data vertex replication scheme is applied as described for each cut-net in every RB step in absence of the anomaly. Via the replication, computation load and its required data size are held correctly in computation and data weights (i.e., $w^1(v_i)$, $w^2(v_i)$) in every RB step. RB for each sub-hypergraph goes on until desired K-way partition is acquired. As said, there are cut-net splitting anomalies. Their handling is done for decreasing the cutsize, i.e., decreasing the replication and conclusively total data size. The handling of these anomalies is described in following.

3.3.1.1 Cut-net Splitting Anomalies in the Split Data Vertex Replication

There are three (3) cut-net splitting anomalies which could increase cutsize during the RB. Their handling is done for preventing the increase and each anomaly, its handling are explained separately in next three sections.

3.3.1.1.1 Cut-net Splitting Anomaly-1 (Single Data Vertex Anomaly)

The handling of data vertex neighborhood anomaly is described in Algorithm 3. One data vertex v_j^D represents data needed for computation vertices v_i^C such that $v_i^C \in n_j$. Cut-net n_{j_c} usually denotes there are two subsets of vertices in n_{j_c} requiring same data vertex v_j^D for their computations in our RB framework. So, v_j^D is replicated onto either L or R in RB as described (See Fig. 3.3). However, when the data vertex v_j^D is on one part such as R and all other computation



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_d^C	d^1	d^2
v_e^C	e^1	e^2
v_f^D	f^1	f^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_d^C	d^1	d^2
v_e^C	e^1	e^2
v_f^D	f^1	f^2
$v_{f^{rep}}^D$	f^1	f^2

Figure 3.3: Split Data Vertex Replication in RB of $H_{DVB_{Dr}}$

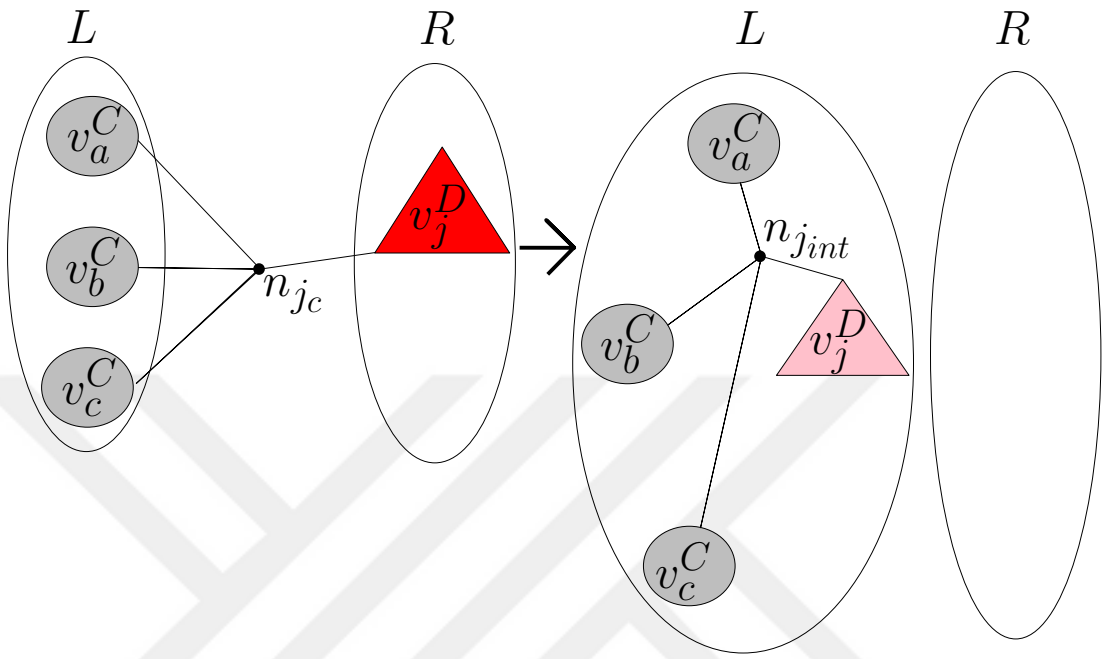


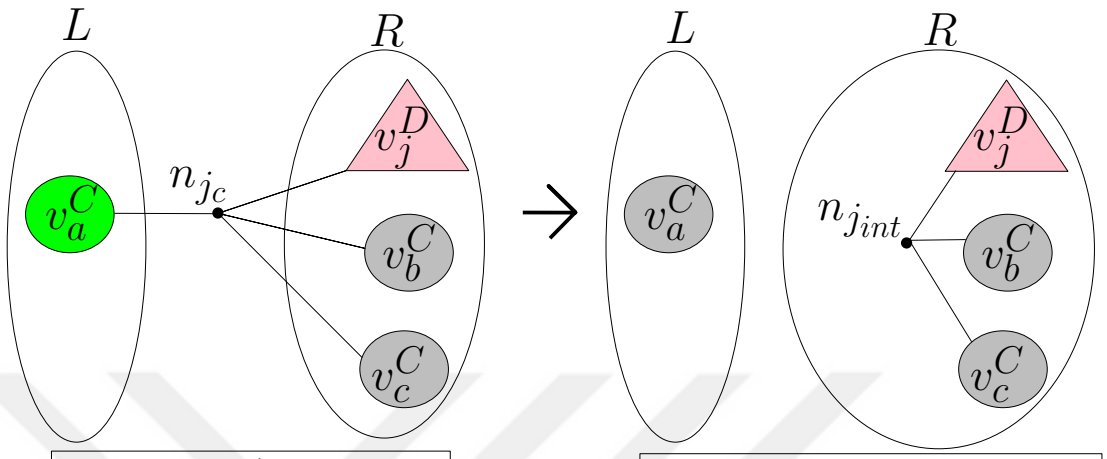
Figure 3.4: Red triangle: Single data vertex anomaly, The Anomaly-1 Handling

vertices of n_{j_c}, i.e., $v_i^C \in Pins(n_{j_c}) \setminus v_j^D$, are on L (See Fig. 3.4). Then, there is no need for the replication. Since, there is no v_i^C requiring v_j^D on R. The v_j^D's part is changed to R (Alg. 3, Line 5-7), n_{j_c} becomes internal net $n_{j_{int}}$. This move affects given cutsize, imbalance negatively. Therefore, cutsize computation is done after the move accordingly.

Algorithm 3 Cut-net Splitting Anomaly-1 Handling

Input : **Partition vector** partVec, **Hypergraph to be BP** $H_{DVB_{Dr}}$
Output: **Partition vector** partVec

1: **for** each cut-net n_{j_c} in $\mathcal{N}_c$ **do**
2: lDCnt $\leftarrow$ 0, rDCnt $\leftarrow$ 0, lCCnt $\leftarrow$ 0, rCCnt $\leftarrow$ 0;
3: countLRCvDv(lDCnt, rDCnt, lCCnt, rCCnt);
4: // Evaluate the data and computation vertex numbers computed above
5: **if** (lDCnt == 1 && rCCnt $\geq$ 1) or (rDCnt == 0 && lCCnt $\geq$ 1) **then**
6: changeDvPart(v_j^D, partVec)
7: **end if**
8: **end for**



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_j^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_j^D	d^1	d^2

Figure 3.5: Green circle: Single computation vertex anomaly, The Anomaly-2 Handling

3.3.1.1.2 Cut-net Splitting Anomaly-2 (Single Computation Vertex Anomaly)

Single computation vertex anomaly handling is described in Algorithm 4. If cut-net n_{j_c} contains a single computation vertex v_a^C in one part such as L and all other vertices are on other part R (See Fig. 3.5), the replication is expected. However, if replication is done, the net to be created on part R during cut-net splitting will connect only one computation vertex v_a^C and single data vertex $v_{j^{rep}}^D$. The net would be highly probably cut-net in next RB step(s), which would increase cutsize unnecessarily. In lieu of the replication, the data weight $w^2(v_j^D) = d^2$ of the data vertex v_j^D is compressed by summation onto the single computation vertex v_a^C data weight $w^2(v_a^C) = a^2 + d^2$ as seen in Fig. 3.5. After the handling, total computation and total data weight are preserved for L and R.

Algorithm 4 Cut-net Splitting Anomaly-2 Handling

Input : **Partition vector** partVec, **Hypergraph to be BP** $H_{DVB_{Dr}}$
Output: **Vertex weight vectors**

1: **for** each cut-net n_{j_c} in $\mathcal{N}_c$ **do**
2: totalVCnt ← 0, lDCnt ← 0, rDCnt ← 0, lCCnt ← 0, rCCnt ← 0;
3: countTotalVLRCvDv(totalVCnt, lDCnt, rDCnt, lCCnt, rCCnt);
4: // Evaluate number of vertices computed above
5: **if** (lCCnt == 1 && (rCCnt + rDCnt) == totalVCnt - 1) or Inverse **then**
6: comressDWeight(v^D_{jc}, $Singv^C_{jint}$)
7: **end if**
8: **end for**

3.3.1.1.3 Cut-net Splitting Anomaly-3 in the Hypergraph RB (Single Computation and Single Data Vertex Anomaly)

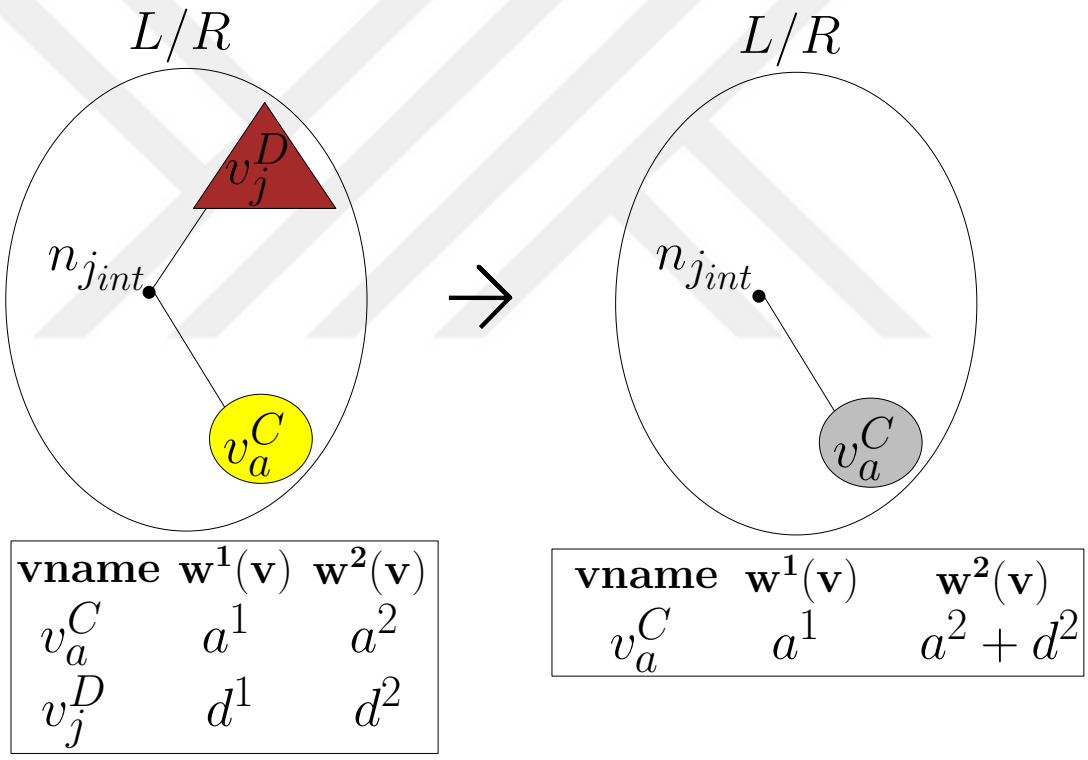
The anomaly's handling is described in Algorithm 5. In the framework, a cut-net is splitted into L and R. Either the L or the R of the cut-net could connect single computation vertex v^C_a and single data vertex v^D_j (See Fig. 3.6). It may lead to useless increase in cutsize as described. The same compression operation onto v^C_a data weight $w^2(v^C_a) = a^2 + d^2$, described in Section 3.3.1.1.2, is applied as seen in Fig. 3.6. In the handling of this anomaly, the data vertex v^D_j is deleted in the internal net, after the replication if the replication is required. This handling doesn't disturb preservation of total data and computation weight.

Algorithm 5 Cut-net Splitting Anomaly-3 Handling

Input : **Partition vector** partVec, **Hypergraph to be BP** $H_{DVB_{Dr}}$
Output: **Vertex weight vectors**

1: **for** each cut-net n_{j_c} in $\mathcal{N}_c$ **do**
2: lDCnt ← 0, rDCnt ← 0, lCCnt ← 0, rCCnt ← 0;
3: countLRCvDv(lDCnt, rDCnt, lCCnt, rCCnt);
4: // Evaluate the data and computation vertex numbers computed above
5: **if** (lDCnt == 1 && lCCnt == 1 or rDCnt == 0 && rCCnt == 1) **then**
6: comressDWeight(v^D_{jint}, $Singv^C_{jint}$)
7: deleteDV(v^D_{jint}, $n_{j_{int}}$)
8: **end if**
9: **end for**

There are two important points to note. First, multiple anomalies could occur simultaneously. For instance, the anomaly-1 could occur while there is



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_j^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$

Figure 3.6: Brown triangle-yellow circle: Single computation and data vertex anomaly, Anomaly-3 Handling

anomaly 2 (See Fig. 3.7). Those multiple anomalies are handled together properly. Two exemplary multiple handlings are step-by-step demonstrated in Fig. 3.7 and Fig. 3.8. Second, we make a permutation separately within computation vertices and within data vertices after having constructed $H_{DVB_{Dr}}$ model (Fig. 3.2) in order to provide randomization for better performance of PATOH [1, 43].

3.4 Bipartite Graph Direct K-way Partitioning Model

"A bipartite graph, also called a bigraph, is a set of graph vertices decomposed into two disjoint sets such that no two graph vertices within the same set are adjacent. A bipartite graph is a special case of a k-partite graph with k=2." [73]. A mesh $M = (C_m, N_m)$ is modelled as a bipartite graph $\mathcal{BG}_{CDV_{Dr}} = (\mathcal{V}_C \cup \mathcal{V}_D, \mathcal{E}_d^c)$ to represent computation load and data requirement simultaneously (See Fig. 3.9 and Fig. 3.1a). $\mathcal{BG}_{CDV_{Dr}}$, in terms of structure, is previously used in [2, 3].

$\mathcal{V}_C$ represents the computation-vertex set (See left part of Fig. 3.9) and $\mathcal{V}_D$ represents the data-vertex set (See right part of Fig. 3.9). A computation vertex $v_i^C \in \mathcal{V}_C$ associates to computation realized for $cell_i \in C_m$. A data vertex $v_j^D \in \mathcal{V}_D$ associates to data contained in $cell_j \in C_m$. As $cell_i \in C_m$ signifies both computation and data, the $cell_i$ is represented both in $\mathcal{V}_C$, $\mathcal{V}_D$ as v_i^C, v_i^D respectively. Naturally, cardinalities of $\mathcal{V}_C$ and $\mathcal{V}_D$ are equal to each other, i.e., $|\mathcal{V}_C| = |\mathcal{V}_D|$ for mesh applications (Fig. 3.9 and Fig. 3.1a). $\mathcal{E}_d^c$ denotes the edge set. It models computation-data dependency. The task associated with $cell_i$ requires the data associated with other cells with which it shares an edge, a face or a region for its computation task in M, i.e., $\forall \text{cell}_{neig} \in N(i)$. An edge $e = (v_i^C, v_j^D) \in \mathcal{E}_d^c$ denotes v_i^C requires, for its computation, data contained in v_j^D. So, there is an edge $e = (v_i^C, v_j^D) \in \mathcal{E}_d^c$ incurred by neighborhood for $\forall \text{cell}_j \in N(i)$ and $\forall \text{v}_i^C \in \mathcal{V}_C$. For mesh applications (See Fig. 3.9 and Fig. 3.1a),

$$Adj(v_i^C) = \{v_j^D : j \in \{1, 2, ..., k\} \text{ where } N(i) = \{cell_{neig_1}, cell_{neig_2}, ..., cell_{neig_k}\}\}$$

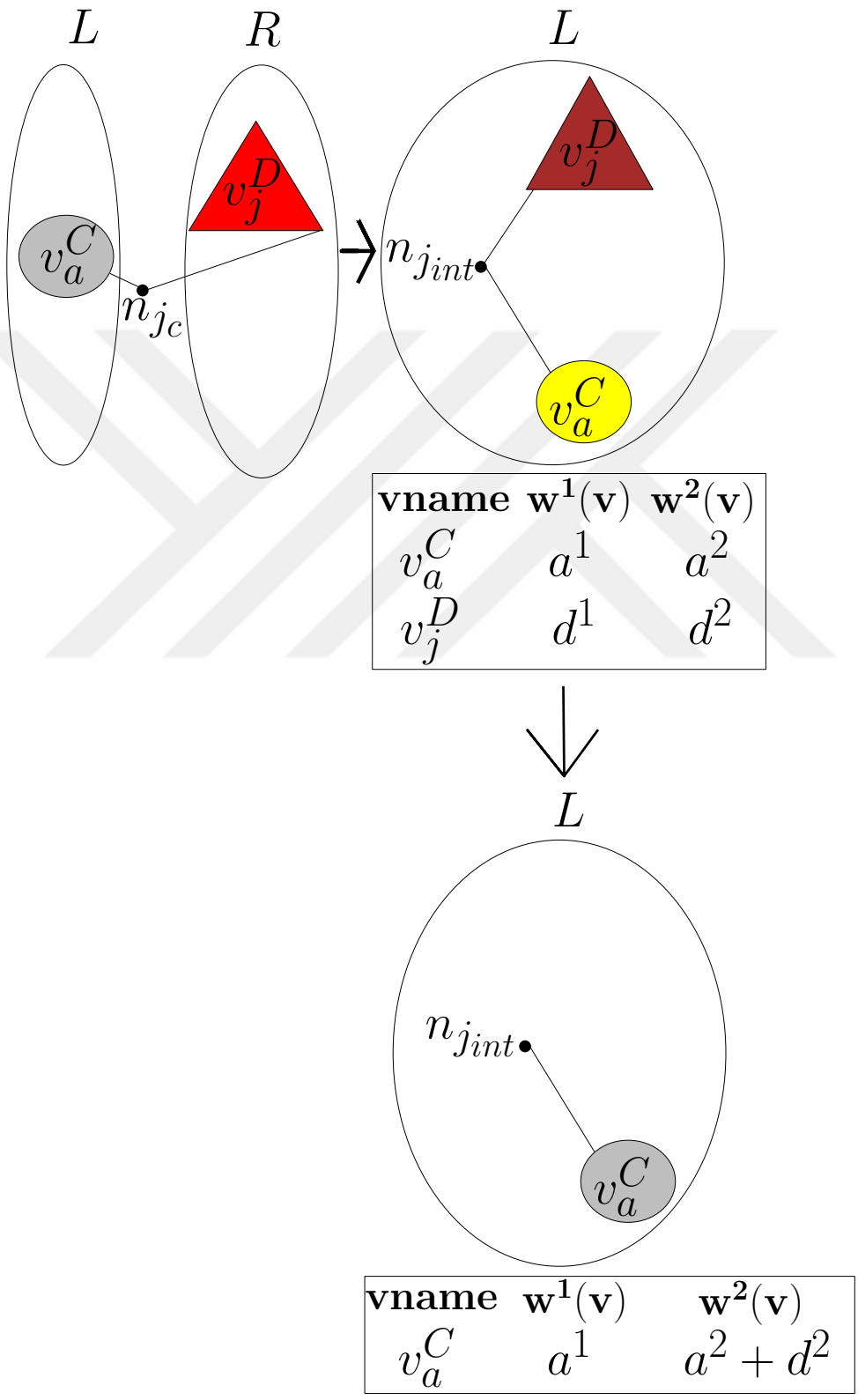
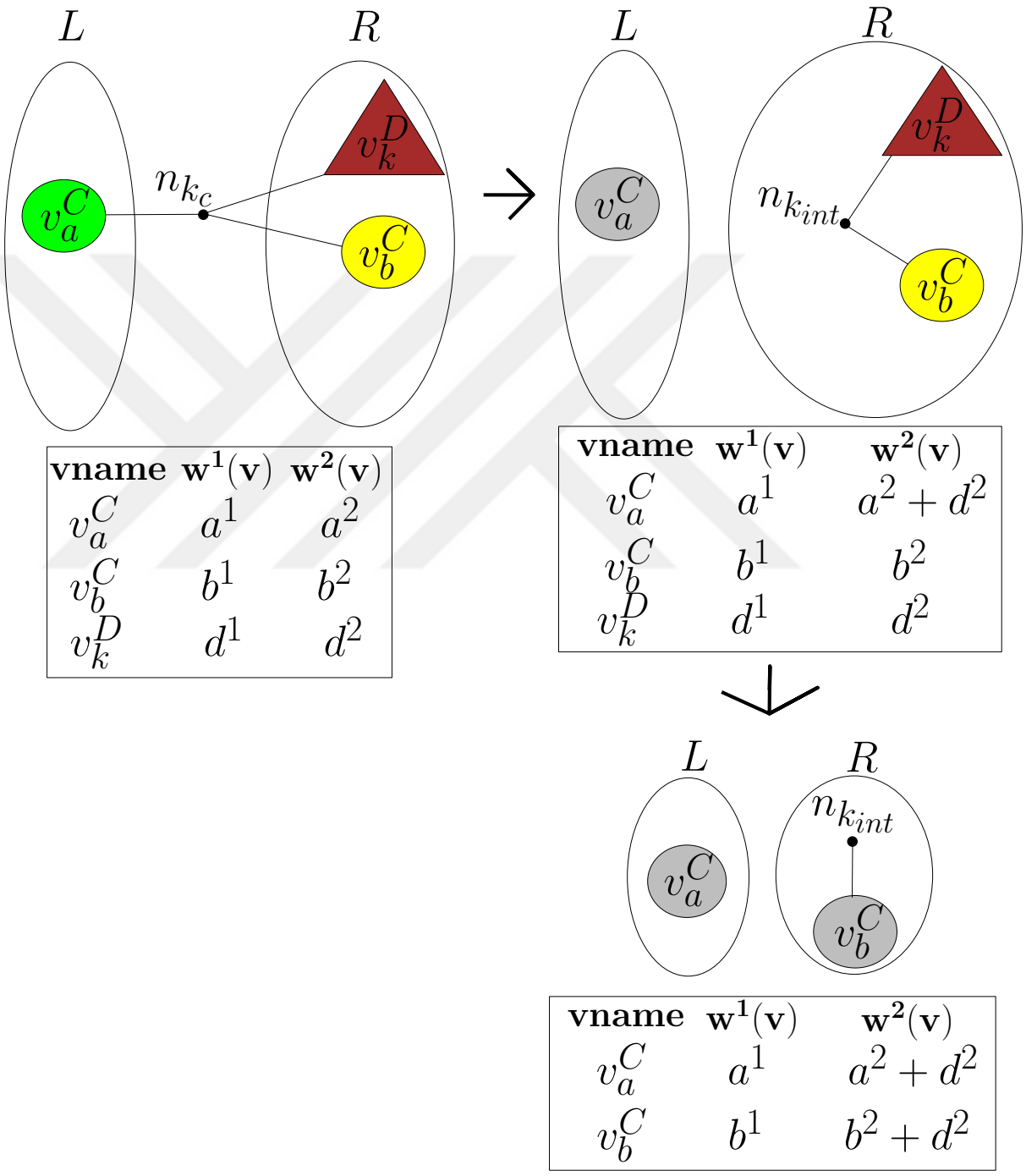


Figure 3.7: Simultaneous multiple cut-net splitting anomalies handling(Anomaly-1 and Anomaly-2).



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_k^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	b^2
v_k^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	$b^2 + d^2$

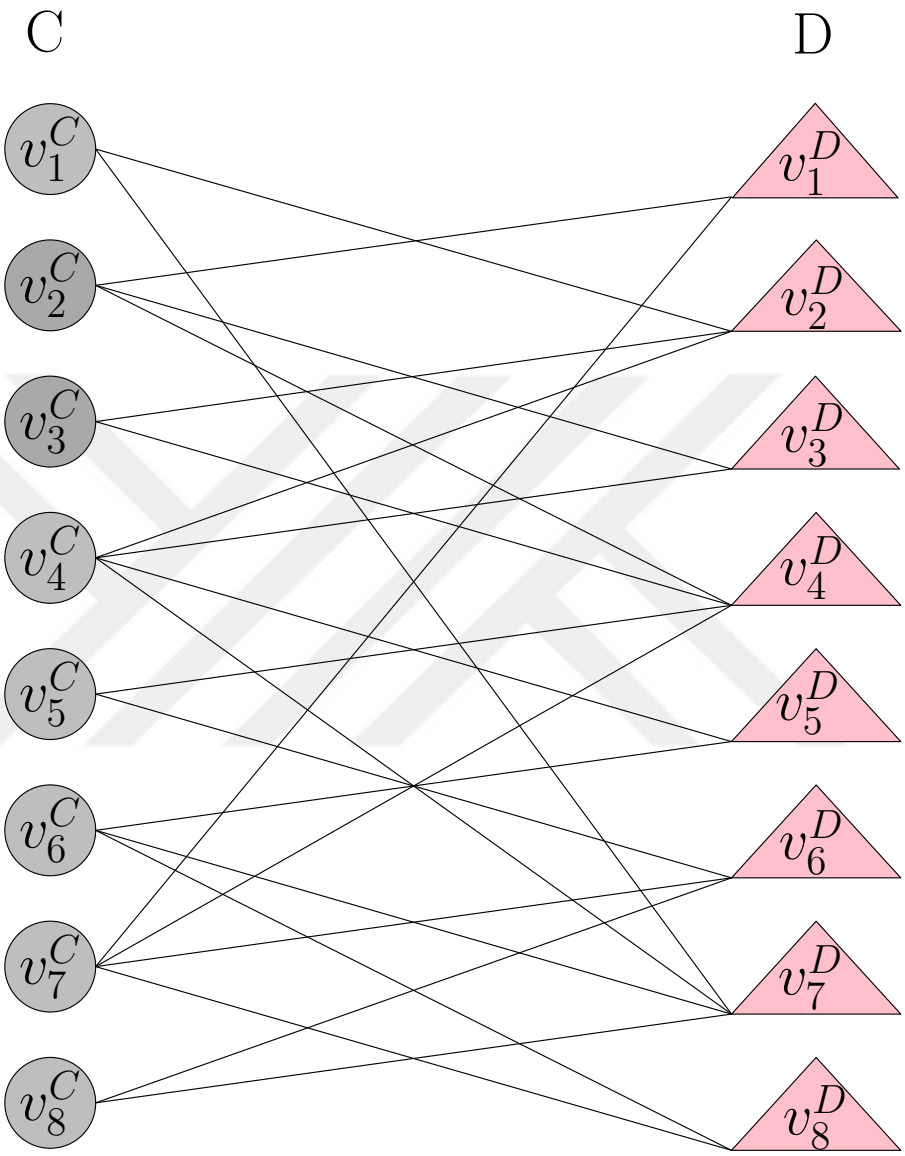
Figure 3.8: Simultaneous multiple cut-net splitting anomalies handling(Anomaly-2 and Anomaly-3).

$Adj(v_j^D) = \{v_i^C : i \in \{1, 2, ..., s\}$ where $N(j) = \{cell_{neig_1}, cell_{neig_2}, ..., cell_{neig_s}\}\}$

Each vertex of $\mathcal{BG}_{CDV_{D_r}}$ is associated with two weights to enable a two constraints formulation, i.e., the first for computation load $w^1(v_i)$ and the second for data size $w^2(v_i)$ such that $v_i \in (\mathcal{V}_C \cup \mathcal{V}_D)$. Weighting scheme used for $\mathcal{BG}_{CDV_{D_r}}$ in Fig. 3.9 is described in Section 4.2.1.

In $C = AB$ kind row-row parallel SPGEMM, vertex v_x represents the pre-multiply of row x of A with matrix B such as $c_{x,*} = a_{x,*}B$ computation [4]. So, $\forall v_x^C \in \mathcal{V}_C$ corresponds to $\forall \mathrm{v}_x$ [4]. $\forall v_j^D \in \mathcal{V}_D$ corresponds to $\forall \mathrm{row}_j \in B$ [4]. $\mathcal{E}_d^c$ includes an edge $e = (v_x^C, v_j^D)$ if vector-matrix multiply $a_{x,*}B$ needs $row_j \in B$ for computation $c_{x,*}$ [4]. The product's weighting is made as described in Section 4.2.2.

A hypergraph H may be represented by a bipartite graph $B(H)$ [74]. It's interesting to note that $\mathcal{BG}_{CDV_{D_r}}$ is a bipartite graph representation of $H_{DVB_{D_r}}$. We use direct k-way bipartite graph partitioning for the $\mathcal{BG}_{CDV_{D_r}}$. The partitioning model tries to build a balance on the both constraints of the processors. It's two-constraint direct k-way graph partitioning.



$\mathbf{i}$	$\mathbf{w^1(v_i^C)}$	$\mathbf{w^2(v_i^C)}$	$\mathbf{w^1(v_i^D)}$	$\mathbf{w^2(v_i^D)}$
1	100	0	0	10
2	144	0	0	12
3	169	0	0	13
4	196	0	0	14
5	225	0	0	15
6	256	0	0	16
7	289	0	0	17
8	324	0	0	18

Figure 3.9: The bipartite graph model of the mesh

3.5 Data-Vertex Replication Based Recursive Bipartite Graph Bipartitioning Model

$\mathcal{BG}_{CDV_{Dr}}$ is bipartitioned through the RB process to obtain sub-graphs (Fig. 3.9). Reason why RB is utilized is to improve the partitioning objective, and for same purpose replication is also used as described in Section 3.3. Main contribution of our RB framework is the split data vertex replication on bipartite graphs for the improvement.

3.5.1 Split Data Vertex Replication on Bipartite Graphs and the RB Framework

In this section, we describe split data vertex replication during RB of the bipartite graphs and the RB framework in detail. We use METIS as-is for two-way partitioning in each step of the framework. In two-way split data-vertex replicated partition $\Pi^{d_vR} = (V_L, V_R)$, state of vertex belonging is as in Section 3.3. Letters L or R are utilized to express the parts of the bipartition. Split data vertex replication is critical for the construction of two sub-bipartite graphs (BPG_L, BPG_R) in a RB step. All other bipartitioning issues are realized with respect to [32].

For the bipartite graph, split data vertex replication operation in Π^{d_vR} is illustrated in Fig. 3.10, when there is no cut-edge splitting anomaly. Cut-edge $e_c = (v_1, v_2)$ is an edge connecting two vertices v_1, v_2 locating at two different parts of a given k-way graph partitioning Π. In our context, cut-edge $e_c = (v_i^C, v_j^D)$ in Π^{d_vR} implies $v_i^C \in V_L$, $v_j^D \in V_R$ or reverse. It means that the tasks in the L and the R need v_j^D at the same time. If a v_j^D is one end of e_c in Π^{d_vR}, then v_j^D is replicated to other part, i.e, to $V_L \,|\, v_j^D \in V_R$ or to $V_R \,|\, v_j^D \in V_L$. As seen in Fig. 3.10, data vertex $v_f^D \in V_R$ is an end of three different cut-edges such as $e_{c_1} = (v_a^C, v_f^D)$, $e_{c_2} = (v_b^C, v_f^D)$, $e_{c_3} = (v_c^C, v_f^D)$. As such, v_f^D is replicated to V_L as $v_{f^{rep}}^D$ and $v_{f^{rep}}^D$ is evaluated as a normal vertex in V_L, its edges, i.e., $(v_a^C, v_{f^{rep}}^D)$,

$(v_b^C, v_{f_{rep}}^D)$, $(v_c^C, v_{f^{rep}}^D)$, are added to $\mathcal{E}_\mathcal{L}$ (See Fig. 3.10). The split data vertex replication is applied for every data vertex v_j^D which is an end of an cut-edge e_c in $\Pi^{d_v R}$ in each RB step as described, if there is no anomaly. As in Section 3.3, computation load and data size are stored precisely in computation and data weights (i.e., $w^1(v_i)$, $w^2(v_i)$) in each RB step because of the replication. RB for each sub-graph continues until determined K-way partition is obtained. As aforementioned, there are cut-edge splitting anomalies. They are handled to reduce the cutsize, that is to reduce the replication and so total data requirement. Each handling is described in following.

3.5.1.1 Cut-edge Splitting Anomalies in the Split Data Vertex Replication

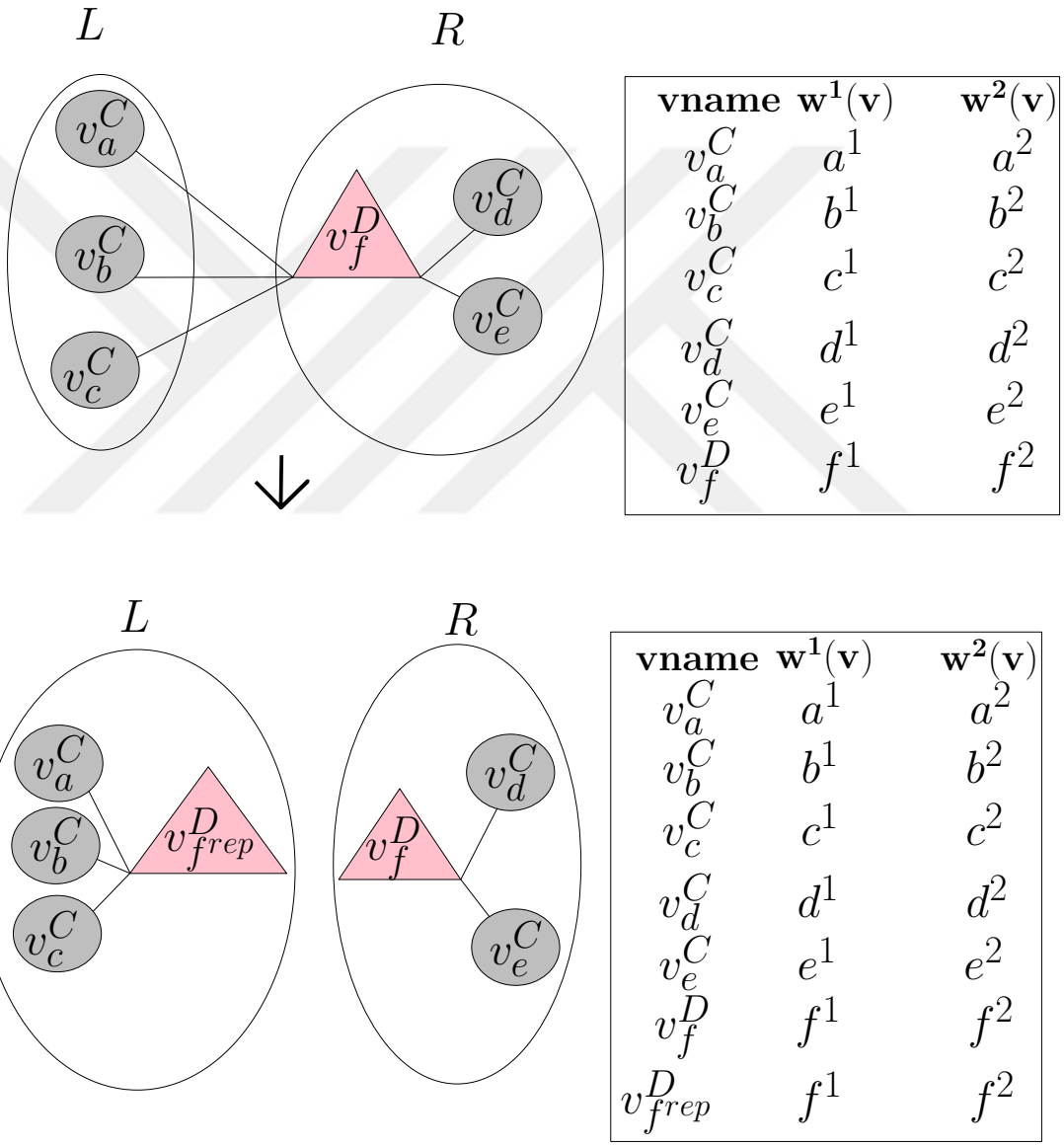
There are three (3) general cut-edge splitting anomalies. Each of them with its handling is distinctly described in following three (3) sections.

3.5.1.1.1 Cut-edge Splitting Anomaly-1 (Single Data Vertex Anomaly)

This anomaly corresponds to cut-net splitting anomaly-1(Section 3.3.1.1.1). When the data vertex v_j^D is on one part such as R and computation vertices in its adjaceny list, i.e., $v_i^C \in Adj(v_j^D)$, are on L (See Fig. 3.11). Then, there is no necessity for the replication due to absence of computation vertex v_i^C requiring v_j^D on L in $\Pi^{d_v R}$. As seen in Fig. 3.11, v_j^D's part is changed to L (similar to Alg. 3, Line 5-7). Each cut-edge e_c connecting v_j^D becomes uncut(or internal). The part change has a bit negative effect on proposed cutsize, imbalance. So, cutsize is computed after the change.

3.5.1.1.2 Cut-edge Handling Anomaly-2 (Single Computation Vertex Anomaly)

The anomaly corresponds to cut-net splitting anomaly-2 (Section-3.3.1.1.2). When data vertex v_j^D is an end of only one cut-edge $e_c = (v_a^C, v_j^D)$ in $\Pi^{d_v R}$, i.e., $(v_j^D \cup Adj(v_j^D) \setminus v_a^C) \in \mathcal{V}_R$, $v_a^C \in \mathcal{V}_L$, data weight of the data vertex v_j^D



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_d^C	d^1	d^2
v_e^C	e^1	e^2
v_f^D	f^1	f^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_d^C	d^1	d^2
v_e^C	e^1	e^2
v_f^D	f^1	f^2
v_{frep}^D	f^1	f^2

Figure 3.10: Split Data Vertex Replication in RB of BPG_{DVB}

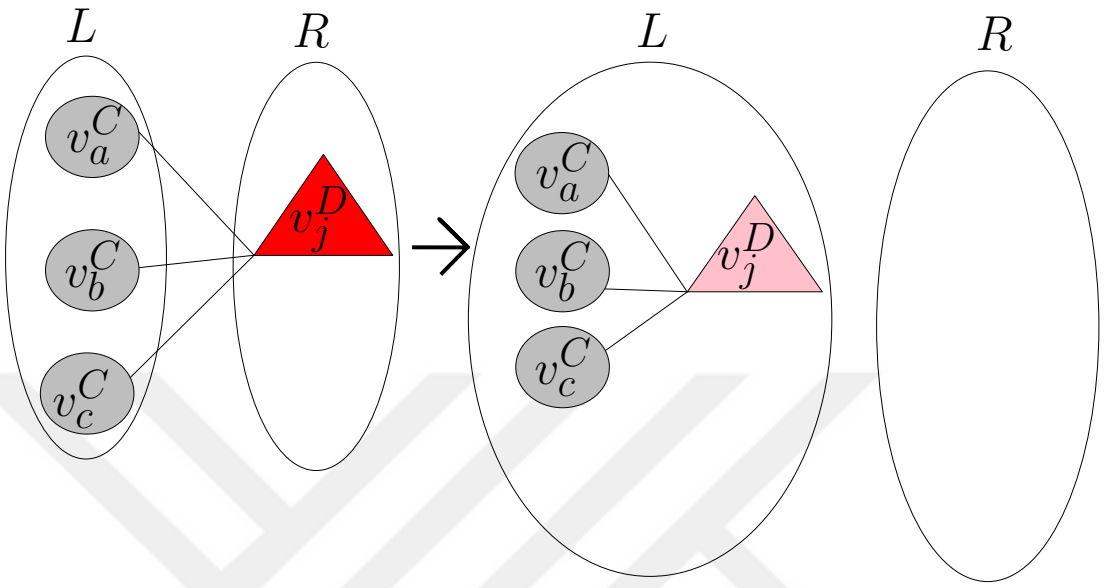
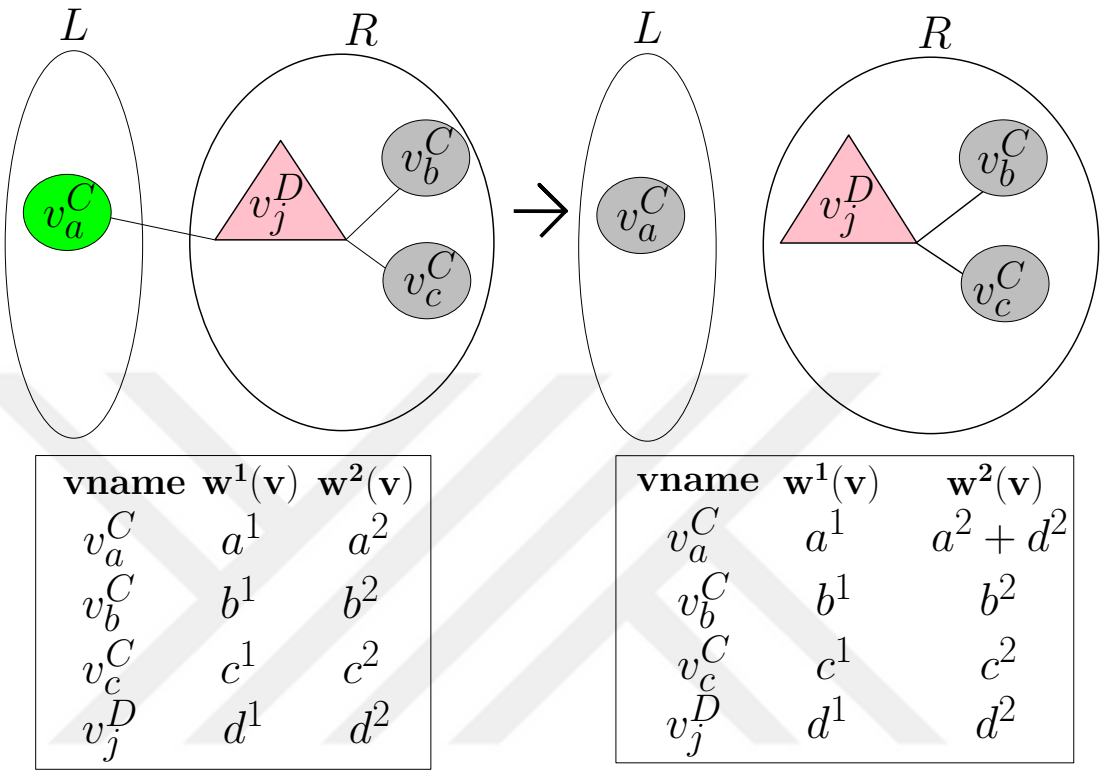


Figure 3.11: Red triangle: Single data vertex anomaly, Cut-edge Splitting Anomaly-1 Handling

is compressed onto data weight of the computation vertex v_a^C by addition of $w^2(v_j^D) = d^2$ to the $w^2(v_a^C) = a^2$ (See Fig. 3.12). So, $w^2(v_a^C) = w^2(v_a^C) + w^2(v_j^D)$, and finally $w^2(v_a^C) = a^2 + d^2$ as seen in Fig. 3.12. We prefer the compression instead of the replication. Because, internal edge $e_{int} = (v_a^C, v_{j^{rep}}^D)$ is created in the replication, $v_{j^{rep}}^D$ will have only v_a^C in its adjacency list, and e_{int} could easily be cut in further bipartitions of the RB. The possible cut-edge formation e_{int_c} could increase cutsize needlessly. At the end of the anomaly handling, total weights are preserved for L and R.

3.5.1.1.3 Cut-edge Handling Anomaly-3 (Single Computation and Single Data Vertex Anomaly)

The anomaly corresponds to the cut-net splitting anomaly-3 (Section 3.3.1.1.3). In the RB framework, v_j^D belongs to L or R (or to both in case of the replication). Data vertex v_j^D could be an end of only one internal edge $e_{int} = (v_a^C, v_j^D) \in \mathcal{E}_R$ in $\Pi^{d_v R}$, i.e., $(v_j^D \cup v_a^C) \in \mathcal{V}_R$, $(Adj(v_j^D) \setminus v_a^C) \in \mathcal{V}_L$ or reverse (See Fig. 3.13). In this case, compression operation onto the $w^2(v_a^C) = a^2$ is applied as explained in Section 3.3.1.1.3. As seen in Fig. 3.13, post-compression state becomes $w^2(v_a^C) = a^2 + d^2$. $e_{int} \in \mathcal{E}_R$ or $e_{int} \in \mathcal{E}_L$ is deleted from $\mathcal{E}_R$ or $\mathcal{E}_L$ and v_j^D is deleted



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_j^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	b^2
v_c^C	c^1	c^2
v_j^D	d^1	d^2

Figure 3.12: Green circle: Single computation vertex anomaly, Cut-edge splitting Anomaly-2 Handling

from $\mathcal{V}_R$ or $\mathcal{V}_L$ after the replication if the replication is required (See Fig. 3.13). Total weights are preserved as usual.

Finally, we should note two points. First, we didn't give pseudocode for handling the cut-edge splitting anomalies in Section 3.5.1.1. Since, each anomaly is so similar to each respective cut-net splitting anomaly described in Section 3.3.1.1. Therefore, Alg. 3, Alg. 4 and Alg. 5 are applied to the respective cut-edge splitting anomaly with minor adaptations. Second, there could be multiple cut-edge splitting anomalies. For example, while there is an cut-edge splitting anomaly-2, there could be cut-edge splitting anomaly-3 at the same time (See Fig. 3.15). Those kinds of multiple anomalies are handled together properly. Two multiple anomaly handlings are illustrated step by step in Fig. 3.14 and Fig. 3.15.

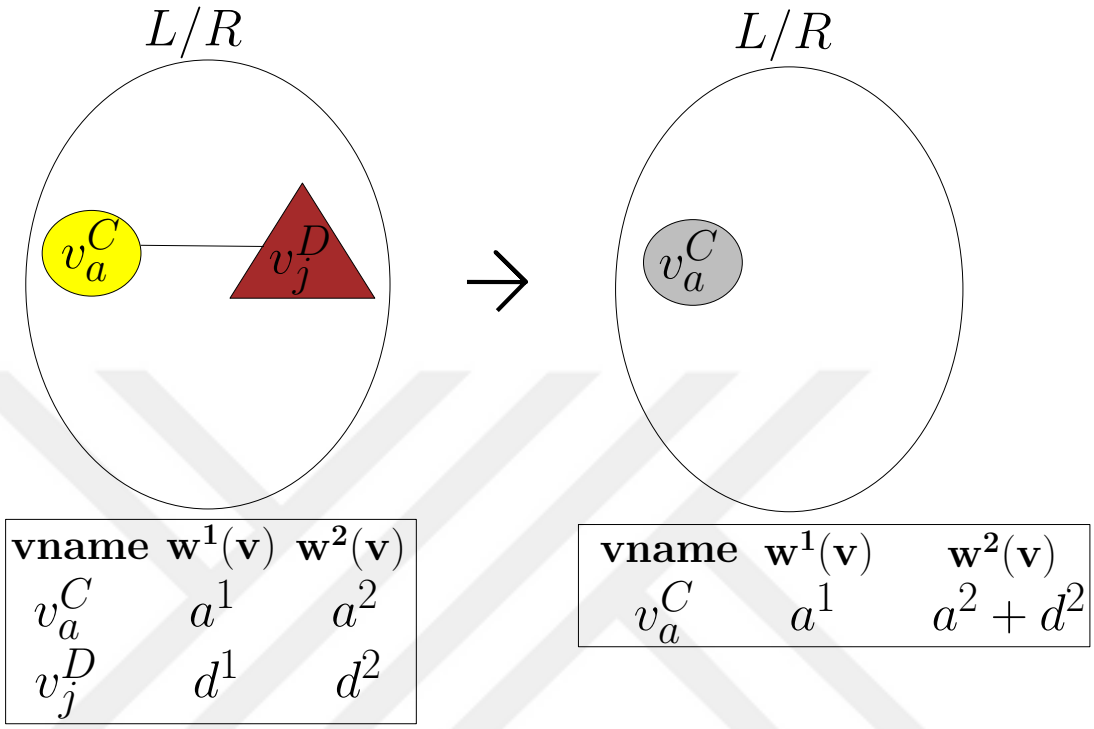


Figure 3.13: Brown triangle-yellow circle: Single computation and data vertex anomaly, Cut-edge splitting Anomaly-3 Handling

3.6 Inverse Data Weight Distribution Based Direct K-way Hypergraph Partitioning Model

Main contribution of this model is inverse data weight distribution. As seen from Fig. 3.17 and Fig. 3.1b, construction of $H_{IDWD_{Dr}} = (V_{IDWD_{Dr}}, N_{IDWD_{Dr}})$ from the examplary mesh in Fig. 3.1a is structurally same as the hypergraph H_{Sc}. Structure implies formation of vertex set and net construction. Nevertheless, there are two constraints, i.e., first for computation and second for data, in $H_{IDWD_{Dr}}$ unlike H_{Sc} (See Fig. 3.17 and Fig. 3.1b). Main difference comes from data constraint $w^2(v_i)$ acquired by inverse data weight distribution.

The distribution is described in Algorithm 6. Nets represent data requirement

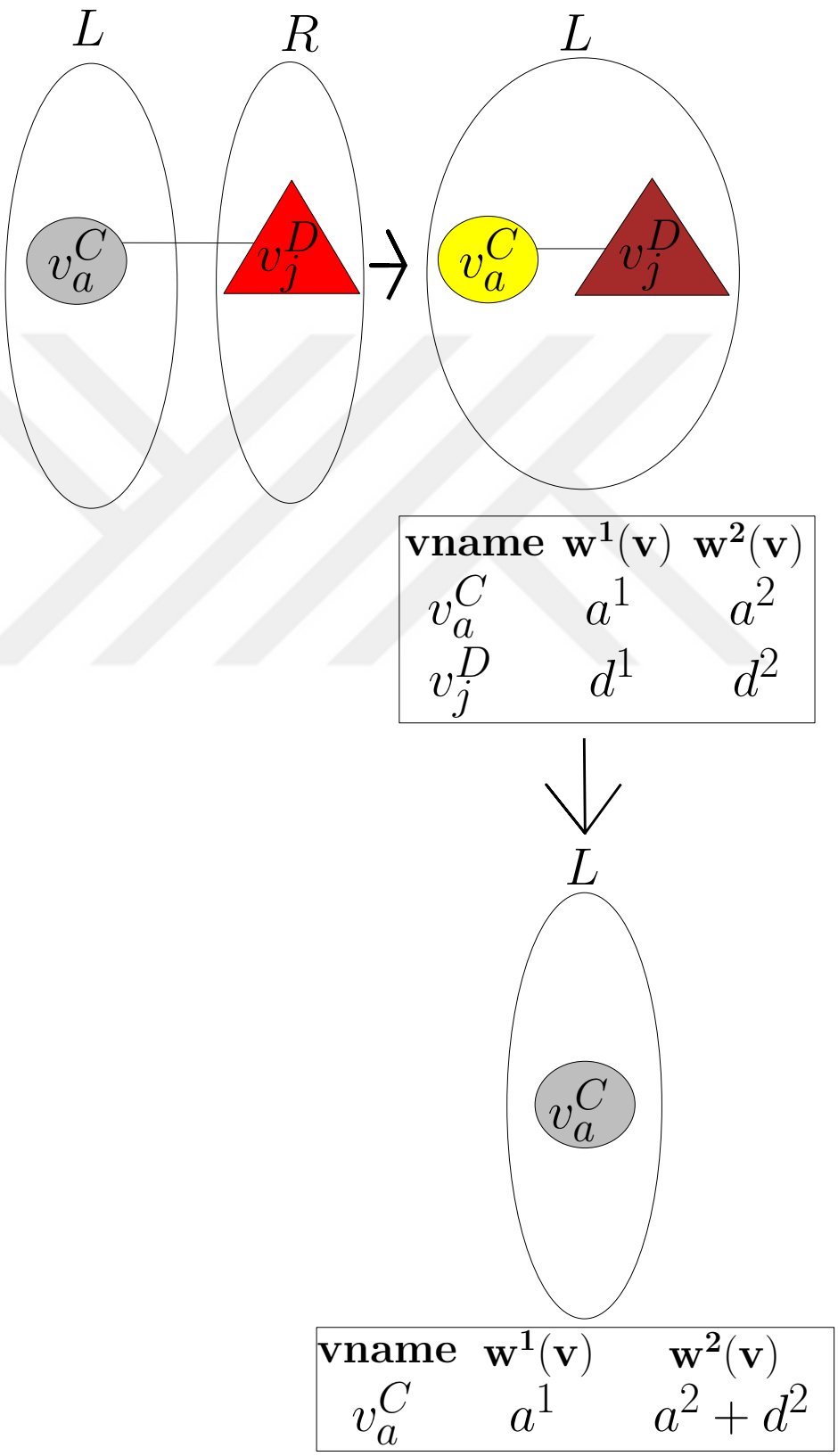
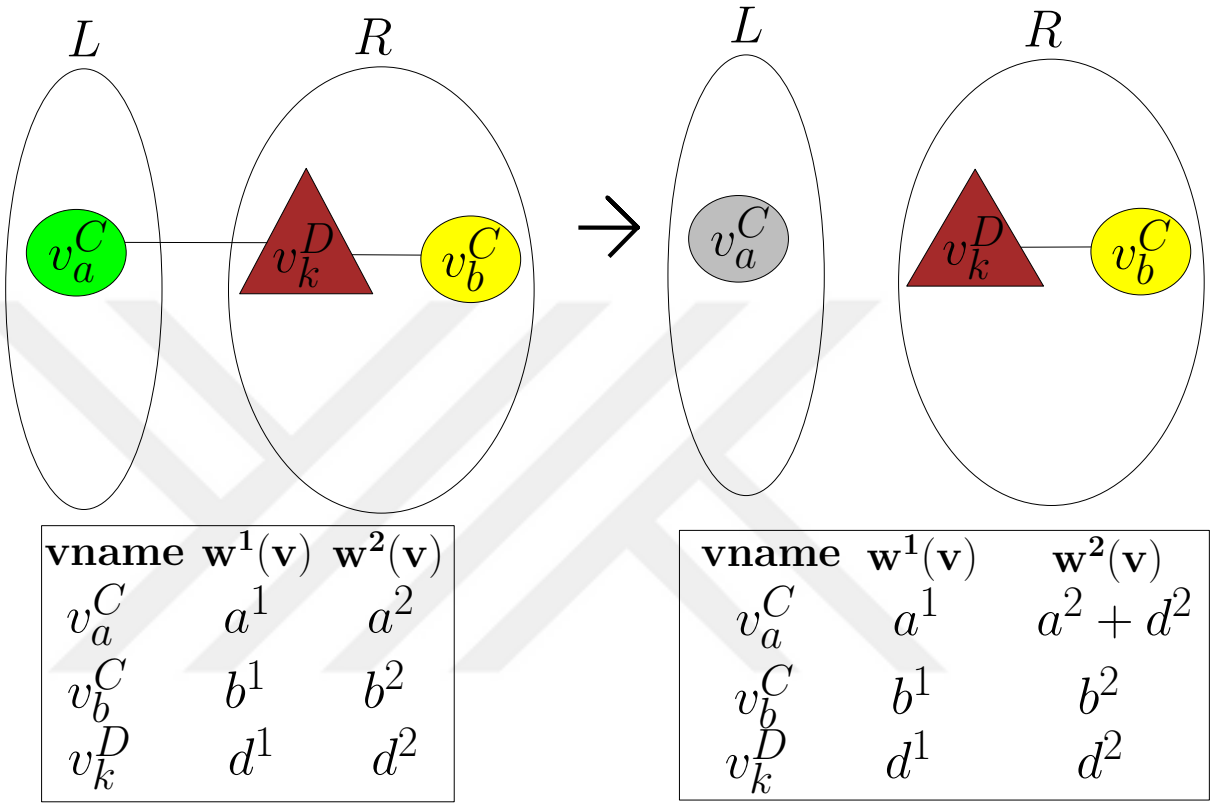
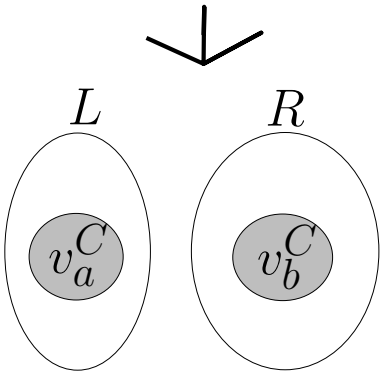


Figure 3.14: Simultaneous multiple cut-edge splitting anomalies handling(Anomaly-1 and Anomaly-2).



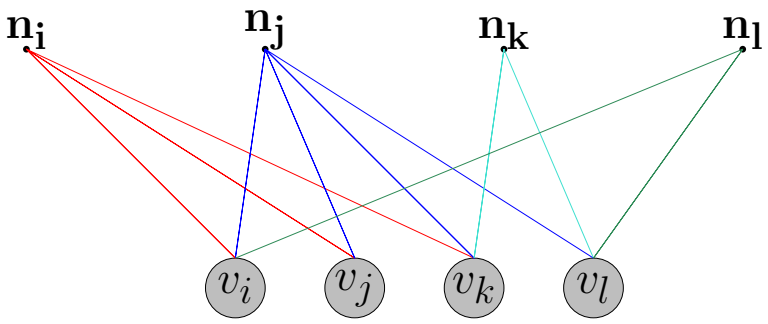
vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	a^2
v_b^C	b^1	b^2
v_k^D	d^1	d^2

vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	b^2
v_k^D	d^1	d^2



vname	$\mathbf{w^1(v)}$	$\mathbf{w^2(v)}$
v_a^C	a^1	$a^2 + d^2$
v_b^C	b^1	$b^2 + d^2$

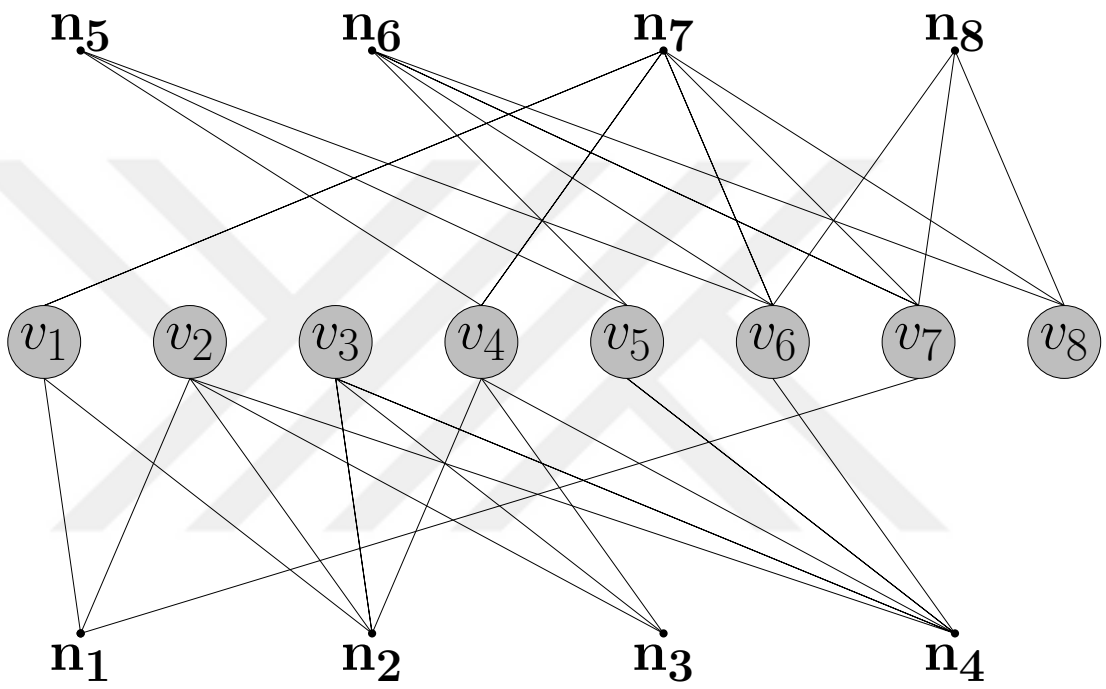
Figure 3.15: Simultaneous multiple cut-edge splitting anomalies handling(Anomaly-2 and Anomaly-3).



$$w^2(v_i) = c_(n_i)/|Pins(n_i)| + c_(n_j)/|Pins(n_j)| + c_(n_l)/|Pins(n_l)|, w^2(v_i) = round(w^2(v_i) * 1000)$$
$$w^2(v_j) = c_(n_i)/|Pins(n_i)| + c_(n_j)/|Pins(n_j)| \quad , w^2(v_j) = round(w^2(v_j) * 1000)$$
$$w^2(v_k) = c_(n_i)/|Pins(n_i)| + c_(n_j)/|Pins(n_j)| + c_(n_k)/|Pins(n_k)|, w^2(v_k) = round(w^2(v_k) * 1000)$$
$$w^2(v_l) = c_(n_j)/|Pins(n_j)| + c_(n_k)/|Pins(n_k)| + c_(n_l)/|Pins(n_l)|, w^2(v_l) = round(w^2(v_l) * 1000)$$

Figure 3.16: Inverse data weight distribution realization

(or load) in $H_{IDWD_{Dr}}$ as in Section 3.1. Each net weight is distributed to its vertices (See Fig. 3.16). Each net n_j contributes to data weight of its connected vertices (each v_x) equally by $c(n_j)/s_j$ where $v_x \in Pins(n_j)$ and $s_j = |Pins(n_j)|$ (Algorithm 6, Line 1-6). Additionally, $nnz(b_{*,x})$ is added to each vertex v_x's data weight $w^2(v_x)$ right after the distribution for the $C = AB$ SPGEMM operation as described in Section-4.2.2. Then, we scale each data weight $w^2(v_x)$ by 1000 (Algorithm 6, Line 8-10). One last step is to round each scaled data weight $w^2(v_x)$. This scaling is performed since PATOH utilizes only integer weights. The whole distribution procedure's exemplary representation is given in Fig. 3.16. Mesh weighting scheme (Section 4.2.1) is used in the representation. As seen in Fig. 3.16, $Nets(v_i) = \{n_i, n_j, n_k\}$. v_i's data weight $w^2(v_i)$ takes contribution from each respective net by c_y/s_y such that $y \in \{i, j, k\}$ and $s_y = |Pins(n_y)|$. Same distribution logic is applied for each vertex in Fig. 3.16. Each data weight $w^2(v_x)$ is resulting value of the distribution algorithm in Fig. 3.17. Finally, we apply direct k-way hypergraph partitioning to the model in Fig. 3.17.



$\mathbf{i}$	$\mathbf{w^1(v_i)}$	$\mathbf{w^2(v_i)}$	$\mathbf{c(n_i)}$
1	100	8250	10
2	144	15000	12
3	169	7500	13
4	196	22250	14
5	225	8830	15
6	256	24250	16
7	289	19330	17
8	324	9580	18

Figure 3.17: Inverse data weight distribution based hypergraph representation of the mesh

Algorithm 6 Inverse Data Weight Distribution Handling

Input : **Hypergraph** H_{IDWD}
Output: **Vertex weight vectors**

```
1: for each net n_j ∈ N_IDWD do
2:     dwContr = c(n_j) / |Pins(j)|;
3:     for each v_i ∈ Pins(n_j) do
4:         w^2(v_i) += dwContr;
5:     end for
6: end for
7: // Scale weights by multiplying 1000
8: for each data weight w^2(v_i) do
9:     w^2(v_i) *= 1000
10: end for
```

3.7 Inverse Data Weight Distribution Based Recursive Hypergraph Bipartitioning Model

As the name suggests, this model is recursive bipartitioning (RB) framework implementation of inverse data weight distribution. The model $H_{IDWD_{D_r}}$ in Fig. 3.17 is obtained at the beginning. Then, we recursively bipartition the initial hypergraph $H_{IDWD_{D_r}}$ (Fig. 3.17) applying inverse data weight distribution in each RB step. Via utilization of RB, our objective is to better encapsulating processors' data size so that maintaining balance of data sizes of processors while minimizing total amount of data replication better encodes minimizing max data size of processors.

3.7.1 Cut-Net Splitting in the Model and RB Framework

We use PATOH as-is for two-way partitioning to achieve bipartition vector in each step of RB. Then, cut-net splitting is handled as in [1] for formation of two new sub-hypergraphs. When we built the sub-hypergraphs by bisection, the distribution (Alg. 6) is distinctly applied for each sub-hypergraph. RB for each sub-hypergraph continue up until aimed K-way partition is obtained.

Chapter 4

Experimental Results

In this chapter, we present and discuss our conducted experiments to evaluate the performance of the proposed graph/hypergraph models for simultaneous computation and data load balancing.

4.1 Experimental Dataset

We assessed the performance of the proposed graph/hypergraph models for distributed mesh simulations on a wide range of sparse matrices of Tim Davis, University of Florida (UFL) sparse matrix collection [75]. From this dataset, we choose the matrices which have either 2D or 3D coordinate values (The sparsity patterns of these matrices are considered as representing the neighborhood structure of the mesh represented in FEM or VEM applications) in order to generate computation and data weights with "Particle-In-Cell"-like simulations vertex weight generation algorithm in [50] as described in Section 4.2.1. We didn't include matrices with less than averagely 100 rows locating in a part of given K-way vertex partition $\Pi(G)$ or $\Pi(H)$. That is, we have matrices who have at least 6400 rows for 64-way graph/hypergraph vertex partition and 12800 rows for 128-way vertex partition so on and so forth, to obtain purposeful partitioning results for

effective parallelization. Therefore, we have different number of instances for different K-way partitions (See Table 4.1, 4.2, 4.3, 4.4, 4.5, 4.6). Also, we excluded some instances whose partitioning experiments dures so much for big K values (i.e., K = 512, 1024) due to sequential partitioning environment and hardware capabilities.

The C = AB kind SPGEMM instances, we tested the models, as in algebraic multigrid solver are obtained by using the tool in [76]. On those SPGEMM instances, generated interpolation operators by the tool, i.e., ".P" suffix denotes operator matrix, are used in usual as B matrix. In addition to those, there are several instances from first C = AB kind SPGEMM operation of the Galerkin product RAP, i.e., C= RA, and certain instances including SPGEMM from recommendation systems field [4] and thermomech_dK-thermomech_dM instance [4] are taken into experiments. Same exclusion principle related with average number of rows in a part of given K-way vertex partition $\Pi(G)$ or $\Pi(H)$ have been also applied for the SPGEMM dataset.

4.2 Experimental Weighting Scheme

Computation and data weighting schemes for two applications which are used in the models are described.

4.2.1 Computation and Data Weighting Scheme Used for Modelling the Distributed Mesh Computations

First, one should recognize that various weighting schemes arise from application nature of mesh computations [2, 3, 50, 77]. It seems problematic that no weight distribution exist along with meshes in [75]. Weight distributions may change algorithm performance [50]. Putting together meshes with their multi-constraint weight distributions related with their application nature in a database is reported to be valuable [50].

In the thesis, we use weighting scheme produced by a method (or algorithm) for generating realistic weight distributions corresponding to "Particles-in-Cells"-like simulations [50]. Relation between computation load and memory footprint in [2] is used identically in [50]. "Relationship between computational cost and memory weight is arbitrary" [2]. It means the relation may be quadratic, cubic etc. Nevertheless, the computational cost of a mesh element is reported to be equal to the square of its memory size for the weighting [2]. Reasoning behind the said relation in "Particles-in-Cells"-like simulations is described as in following: Computations are related to the the number of particles located in a mesh cell and whilst computation of one cell generally increases with the square of the number of particles in this cell, the amount of memory needed for holding a cell is linear with the number of particles located in this cell [50]. Including the said weighting, the method for creating memory and computation weight distributions of "Particle-In-Cell"-like simulations is proposed [50] (Algorithm 51/Generation of the Vertex Weights). Very similar algorithm is proposed in [51]. As said, we used the algorithm [50].

We faced with a problem of finding a solution for the weight generation in certain number of sparse matrices from [75] due to topology of the matrices. Hence, interval for finding radius, i.e., radius of each peak, in the algorithm [50] was changed for expanding the solution space as follows:

The interval was: $L/8 \leq \text{radius} \leq L/4$ [50],
It became: $L/64 \leq \text{radius} \leq L/32$.

where L is length of the mesh [50]. We defined L as arithmetical average of differences of each mesh dimension's minimum and maximum values in cartesian plane. We utilized 2D and 3D data sets from Tim Davis, University of Florida Sparse Matrix Collection [75]. Therefore, L was computed as the average of at most respective three differences.

In Section 3.1, we only used generated computation weights cw_i for weighting vertex $v_i \in \mathcal{V}$ and generated data weight dw_j is utilized for weighting net $n_j \in \mathcal{N}$ (See Fig. 3.1b). Naturally, computation weight $w^1(v_i)$ is square of net weight

$c_(n_i)$ as seen in Fig. 3.1b. Formally,

$$\forall v_i \in \mathcal{V}\,,\, w^1(v_i) \;\; = cw_i \tag{4.1}$$

$$\forall n_j \in \mathcal{N}\,,\, c(n_j) \;\; = dw_j \tag{4.2}$$

Difference of Sections 3.2 and 3.3 from Section 3.1 is that a vertex is additionally weighted with data weight $w^2(v_i) \,|\, v_i \in \mathcal{V}$ (See Fig. 3.9 and Fig. 3.1). Also, we differentiate vertex type, i.e., computation vertex v_i^C and data vertex v_j^D (See Fig. 3.9). v_i^C has non-zero computation weight $w^1(v_i^C)$ equals to cw_i, zero data weight $w^2(v_i^C)$. On the contrary, v_j^D has non-zero data weight $w^2(v_j^D)$ equals to dw_j, zero computation weight $w^1(v_j^D)$. dw_j is used to assign net cost $c(n_j)$. So, computation weight $w^1(v_i^C)$ is square of net cost $c(n_i)$ and data weight $w^2(v_i^D)$ as seen in Fig. 3.2. Formally,

$$\forall v_i^C \in \mathcal{V}^{\mathcal{C}},\, w^1(v_i^C) = cw_i \tag{4.3}$$

$$w^2(v_i^C) = 0 \tag{4.4}$$

$$\forall v_j^D \in \mathcal{V}^{\mathcal{D}},\, w^1(v_j^D) = 0 \tag{4.5}$$

$$w^1(v_j^D) = dw_j \tag{4.6}$$

$$\forall n_j \in \mathcal{N},\, c(n_j) = dw_j \tag{4.7}$$

In Section 3.4 and Section 3.5, the right above weighting scheme is applied almost in same way (See Fig. 3.9 and Fig. 3.2). But, there is no edge cost corresponding to net cost (See Fig. 3.9).

In Section 3.6 and Section 3.7, vertex $v_i \in \mathcal{V}$ has a data weight $w^2(v_i)$ apart from Section 3.1 (See Fig. 3.17 and Fig. 3.1). Computation weight $w^1(v_i)$ and net weight $c(n_j)$ is obtained in same way with Section 3.1 (See Fig. 3.17 and Fig. 3.1). Data weight $w^2(v_i)$ is assigned by inverse data weight(or net cost) distribution heuristic described in Section 3.6 (See Fig. 3.16 and Fig. 3.17).

4.2.2 Computation and Data Weighting Scheme Used for Modelling the Row-row parallel C=AB kind SPGEMM Computations

Weighting is done with respect to structure of A, B and C matrices and nature of the SPGEMM [4]. Computation weight $w^1(v_x)$ and net cost $c(n_j)$ is defined as follows [4]:

$$w^1(v_x) = \sum_{i \in cols(a_{x,*})} nnz(b_{i,*}) \tag{4.8}$$

$$c(n_j) = nnz(b_{j,*}) \tag{4.9}$$

where, v_x represents each the pre-multiply of row x of A with matrix B such as $c_{x,*} = a_{x,*}B$ computation, a net n_j represents each row j of B.

In contrast to distributed mesh computations (Section-4.2.1),there is an automatic data weight $w^2(v_x)$ of a vertex $v_x \in \mathcal{V}$ representing a computation in the SPGEMM (when, a model is two-constraint, e.g., Section 3.6.):

$$w^2(v_x) = nnz(b_{*,x}) \tag{4.10}$$

In Section 3.1, computation and net weighting is done as described in Eq. (4.8) and Eq. (4.9). Formally,

$$\forall v_i \in \mathcal{V}\,,\; w^1(v_i) = \sum_{j \in cols(a_{i,*})} nnz(b_{j,*}) \tag{4.11}$$

$$\forall n_j \in \mathcal{N}\,,\; c(n_j) = nnz(b_{j,*}) \tag{4.12}$$

In Section 3.2 and Section 3.3, computation vertex v_i^C has computation weight $w^1(v_i^C)$ given in Eq. (4.8), data weight $w^2(v_i^C)$ given in Eq. (4.10). Data vertex

v_j^D has zero computation weight $w^1(v_j^D)$, positive data weight $w^2(v_j^D)$ equals to cost $c(n_j)$ of net n_j (4.9). Formally,

$$\forall v_i^C \in \mathcal{V}^{\mathcal{C}},\ w^1(v_i^C) = \sum_{j \in cols(a_{i,*})} nnz(b_{j,*}) \tag{4.13}$$

$$w^2(v_i^C) = nnz(b_{*,i}) \tag{4.14}$$

$$\forall v_j^D \in \mathcal{V}^{\mathcal{D}},\ w^1(v_j^D) = 0 \tag{4.15}$$

$$w^2(v_j^D) = c(n_j) \tag{4.16}$$

$$\forall n_j \in \mathcal{N},\ c(n_j) = nnz(b_{j,*}) \tag{4.17}$$

In Section 3.4 and Section 3.5, computation and data weighting are same as the weighting scheme specified in Eq. (4.13), Eq. (4.14), Eq. (4.15), Eq. (4.17). As above (Section 4.2.1), there is no edge cost which could correspond to net cost in the models (Section 3.4, Section 3.5).

In Section 3.6 and Section 3.7, vertex $v_i \in \mathcal{V}$ has positive computation and data weights, i.e., $w^1(v_i) > 0, w^2(v_i) > 0$. Computation weight $w^1(v_i)$ and net weight $c(n_j)$ are determined as in Section 3.1. Data weight $w^2(v_i)$ is obtained in three steps. First, we apply inverse data weight(or net cost) distribution Algorithm 6, Line 1-6 (See Fig. 3.16). Second, we add $nnz(b_{*,i})$ given in Eq. (4.10) before scaling in Alg. 6 (Line 8-10). Third, we scale the weights by 1000 as described in Alg. 6 (Line 8-10).

4.3 Experimental Setup

For partitioning hypergraphs as described in the hypergraph partitioning models (Section 3.1, 3.2, 3.3, 3.6, 3.7), we employed state-of-the-art serial hypergraph partitioner tool PATOH [1, 43]. For bipartite graph partitioning as described in Section 3.4, 3.4, we utilized state-of-the-art serial graph partitioner tool METIS

[31, 32]. In literature, METIS is also known as a widely used tool for partitioning meshes, it has an interface to partition the mesh data directly [32]. The RB frameworks (Section 3.3, 3.5, 3.7) have been built via using the tools as-are for two-way partitioning. Both tools use multi-level partitioning schemes (i.e., Coarsening, initial partitioning, refinement phases in each level) for obtaining a bipartition.

We conducted experiments presented in this thesis with K $\in$ {64, 128, 256, 512, 1024} where K signifies number of processors. In other words, the test instances are partitioned for each K value. K denotes number of distinct processors in a distributed parallel platform. As both PATOH and METIS utilize randomized algorithms, we made five (5) consecutive runs, acquired 5 different partition results for each test instance. Then, we took geometric average of 5 partitions for each performance metric (Section 4.4.1.1, 4.4.1.2, 4.4.1.3), to use the average as representative result. In all test instances, we used maximum allowable imbalance ratio $\epsilon = 0.05$ for both computation and data weights, i.e., $\epsilon^1 = 0.05$ and $\epsilon^2 = 0.05$. All experiments have been conducted by use of random seed in the respective tools.

We employed PATOH [1, 43] via using default parameters for all models except the partitioning model described in Section 3.3. In this model (Section 3.3), we preferred cell visit order for coarsening algorithms as continuous/sequential. It signifies that there is increasing vertex ID order in visits of coarsening phase (i.e., $crs\_VisitOrder = 0$), i.e., PATOH visits first computation vertices $\forall v_i^C \in \mathcal{V}^{\mathcal{C}}$ and then data vertices $\forall v_j^D \in \mathcal{V}^{\mathcal{D}}$ in increasing manner [43]. Additionally, we make a permutation separately within computation vertices $\forall v_i^C \in \mathcal{V}^{\mathcal{C}}$ and within data vertices $\forall v_j^D \in \mathcal{V}^{\mathcal{D}}$ after having constructed the model (Fig. 3.2) in order to provide randomization in vertex visit order for better performance of PATOH on the model. We used the METIS [31, 32] with default parameters, partitioning objective of totalv (which refers to total communication volume) is selected for better encapsulating total communication volume. Please note that minimizing total communication volume corresponds to minimizing total amount of data replication since the size of the vertex is set to the size of data represented by the vertex.

4.4 Performance Evaluation of the Proposed Models

In this section, we describe performance metrics that we use for the evaluation of load balance quality and their calculations (Alg. 7) in detail.

4.4.1 Performance Metrics

In a given graph/hypergraph K-way vertex partition vector, we know which vertex is in which part, i.e., $v_i \in P_{belong}$ where $belong \in \{1, 2, ..., K\}$ and that each vertex is connected by which nets, i.e, $Nets(v_i)$ or nets connected which vertices, i.e., $Pins(n_j)$. By these informations, we compute three performance metrics such as maximum computation load proportion $W^1_{maxLoadProp}$ (Section 4.4.1.1), maximum data load proportion $W^2_{maxLoadProp}$ (Section 4.4.1.2), average data load proportion$W^2_{avgLoadProp}$ (Section 4.4.1.3) to measure proposed computation and data load balance's quality.

4.4.1.1 Maximum Computation Load Proportion

A computation load for a part in a K-way partition Π is sum of computation weights of computation vertices, i.e., $W^1(P_k)$. It's important to note that ideal average computation load $W^1_{ideal-avg}$ is equal to average computation load W^1_{avg} due to theoretical fact that total computation load doesn't change after a partitioning where W^1_{avg} denotes the average of all K parts' computation loads. The maximum computation load is the highest computation load amongst K parts, i.e., $W^1_{max} = \max_{1 \leq k \leq K} \mathrm{W}^1(P_k)$. Maximum computation load proportion $W^1_{maxLoadProp}$ is defined as maximum computation load W^1_{max} over ideal average computation load $W^1_{ideal-avg}$.

$$W^1_{maxLoadProp} = W^1_{max} / W^1_{ideal-avg} \tag{4.18}$$

4.4.1.2 Maximum Data Load Proportion

A data load for a part in a K-way partition Π is sum of data weights of all vertices, i.e., $W^2(P_k)$. Average data load W^2_{avg} is average of all K parts' data loads for a given partition. Ideal average data load $W^2_{ideal-avg}$ is total data load which is sum of each data weight $w^2(v_i)$ where $v_i \in V$ over K before partitioning. We take into consideration that ideal average data load $W^2_{ideal-avg}$ is not equal to average data load W^2_{avg}. Because, total data load could change according to a task partitioning Π (See Section 3). Maximum data load is the highest data load amongst K parts, i.e., $W^2_{max} = \max_{1 \leq k \leq K} \mathrm{W}^2(P_k)$. Maximum data load proportion $W^2_{maxLoadProp}$ is defined as maximum data load W^2_{max} over ideal average data load $W^2_{ideal-avg}$. Because, $W^2_{ideal-avg} \neq W^2_{avg}$,

$$W^2_{maxLoadProp} = W^2_{max} / W^2_{ideal-avg} \tag{4.19}$$

4.4.1.3 Average Data Load Proportion

In order to measure the amount of data replication incurred by a partition, we define average data load proportion $W^2_{avgLoadProp}$ as average data load W^2_{avg} over ideal average data load $W^2_{ideal-avg}$.

$$W^2_{avgLoadProp} = W^2_{avg} / W^2_{ideal-avg} \tag{4.20}$$

We consider that since total computation theoretically doesn't change for any given partition. Average computation load W^1_{avg} over ideal average computation load $W^1_{ideal-avg}$ should be always one.

$$W^1_{avgLoadProp} = W^1_{avg} / W^1_{ideal-avg} = 1 \tag{4.21}$$

Hence, $W^1_{avgLoadProp}$ (Eq. 4.21) is naturally useless for sake of the balance quality measurement.

4.4.1.4 General Method for Computation of the Performance Metrics

Calculation of the metrics is presented in Algorithm 7. Algorithm 7 describes general method which is valid for all the hypergraph partitioning models (i.e., Section 3.1, 3.2, 3.3, 3.6, 3.7). When task partitioning is given as in Algorithm 7, then we would compute unique datas (i.e., nets) required for tasks (Alg. 7 - Line 7-16, unique net set $N_{UniqNetSet}$ acquisition for each part, formally $Nets(P_i)$ such that $P_i \in \Pi(H)$ and $i \in 1, 2, ..., K$). Then, we compute three metrics (Alg. 7 - Line 24-28). Adaptation of the Algorithm 7 for partitioned bipartite graphs (Section 3.4, 3.5) requires minor, simple transformations. It's easily adaptable for the bipartite graphs.

4.4.2 Performance Comparison

All tables (Table-4.1, 4.3, 4.5, 4.2, 4.4, 4.6) contain geometrically averaged results for all instances belonging to each K-way partition, i.e., K $\in$ {64, 128, 256, 512, 1024}. Three tables (Table-4.1, 4.3, 4.5) are averaged results of the mesh-related dataset, other three tables (Table-4.2, 4.4, 4.6) are averaged results of the SPGEMM dataset.

In all tables (Table-4.1, 4.3, 4.5, 4.2, 4.4, 4.6), we have the following representations. The first column represents number of parts in given partition, i.e., K of K-way partition. In the second column, we showed number of instances we tested for the particular K value. The third and the following columns represent average results of following partitioning schemes in order: Baseline model: single-constraint hypergraph partitioning model H_{Sc} (Section 3.1), data-vertex based direct K-way hypergraph partitioning model $H_{DVB_{D_r}}$ (Section 3.2), data-vertex replication based recursive hypergraph bipartitioning model $H_{DVB_{RB}}$ (Section 3.3), inverse data weight distribution based direct K-way hypergraph partitioning model $H_{IDWD_{D_r}}$ (Section 3.6), inverse data weight distribution based recursive hypergraph bipartitioning model $H_{IDWD_{RB}}$ (Section 3.7), bipartite graph direct K-way partitioning model $\mathcal{BG}_{CDV_{D_r}}$ (Section 3.4), data-vertex replication based

Algorithm 7 Computation of Performance Metrics

Input : **Partition vector** partVec, **Hypergraph** H, K-way partitioned by $\Pi(H)$
Output: **Performance metrics**

1: $totCWeight \leftarrow 0$, $totDWeight \leftarrow 0$,
2: $maxCWeight \leftarrow 0$, $maxDWeight \leftarrow 0$;
3: $idealCWeightAvg \leftarrow 0$, $idealDWeightAvg \leftarrow 0$;
4: obtainIdealCDWegAvg($idealCWeightAvg$, $idealDWeightAvg$);
5: **for** each part P_k in $\Pi(H)$ **do**
6: $totPartCWeight \leftarrow 0$, $totPartDWeight \leftarrow 0$;
7: $N_{UniqNetSet} \leftarrow \varnothing$;
8: **for** each v_i in P_k **do**
9: $totPartCWeight$ += $w^1(v_i)$;
10: $totPartDWeight$ += $w^2(v_i)$;
11: **for** each net n_j in $Nets(v_i)$ **do**
12: **if** $N_{UniqNetSet} \cap n_j = \varnothing$; **then**
13: $N_{UniqNetSet} = N_{UniqNetSet} \cup n_j$;
14: **end if**
15: **end for**
16: **end for**
17: **for** each net n_u in $N_{UniqNetSet}$ **do**
18: $totPartDWeight$ += c_u;
19: **end for**
20: incrementTotalCDWeight($totCWeight$, $totPartCWeight$, $totDWeight$, $totPartDWeight$);
21: assignMaxCDWeight($maxCWeight$, $totPartCWeight$, $maxDWeight$, $totPartDWeight$);
22: **end for**
23: // Compute performance metric-1, i.e., $W^1_{maxLoadProp}$
24: $maxCLoadPr = maxCWeight/\ idealCWeightAvg$;
25: // Compute performance metric-2, i.e., $W^2_{maxLoadProp}$
26: $maxDLoadPr = maxDWeight/\ idealDWeightAvg$;
27: // Compute performance metric-3, i.e., $W^2_{avgLoadProp}$
28: $avgDLoadPr = (totPartDWeight/K)/\ idealDWeightAvg$;
29: **return** $maxCLoadPr$, $maxDLoadPr$, $avgDLoadPr$;

recursive bipartite graph bipartitioning model $\mathcal{BG}_{CDV_{RB}}$ (Section 3.5). Preference reason of the baseline model H_{Sc} (Section 3.1) is described in the beginning of the Chapter 3.

For data load balancing quality, we examine minimization of maximum load, i.e., maximum data load proportion metric $W^1_{maxLoadProp}$ (Table 4.1, 4.2). In both dataset, we observed significant improvements in proposed models other than the baseline model H_{Sc} according to the baseline model H_{Sc} for each K value (Table 4.1, 4.2). We also see that RB paradigm always achieves better solution as expected (e.g., $\mathcal{BG}_{CDV_{RB}}$ gave consistently better results than $\mathcal{BG}_{CDV_{Dr}}$). While $\mathcal{BG}_{CDV_{RB}}$ is the method giving best results in average in the mesh-related dataset (Table 4.1), $H_{IDWD_{RB}}$ is the best method in the SPGEMM dataset (Table 4.2).

Table 4.1: Overall performance comparison in terms of maximum data load proportion in the mesh-related dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	117	1.83	1.33	1.31	1.29	1.25	1.30	1.24
128	108	2.02	1.44	1.41	1.40	1.34	1.40	1.33
256	94	2.28	1.56	1.51	1.50	1.42	1.50	1.40
512	82	2.59	1.69	1.58	1.61	1.49	1.58	1.46
1024	63	2.88	1.71	1.59	1.66	1.50	1.60	1.48

Table 4.2: Overall performance comparison in terms of maximum data load proportion in the SPGEMM dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	69	1.66	1.12	1.10	1.11	1.09	1.13	1.10
128	63	1.87	1.15	1.12	1.14	1.11	1.17	1.13
256	54	1.94	1.19	1.14	1.18	1.13	1.19	1.15
512	45	2.03	1.23	1.16	1.22	1.15	1.22	1.18
1024	28	2.09	1.26	1.19	1.25	1.16	1.25	1.21

Average data proportion load $W^2_{avgLoadProp}$ is measuring average data assigned to processors with respect to a given task partition Π. In the mesh-related dataset (Table 4.3), best method is baseline method H_{Sc}. Effect of RB we observed in maximum data load proportion $W^2_{maxLoadProp}$ is again observed with one exception (i.e., $\mathcal{BG}_{CDV_{RB}}$ and $\mathcal{BG}_{CDV_{Dr}}$, they are almost same in average, but the case changes negatively for RB paradigm for greater K values). In the SPGEMM dataset (Table 4.4), the best model is $H_{DVB_{RB}}$, while worst result comes from the baseline model H_{Sc} in contrast to the mesh-related dataset (Table 4.3).

Table 4.3: Overall performance comparison in terms of average data load proportion in the mesh-related dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	117	1.11	1.18	1.16	1.18	1.16	1.15	1.15
128	108	1.15	1.24	1.21	1.24	1.22	1.20	1.20
256	94	1.18	1.29	1.26	1.29	1.27	1.25	1.25
512	82	1.21	1.34	1.29	1.35	1.31	1.27	1.28
1024	63	1.18	1.33	1.26	1.33	1.28	1.24	1.26

Table 4.4: Overall performance comparison in terms of average data load proportion in the SPGEMM dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	69	1.45	1.07	1.06	1.07	1.06	1.06	1.06
128	63	1.58	1.09	1.07	1.09	1.07	1.08	1.08
256	54	1.61	1.11	1.08	1.11	1.09	1.09	1.10
512	45	1.66	1.14	1.10	1.13	1.10	1.10	1.12
1024	28	1.66	1.14	1.10	1.14	1.11	1.11	1.13

Maximum computation load proportion $W^1_{maxLoadProp}$ is utilized for observing minimization of maximum computation load. It's tried to be achieved by respective tools as a first balance constraint. Via $W^1_{maxLoadProp}$, we examine computation load balance quality of a given task partition Π. In the mesh-related dataset (Table 4.5), the best method is $H_{DVB_{Dr}}$ and its closest method is $H_{IDWD_{Dr}}$. As seen from those observations, the RB effect is positive for the graph models as opposed to the the hypergraph models (Table 4.5). In the SPGEMM dataset (Table 4.6), very similar finding is reported, the best two methods are $H_{DVB_{Dr}}$, $H_{IDWD_{Dr}}$ as in the mesh-related dataset results and RB's effect is almost same. However, the proposed models work better than baseline model H_{Sc} as we expect to see.

When we evaluate maximum data load proportion $W^2_{maxLoadProp}$ results (Table 4.1, 4.2) with maximum computation load proportion $W^1_{maxLoadProp}$ results (Table 4.5, 4.6), we conclude that there are clear improvements in simultaneous data and computation load balancing purpose. While maintaining balance on given

computation load constraints, we dramatically decreased maximum data load.

One could observe instance-by-instance performance assessments of the methods on the datasets. Especially, fractional performance evaluations on the datasets of the methods could be seen by scrutinizely analyzing performance profiles presented in Appendix A.

Table 4.5: Overall performance comparison in terms of maximum computation load proportion in the mesh-related dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	117	1.04	1.03	1.04	1.03	1.04	1.05	1.04
128	108	1.04	1.03	1.04	1.03	1.04	1.05	1.04
256	94	1.05	1.03	1.05	1.04	1.05	1.06	1.05
512	82	1.06	1.04	1.05	1.05	1.06	1.06	1.06
1024	63	1.06	1.03	1.05	1.04	1.06	1.06	1.05

Table 4.6: Overall performance comparison in terms of maximum computation load proportion in the SPGEMM dataset

K	#ofInstances	Bas.: H_{Sc}	$H_{DVB_{Dr}}$	$H_{DVB_{RB}}$	$H_{IDWD_{Dr}}$	$H_{IDWD_{RB}}$	$BG_{CDV_{Dr}}$	$BG_{CDV_{RB}}$
64	69	1.03	1.02	1.02	1.02	1.03	1.05	1.04
128	63	1.03	1.02	1.02	1.02	1.03	1.05	1.04
256	54	1.03	1.02	1.02	1.02	1.03	1.05	1.04
512	45	1.03	1.02	1.02	1.02	1.03	1.05	1.04
1024	28	1.03	1.02	1.03	1.02	1.03	1.05	1.04

Chapter 5

Conclusion

Graph/hypergraph partitiong is widely used to partition workload in a way that the computation load among the processors would be balanced for efficient parallelization of several irregularly sparse applications while minimizing the total communication volume. In that regard, several successful partitioning models and methods have been proposed and used for computational load balancing of irregularly sparse applications on distributed-memory architectures.

Besides, data load is also an important issue while fulfilling the computation requirements. In the thesis, we proposed graph/hypergraph partitioning models and methods for simultaneous computation and data load balancing. We utilized the RB framework for better encoding data size of processor, proposed the split data vertex replication in our RB framework for the bipartite graphs and the hypergraphs and also proposed inverse data weight distribution for the hypergraphs.

Empirical validity of the proposed models and methods is tested and confirmed by assessing the performance of the models on a wide range of sparse matrices related to distributed parallel mesh simulations and C = AB kind row-row parallel SPGEMM applications. With the proposed models and methods, we observed

significant improvements in simultaneous computation load and data size balancing purpose. While maintaining balance on given computation load constraints, we dramatically decreased maximum data size.

Bibliography

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

[2] S. Morais, "Etude et obtention d'heuristiques et d'algorithmes exacts et approch pour un proble de partitionnement de maillage sous contraintes moire.," *PhD Thesis, Université Paris Saclay*, 2016.

[3] F. Ledoux, S. Morais, C. Chevalier, E. Angel, and D. Regnault, "A multilevel mesh partitioning algorithm driven by memory constraints," *SIAM Workshop on Combinatorial Scientific Computing(CSC 16) Proceedings*, 2016.

[4] K. Akbudak, O. Selvitopi, and C. Aykanat, "Partitioning models for scaling parallel sparse matrix-matrix multiplication," *ACM Transactions on Parallel Computing (TOPC)*, vol. 4, no. 3, pp. 964 – 972, 2018.

[5] A. Buluc and J. R. Gilbert, "Highly parallel sparse matrix-matrix multiplication," *Tech. Rep. UCSB-CS-2010-10, University of California, Santa Barbara, Computer Science Department*, 2010.

[6] A. Buluc and J. R. Gilbert, "On the representation and multiplication of hypersparse matrices," *IEEE International Symposium on Parallel and Distributed Processing, IPDPS'08*, pp. 1–11, 2008.

[7] A. Buluc and J. R. Gilbert, "Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments," *SIAM Journal of Scientific Computing (SISC)*, vol. 34, no. 4, pp. 170–191, 2012.

[8] V. Shah, "An interactive system for combinatorial scientific computing with an emphasis on programmer productivity," *PhD thesis, University of California Santa Barbara*, 2007.

[9] J. R. Gilbert, V. B. Shah, and S. Reinhardt, "A unified framework for numerical and combinatorial computing," *Computing in Science Engineering*, vol. 10, no. 2, pp. 20–25, 2008.

[10] R. Yuster and U. Zwick, "Detecting short directed cycles using rectangular matrix multiplication and dynamic programming," *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '04*, pp. 254–260, 2004.

[11] P. D'Alberto and A. Nicolau, "A high-performance divide-and-conquer algorithm for the all-pair shortest path for densely connected networks," *Algorithmica*, vol. 47, no. 2, pp. 203–213, 2007.

[12] M. Rabin and V. Vazirani, "Maximum matchings in general graphs through randomization," *1989*, vol. 10, no. 4, pp. 557–567, 1989.

[13] E. Boman, O. Parekh, and C. Phillips, "Ldrd final report on massively-parallel linear programming: the parpcx system," *tech. rep., SAND2004-6440, Sandia National Laboratories*, 2005.

[14] R. H. Bisseling, T. M. Doup, and L. Loyens, "A parallel interior point algorithm for linear programming on a network of transputers," *Annals of Operations Research*, vol. 43, pp. 51–86, 1993.

[15] G. Karypis, A. Gupta, and V. Kumar, "A parallel formulation of interior point algorithms," *Supercomputing 94*, 1994.

[16] "Cp2k home page," *http://www.cp2k.org/*.

[17] A. Daniels, J. Millam, and G. Scuseria, "Semiempirical methods with conjugate gradient density matrix search to replace diagonalization for molecular systems containing thousands of atoms," *The Journal of Chemical Physics*, vol. 107, no. 2, pp. 425–431, 1997.

[18] J. Millam and G. Scuseria, "Linear scaling conjugate gradient density matrix search as an alternative to diagonalization for first principles electronic structure calculations," *The Journal of Chemical Physics*, vol. 106, no. 13, p. 5569–5577, 1997.

[19] X. P. Li, R. Nunes, and D. Vanderbilt, "Density-matrix electronic-structure method with linear system-size scaling," *Physical Review B*, vol. 47, p. 10891–10894, 1993.

[20] H. B. Schlegel, J. M. Millam, S. S. Iyengar, G. A. Voth, A. D. Daniels, G. E. Scuseria, and M. J. Frisch, "Ab initio molecular dynamics: Propagating the density matrix with gaussian orbitals," *The Journal of Chemical Physics*, vol. 114, no. 22, p. 9758–9763, 2001.

[21] S. Itoh, P. Ordejn, and R. M. Martin, "Order-n tight-binding molecular dynamics on parallel computers," *Computer Physics Communications*, vol. 88, no. 2-3, pp. 173–185, 1995.

[22] M. Challacombe, "A simplified density matrix minimization for linear scaling self-consistent field theory," *The Journal of Chemical Physics*, vol. 110, no. 5, pp. 2332–2342, 1999.

[23] J. VandeVondele, U. Borstnik, and J. Hutter, "Linear scaling self-consistent field calculations with millions of atoms in the condensed phase," *Journal of Chemical Theory and Computation*, vol. 8, no. 10, p. 3565–3573, 2012.

[24] M. Challacombe, "A general parallel sparse-blocked matrix multiply for linear scaling scf theory," *Computer Physics Communications*, vol. 128, no. 12, pp. 93–107, 2000.

[25] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[26] K. Akbudak, "Hypergraph models for parallel sparse matrix-matrix multiplication," *PhD Thesis, Bilkent Univerrsity*, 2015.

[27] K. Kaya, "Iterative-improvement-based heuristics for adaptive scheduling of tasks sharing files on heterogeneous master-slave environments," *MSc Thesis, Bilkent Univerrsity*, 2004.

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

[29] B. Dal, "A bipartite graph model for placement, scheduling and replication in data grids," *MSc Thesis, Bilkent University*, 2012.

[30] B. Hendrickson and R. Leland, "The chaco user's guide version 2.0," *Sandia National Laboratories. Albuquerque, NM (United States)*, 1995.

[31] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs.," *International Conference on Parallel Processing*, pp. 113 – 112, 1995.

[32] G. Karypis and V. Kumar, "Metis a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices version 5.1.0," *University of Minnesota, Minneapolis, MN*, 2013.

[33] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *10th Intl. Parallel Processing Symposium*, pp. 314 – 319, 1996.

[34] G. Karypis and K. Schloegel, "Parmetis parallel graph partitioning and sparse matrix ordering library version 4.0," *University of Minnesota, Minneapolis, MN*, 2013.

[35] R. Preis and R. Diekmann, "Party-a software library for graph partitioning," *Advances in Computational Mechanics with Parallel and Distributed Processing*, pp. 63 – 71, 1997.

[36] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs,"

High-Performance Computing and Networking. HPCN-Europe 1996. Lecture Notes in Computer Science, vol. 1067, pp. 493 – 498, 1996.

[37] F. Pellegrini, "Scotch and libscotch 6.0 user's guide," *Bacchus team, INRIA Bordeaux Sud-Ouest Technical Report: CNRS*, vol. 5800, 2012.

[38] C. Chevalier and F. Pellegrini, "Pt-scotch: A tool for efficient parallel graph ordering," *Parallel computing*, vol. 34, no. 6-8, pp. 318 – 331, 2008.

[39] F. Pellegrini, "Pt-scotch and libptscotch 6.0. user's guide," *Technical report, INRIA Bordeaux Sud-Ouest*, 2012.

[40] C. Walshaw and M. Cross, "Jostle: parallel multilevel graph-partitioning software–an overview," *Mesh partitioning techniques and domain decomposition techniques*, pp. 27 – 58, 2007.

[41] C. Walshaw, "The jostle executable user guide : Version 3.1," *University of Greenwich, London, England.*, 2005.

[42] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale, "Drama: A library for parallel dynamic load balancing of finite element applications," *European Conference on Parallel Processing*, pp. 313 – 316, 1999.

[43] U. V. Çatalyürek and C. Aykanat, "Patoh (partitioning tool for hypergraphs)," *Encyclopedia of Parallel Computing*, pp. 1479 – 1487, 2011.

[44] A. Caldwell, A. Kahng, and I. Markov, "Improved algorithms for hypergraph bipartitioning," *Proceedings of the IEEE ACM Asia and South Pacific Design Automation Conference*, pp. 661–666, 2000.

[45] 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.

[46] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in vlsi domain," *34th Design and Automation Conference*, pp. 526 – 529, 1997.

[47] G. Karypis and V. Kumar, "hmetis a hypergraph partitioning package version 1.5.3," *University of Minnesota, Minneapolis, MN*, 1998.

[48] A. Trifunovic and W. Knottenbelt, "A parallel algorithm for multilevel k-way hypergraph partitioning," *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004.*, pp. 114 – 121, 2004.

[49] A. Trifunovic and W. Knottenbelt, "Parkway 2.0: A parallel multilevel hypergraph partitioning tool," *Computer and Information Sciences - ISCIS 2004*, pp. 789 – 800, 2004.

[50] R. Barat, "Load balancing of multi-physics simulation by multi-criteria graph partitioning.," *PhD Thesis, Université de Bordeaux*, 2017.

[51] R. Barat, C. Chevalier, and F. Pellegrini, "Multi-criteria graph partitioning with scotch," pp. 66–75, SIAM, 2018.

[52] U. V. Çatalyürek and C. Aykanat, "A hypergraph-partitioning approach for coarse-grain decomposition," *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing*, p. 28, 2001.

[53] S. Acer, O. Selvitopi, and C. Aykanat, "Improving performance of sparse matrix dense matrix multiplication on large-scale parallel systems," *Elsevier, Parallel Computing*, vol. 59, pp. 71 – 96, 2016.

[54] G. Karypis, "Multilevel algorithms for multi-constraint graph partitioning," *University of Minnesota, Minneapolis, MN, Technical Report 98-019*, 1999.

[55] G. Karypis, "Multilevel algorithms for multi-constraint hypergraph partitioning," *University of Minnesota, Minneapolis, MN, Technical Report 99-034*, 1999.

[56] G. Karypis, "Multilevel algorithms for multi-constraint hypergraph partitioning," *University of Minnesota, Minneapolis, MN, Technical Report 03-022*, 2003.

[57] M. R. Garey, D. S. Johnson, and L. Stockmeyer, "Some simplified np-complete problems," *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, STOC '74*, pp. 47–63, 1974.

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

[59] C. Kring and A. R. Newton, "A cell-replicating approach to mincut-based circuit partitioning.," *Comput.-Aided Design, 1991. ICCAD- 91. Digest of Technical Papers., 1991 IEEE Internat. Conf.*, p. 2–5, 1991.

[60] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli, "On clustering for minimum delay/area.," *Comput.-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE Internat. Conf.*, pp. 6–9, 1991.

[61] J. Hwang and A. E. Gamal, "Optimal replication for min-cut partitioning," *ICCAD '92: Proc. 1992 IEEE/ACM Internat. Conf. Comput.-Aided Design*, pp. 432–435, 1992.

[62] R. Kužnar, F. Brglez, and B. Zajc, "Multi-way netlist partitioning into heterogeneous fpgas and minimization of total device cost and interconnect.," *DAC '94: Proc. 31st Annual Design Automation Conf.*, p. 238–243, 1994.

[63] H. H. Yang and D. F. Wong, "New algorithms for min-cut replication in partitioned circuits.," *Proceedings 1995 IEEE/ACM International Conference Computer Aided Design*, pp. 216–222, 1995.

[64] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integrated VLSI Journal*, vol. 19, pp. 1–81, 1995.

[65] M. Enos, S. Hauck, and M. Sarrafzadeh, "Replication for logic bipartitioning.," *Computer-Aided Design, 1997. Digest of Technical Papers, 1997 IEEE/ACM International Conference*, pp. 432–349, 1997.

[66] E. Demir, C. Aykanat, and B. B. Cambazoglu, "Clustering spatial networks for aggregate query processing: A hypergraph approach," *Information System*, vol. 33, pp. 1–17, 2008.

[67] R. O. Selvitopi, A. Türk, and C. Aykanat, "Replicated partitioning for undirected hypergraphs.," *Journal of Parallel Distributed Computing*, vol. 72, p. 547–563, 2012.

[68] R. Selvitopi, "Replicated hypergraph partitioning," *MSc Thesis, Bilkent Univerrsity*, 2010.

[69] R. O. Selvitopi, A. Türk, and C. Aykanat, "rppatoh: Replicated partitioning tool for hypergraphs," tech. rep., Tech. Rep. BU-CE-1203, Bilkent Univ. Comp. Eng. Dept, 2012.

[70] V. Yazıcı, "Balance preserving min-cut replication set for a k-way hypergraph partitioning," *MSc Thesis, Bilkent Univerrsity*, 2010.

[71] V. Yazici and C. Aykanat, "Constrained min-cut replication for k-way hypergraph partitioning," *INFORMS Journal on Computing*, vol. 26, no. 2, pp. 303–320, 2013.

[72] A. Türk, R. O. Selvitopi, H. Ferhatosmanoglu, and C. Aykanat, "Temporal workload-aware replicated partitioning for social networks," *IEEE Transactions on knowledge and Data Engineering*, vol. 26, no. 11, pp. 2832–2845, 2014.

[73] "Bipartite graph," *http://mathworld.wolfram.com/BipartiteGraph.html*.

[74] "Hypergraph," *https://www.encyclopediaofmath.org/index.php/Hypergraph*.

[75] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS), volume=*.

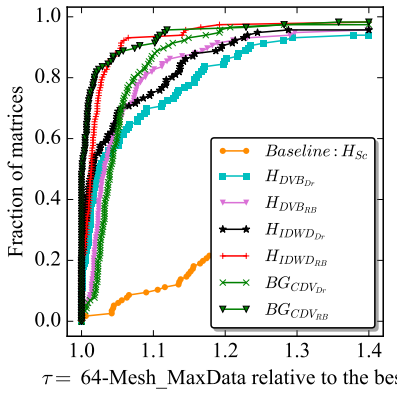
[76] L. N. Olson and J. B. Schroder, "Pyamg: Algebraic multigrid solvers in python v4.0," 2018. Release 4.0.

[77] J. E. Flaherty, R. M. Loy, P. C. Scully, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, "Load balancing and communication optimization for parallel adaptive finite element methods," in *Computer Science Society, 1997. Proceedings., XVII International Conference of the Chilean*, pp. 246–255, IEEE, 1997.

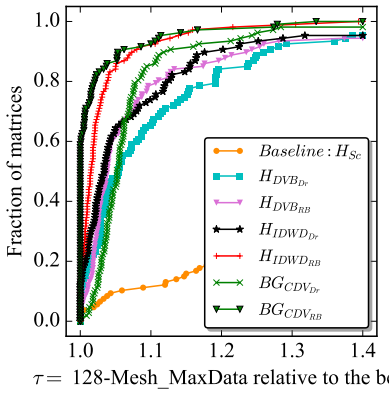
Appendix A

Performance Profiles

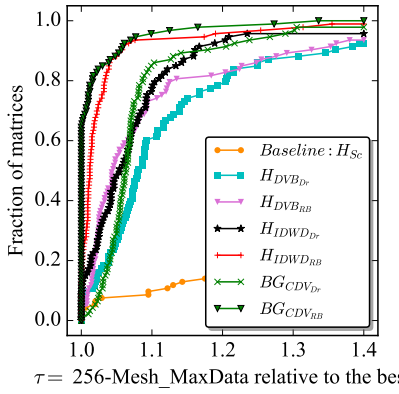
A.1 Dataset related to the Mesh Applications: Performance Profiles of the Models



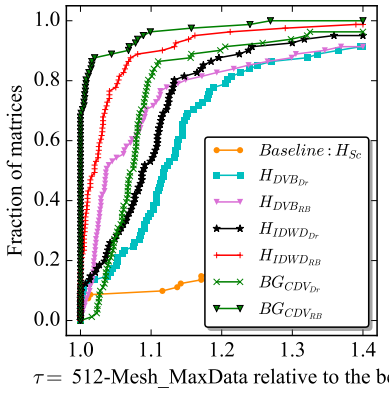
(a) Max. data load prop. metric - Performance profile of the models, K = 64



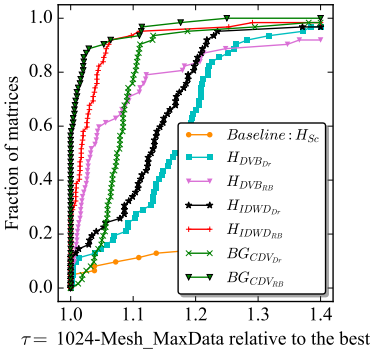
(b) Max. data load prop. metric - Performance profile of the models, K = 128



(c) Max. data load prop. metric - Performance profile of the models, K = 256

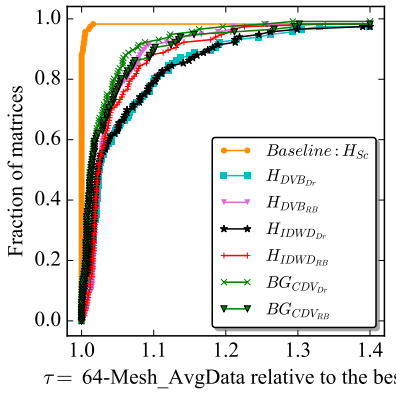


(d) Max. data load prop. metric - Performance profile of the models, K = 512

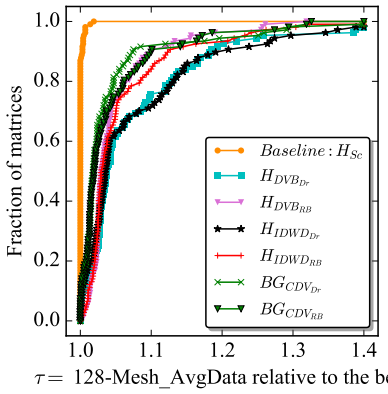


(e) Max. data load prop. metric - Performance profile of the models, K = 1024

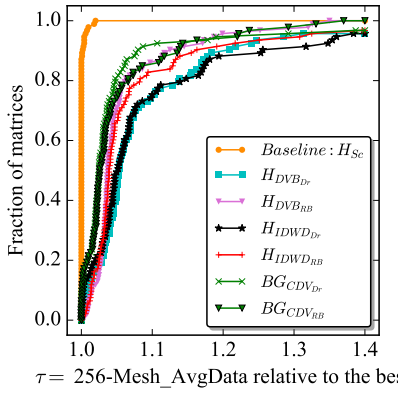
Figure A.1: Performance profiles for maximum data load proportion metric in dataset related to the mesh applications for K $\in$ {64, 128, 256, 512, 1024}



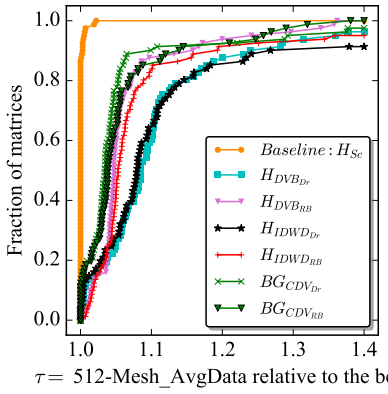
(a) Avg.data load prop. metric - Performance profile of the models, K = 64



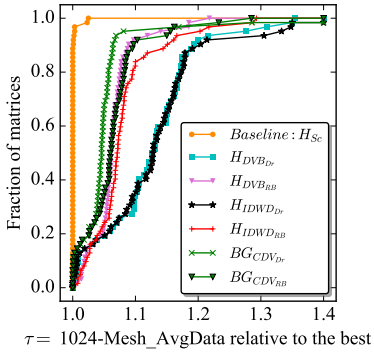
(b) Avg.data load prop. metric - Performance profile of the models, K = 128



(c) Avg.data load prop. metric - Performance profile of the models, K = 256

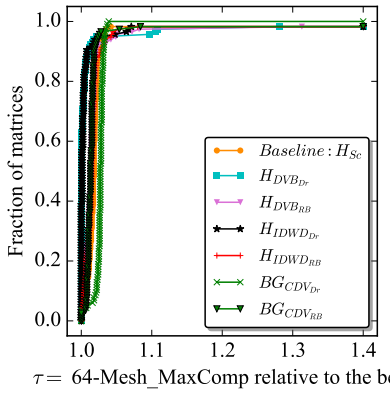


(d) Avg.data load prop. metric - Performance profile of the models, K = 512

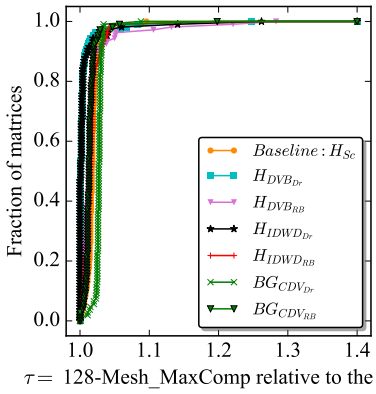


(e) Avg.data load prop. metric - Performance profile of the models, K = 1024

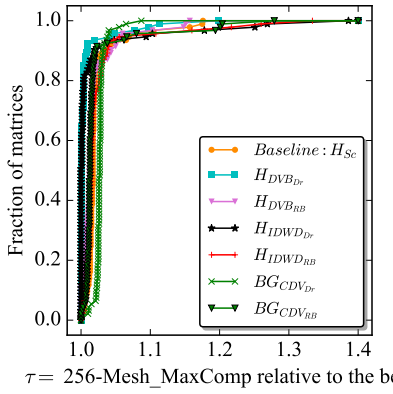
Figure A.2: Performance profiles for average data load proportion metric in dataset related to the mesh applications for K ∈ {64, 128, 256, 512, 1024}



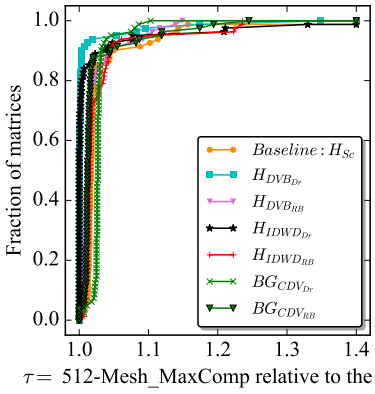
(a) Max. comp. load prop. metric - Performance profile of the models, K = 64



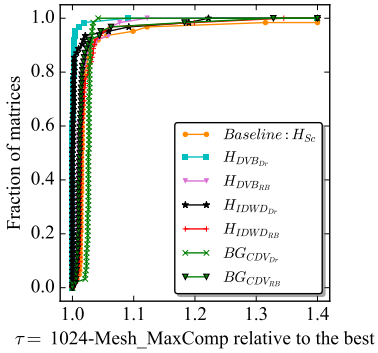
(b) Max. comp. load prop. metric - Performance profile of the models, K = 128



(c) Max. comp. load prop. metric - Performance profile of the models, K = 256



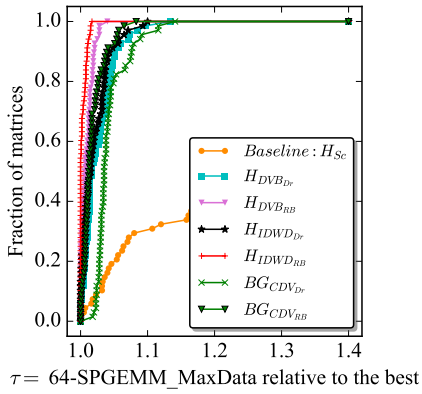
(d) Max. comp. load prop. metric - Performance profile of the models, K = 512



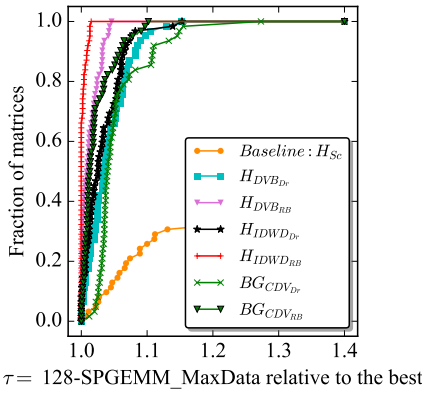
(e) Max. comp. load prop. metric - Performance profile of the models, K = 1024

Figure A.3: Performance profiles for maximum computation load proportion metric in dataset related to the mesh applications for K $\in$ {64, 128, 256, 512, 1024}

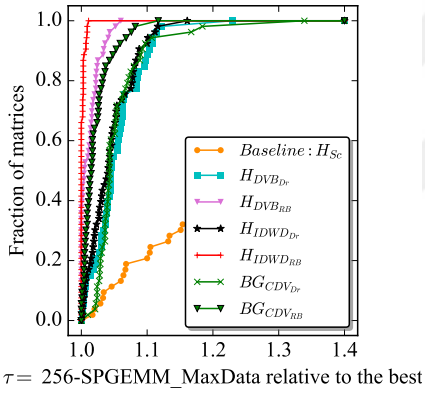
A.2 The C= AB kind SPGEMM Dataset: Performance Profiles of the Models



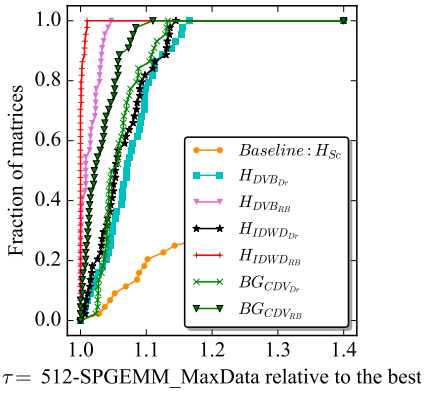
(a) Max. data load prop. metric - Performance profile of the models, K = 64



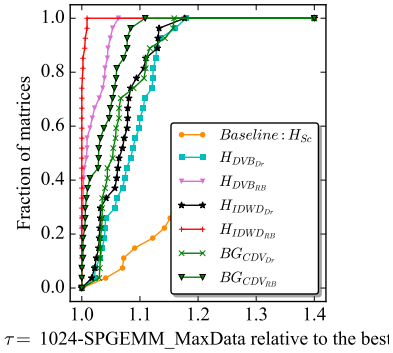
(b) Max. data load prop. metric - Performance profile of the models, K = 128



(c) Max. data load prop. metric - Performance profile of the models, K = 256

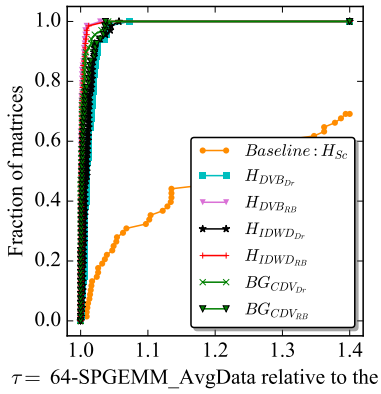


(d) Max. data load prop. metric - Performance profile of the models, K = 512

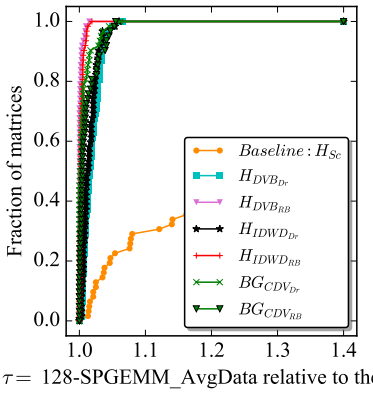


(e) Max. data load prop. metric - Performance profile of the models, K = 1024

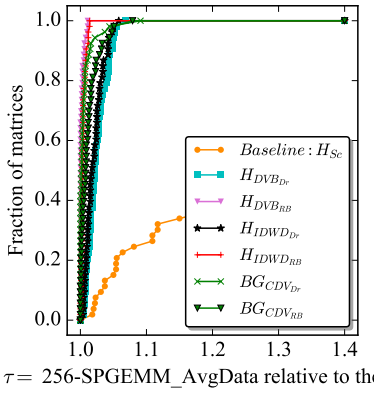
Figure A.4: Performance profiles for maximum data load proportion metric in the C= AB kind SPGEMM dataset for K $\in$ {64, 128, 256, 512, 1024}



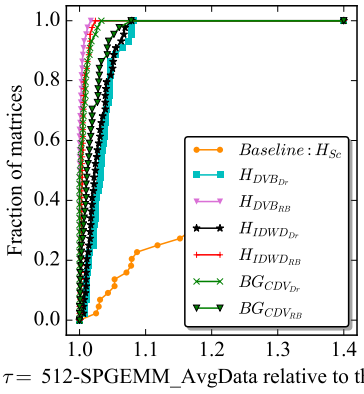
(a) Avg.data load prop. metric - Performance profile of the models, K = 64



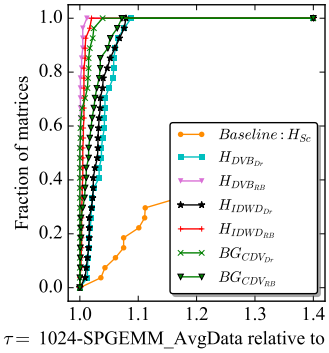
(b) Avg.data load prop. metric - Performance profile of the models, K = 128



(c) Avg.data load prop. metric - Performance profile of the models, K = 256

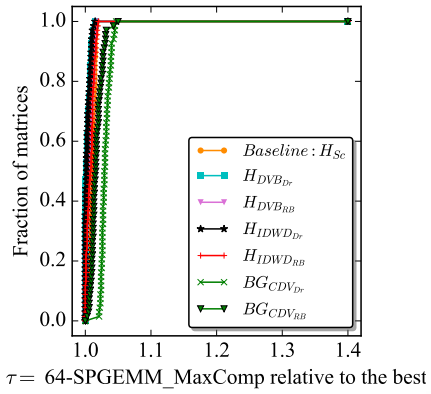


(d) Avg.data load prop. metric - Performance profile of the models, K = 512

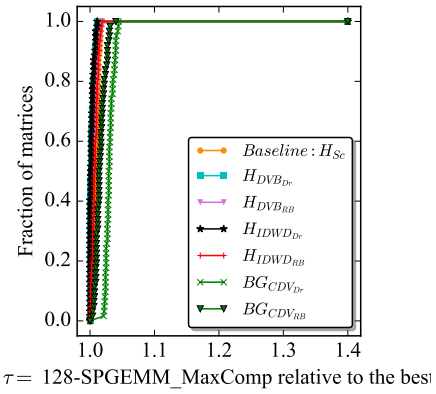


(e) Avg.data load prop. metric - Performance profile of the models, K = 1024

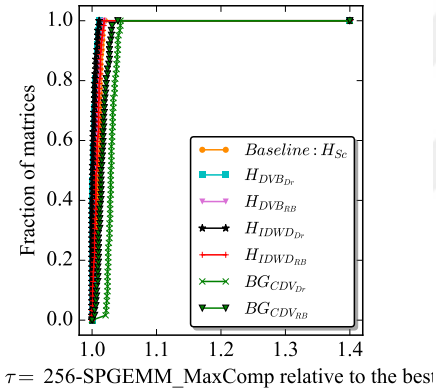
Figure A.5: Performance profiles for average data load proportion metric in the C= AB kind SPGEMM dataset for K ∈ {64, 128, 256, 512, 1024}



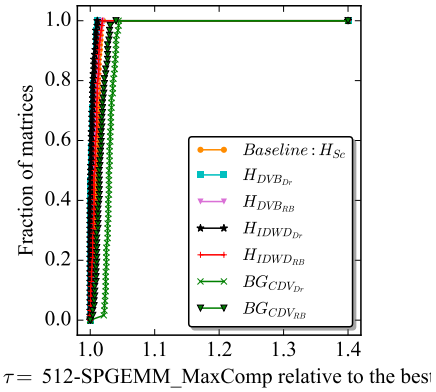
(a) Max. comp. load prop. metric - Performance profile of the models, K = 64



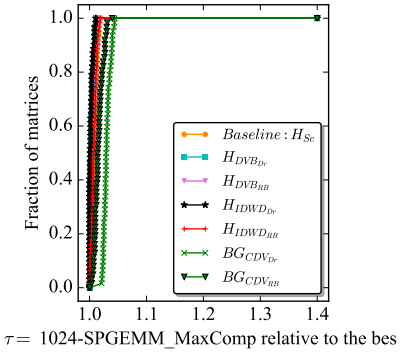
(b) Max. comp. load prop. metric - Performance profile of the models, K = 128



(c) Max. comp. load prop. metric - Performance profile of the models, K = 256



(d) Max. comp. load prop. metric - Performance profile of the models, K = 512



(e) Max. comp. load prop. metric - Performance profile of the models, K = 1024

Figure A.6: Performance profiles for maximum computation load proportion metric in the C= AB kind SPGEMM dataset for K $\in$ {64, 128, 256, 512, 1024}