SEND VOLUME BALANCING IN REDUCE OPERATIONS

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
Muhammed Çavuşoğlu
July 2020

Send Volume Balancing in Reduce Operations
By Muhammed Çavuşoğlu
July 2020

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)

Can Alkan

Fahreddin Şükrü Torun

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

SEND VOLUME BALANCING IN REDUCE OPERATIONS

Muhammed Çavuşoğlu
M.S. in Computer Engineering
Advisor: Cevdet Aykanat
July 2020

We investigate balancing send volume in applications that involve reduce operations. In such applications, a given computational-task-to-processor mapping produces partial results generated by processors to be reduced possibly by other processors, thus incurring inter-processor communication. We define the reduce communication task assignment problem as assigning the reduce communication tasks to processors in a way that minimizes the send volume load of the maximally loaded processor. We propose one novel independent-task-assignment-based algorithm and four novel bin-packing-based algorithms to solve the reduce communication task assignment problem. We validate our proposed algorithms on two kernel operations: sparse matrix-sparse matrix multiplication (SpGEMM) and sparse matrix-matrix multiplication (SpMM). Experimental results show improvements of up to 23% on average for the maximum communication volume cost metric in SpGEMM and up to 12% improvement on average in SpMM.

Keywords: Sparse matrices, maximum communication volume, bipartite graphs, independent task assignment problem, bin packing problem.

ÖZET

İNDİRGEME İŞLEMLERİNDE GÖNDERME YÜKÜNÜN DENGELENMESİ

Muhammed Çavuşoğlu
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Cevdet Aykanat
Temmuz 2020

İndirgeme işlemleri içeren uygulamalarda gönderme yükünü dengeleme incelenmektedir. Bu tür uygulamalarda, verilen bir hesaplama işi-işlemci ataması; işlemcilerin, muhtemelen diğer işlemciler tarafından indirgenmek üzere ürettiği kısmi sonuçlar oluşturmaktadır. Bu durum, işlemciler arası iletişime neden olmaktadır. İndirgeme iletişim işi atama problemi; indirgeme iletişim işlerinin, gönderme yükü en fazla olan işlemcinin yükünü en aza indirgeyecek şekilde işlemcilere atanması olarak tanımlanmaktadır. Bu indirgeme iletişim işi atama problemini çözmek için bir adet bağımsız iş atama problemi bazlı ve dört adet kutu istifleme problemi bazlı yeni algoritma sunulmaktadır. Sunulan algoritmaların başarımı, seyrek matris-seyrek matris çarpımı (SpGEMM) ve seyrek matris-matris çarpımı (SpMM) çekirdek işlemleri için doğrulanmıştır. Deneysel sonuçlar; azami iletişim yükü metriğinde, SpGEMM'de ortalama %23'e varan bir iyileştirme, SpMM'de ise ortalama %12'ye varan bir iyileştirme olduğunu göstermektedir.

Anahtar sözcükler: Seyrek matrisler, azami iletişim yükü, iki bölmeli çizgeler, bağımsız iş atama problemi, kutu istifleme problemi.

Acknowledgement

I would like to thank my advisor Prof. Cevdet Aykanat for his trust in me, guidance, support, and insightful suggestions. I would like to thank the Scientific and Technological Research Council of Turkey (TÜBİTAK) 1001 program for supporting me in the EEEAG-119E035 project. I would like to thank the rest of my thesis committee, Asst. Prof. Can Alkan and Asst. Prof. Fahreddin Şükrü Torun, for reading my thesis and providing valuable feedback. I would also like to thank Mustafa Ozan Karsavuran for sharing his experience and suggestions and answering my endless questions.

I am grateful to my one and only Beliz Uslu for always supporting me, always listening to me patiently, and always being by my side. Your words of comfort and your love never failed to motivate me and cheer me up.

I must express my sincerest gratitude to my father Cengiz Çavuşoğlu, MD, my mother Suzan Çavuşoğlu, MD, and my sister Nur Çavuşoğlu for their neverending love and support; and for being a great company in the quarantine times. Your sacrifices deserve all I can give.

I owe special thanks to my friends at EA-507 for making the office feel like home, and to Duygu Durmuş for being an amazing colleague and a lifelong friend.

Finally, I want to dedicate this thesis to everyone who is working to minimize the impact of the pandemic. I am thankful for all the researchers around the world who are trying to find a cure and for all the healthcare professionals who are fighting the virus. Thanks to you, I never lost my hope when working on my thesis from home.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The focus of this thesis is communication volume balancing by reducing the send volume load of the maximally loaded processor in reduce operations. In reduce operations, each processor gathers data from each of its source processors and reduces these data into a final value.

There are successful hypergraph-based [1] and bipartite-graph-based [2] methods for the minimization of communication cost metrics of reduce operations. However, [1] focuses only on the parallel sparse matrix-vector multiplication (SpMV) problem, and [2] on the parallel sparse matrix-sparse matrix multiplication (SpGEMM) problem. We adapt the bipartite graph proposed in [2] to propose a framework that can be used to model different applications involving reduce operations.

Our bipartite-graph-model-based framework models the computational phase and the communication phase of reduce operations. In the computational phase, processors execute computational tasks that produce partial results, each of which needs to be reduced to a final value. In the communication phase, partial results are communicated to the processors responsible for final results. We refer to the items that the partial results are reduced to as reduce communication tasks. Partitioning this bipartite graph corresponds to assigning computational tasks

and reduce communication tasks to processors. However, the assignment of reduce communication tasks to processors by partitioning fails to minimize the send load of the maximally loaded processor, and more sophisticated algorithms are necessary to achieve this goal.

To solve this, we define the reduce communication task assignment problem, where under a given computational task assignment, the goal is to assign reduce communication tasks to processors in a way that minimizes the send volume load of the maximally loaded processor. We propose five novel algorithms to solve this problem. We show the validity of our algorithms on outer-product-parallel versions of two important kernel operations: SpGEMM and sparse-matrix-matrix multiplication (SpMM).

The rest of this thesis is organized as follows: Chapter 2 gives preliminaries. Chapter 3 presents related work. Chapter 4 defines the reduce communication task assignment problem, introduces our bipartite graph model for task assignments, and explains the adaptations of our framework to two sample applications. Our proposed algorithms are described in Chapter 5. Experimental setup, data, and results are presented and discussed in Chapter 6. Lastly, the thesis is concluded with possible future work in Chapter 7.

Chapter 2

Background

In this chapter, we review the definition of graph and graph partitioning problem.

2.1 Graph Partitioning

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. Each vertex $v_i \in \mathcal{V}$ has weight $w(v_i)$ and each edge $e_{i,j} \in \mathcal{E}$ connecting distinct vertices v_i and v_j has cost $c(e_{i,j})$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2 \dots \mathcal{V}_K\}$ is a K-way partition of the vertices in $\mathcal{G}$ if each part $\mathcal{V}_k \in \Pi$ is non-empty, all parts are mutually exclusive (i.e., $\mathcal{V}_k \cap \mathcal{V}_m = \emptyset$ for $k \neq m$), and the union of parts is $\mathcal{V}$ (i.e., $\bigcup_{\mathcal{V}_k \in \Pi} = \mathcal{V}$).

Weight $w(\mathcal{V}_k)$ of part $\mathcal{V}_k$ is the sum of the weights of the vertices in that part. That is,

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

A partition Π is said to be balanced if all parts $\mathcal{V}_k \in \Pi$ satisfy the balance constraint

$$w(\mathcal{V}_k) \leq w(\mathcal{V}_{\text{avg}})(1 + \epsilon), \quad \text{for } 1 \leq k \leq K, \tag{2.2}$$

where $w(\mathcal{V}_{\text{avg}})$ is the average part weight (i.e., $w(\mathcal{V}_{\text{avg}}) = \sum_{v_i \in \mathcal{V}} w(v_i)/K$) and ϵ is a given imbalance parameter.

An edge $e_{i,j}$ is said to be cut if v_i and v_j are in different parts, and uncut, otherwise. The set of cut edges is denoted with $\mathcal{E}^C$. The cutsize of a partition Π is defined as:

$$cutsize(\Pi) = \sum_{e_{i,j} \in \mathcal{E}^C} c(e_{i,j}). \tag{2.3}$$

In general, the objective of graph partitioning is to minimize the cutsize (2.3) while maintaining the balance constraint (2.2) defined on part weights.

Chapter 3

Related Work

We briefly review the independent task assignment problem [3] since we propose a reformulation of the reduce communication task assignment problem as an instance of the independent task assignment problem. In the independent task assignment problem, we have a set $\mathcal{T} = \{t_1, t_2, \ldots, t_N\}$ of N independent tasks, a set $\mathcal{P} = \{P_1, P_2, \ldots, P_K\}$ of K processors, and an expected-time-to-compute matrix $\mathbf{X} = (x_{i,k})$ where assigning t_i to P_k increases the load of P_k by $x_{i,k}$. MinMin [3], MaxMin [3, 4], Sufferage (Suff) [5] are different heuristics used for solving the independent task assignment problem.

Tabak et al. [6] proposed, among other heuristics, MinMin+; which utilizes priority queues to improve the asymptotic time complexity of MinMin from $O(KN^2)$ to $O(KN \log N)$. We utilize priority queues in MaxMinPQ in a similar fashion; though adapting MaxMin instead of MinMin.

Csirik et al. [7] proposed the Sum-of-Squares Algorithm for bin packing. We adapted this algorithm to create our BP-MaxVol-SumSqrs and BP-TotVol-SumSqrs algorithms.

Akbudak et al. [2] proposed hypergraph and bipartite graph models for outer-product-parallel, inner-product-parallel, and row-by-row-product-parallel formulations of SpGEMM. We use outer-product-parallel SpGEMM as a sample application and adapt a similar bipartite graph model to model task assignments.

Chapter 4

Framework

In this chapter, we define the reduce communication task assignment problem, provide a bipartite graph model for task assignments, and discuss two example applications.

4.1 Problem Definition

The target parallel application consists of a computational phase followed by a communication phase performed in an iterative manner. The computational phase involves processors producing partial results, each of which needs to be reduced to a final value. In the communication phase, partial results are communicated to the processors responsible for the final results. We will refer to the computation operations as computational tasks and the items that the partial results are reduced to as reduce communication tasks.

Two types of assignments are needed to carry out these two phases. We assign computational tasks to processors for the computational phase and assign reduce communication tasks to processors to make them responsible for gathering partial results and then locally computing the final results. Assume that the computational task to processor assignment (as shown in Figure 4.1a) has already

been determined.

Let $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$ denote the set of all reduce communication tasks for which at least two different processors produce a partial result. Let $results(P_k) \subseteq \mathcal{R}$ denote the reduce communication tasks for which processor P_k produces partial results. Let $partials(P_k, r_i)$ denote the partial results produced by processor P_k for the reduce communication task r_i. Let $processors(r_i)$ denote the set of processors that produce partials for r_i. Figure 4.1b illustrates three processors that produce partial results to six different reduce communication tasks. Numbers above the edges represent the number of partial results generated by processors for reduce communication tasks, i.e., $|partials(P_k, r_i)|$ for processor P_k and reduce communication task r_i. For instance, processor P_3 produces two partial results for each of the reduce communication tasks r_3, r_4, r_5, and r_6, i.e., $results(P_3) = \{r_3, r_4, r_5, r_6\}$. Reduce communication task r_3 needs two partial results from both processors P_2 and P_3, i.e., $processors(r_3) = \{P_2, P_3\}$.

$\Pi^{\mathcal{R}} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_K\}$ is a K-way partition of reduce communication tasks where tasks in $\mathcal{R}_k$ are assigned to processor P_k (i.e., P_k is responsible for reduce communication tasks in $\mathcal{R}_k$). Hence, processor P_k is responsible for sending the partial results that will be reduced by other processors.

Communication volume load of processor P_k, i.e., number of words sent by P_k, incurred by partition $\Pi^{\mathcal{R}}$ is

$$vol(P_k) = \sum_{r_i \in \{results(P_k) - \mathcal{R}_k\}} |partials(P_k, r_i)|. \tag{4.1}$$

Then, total communication volume load of a system with K processor becomes

$$vol_{total} = \sum_{k=1}^{K} vol(P_k). \tag{4.2}$$

Number of messages sent by processor P_k, i.e., the number of distinct processors to which the reduce communication tasks in $results(P_k) - \mathcal{R}_k$ are assigned, incurred by partition $\Pi^{\mathcal{R}}$ is

$$msg(P_k) = |\{\exists \mathcal{R}_m \text{ s.t. } m \neq k \ \wedge \ results(P_k) \cap \mathcal{R}_m \neq \emptyset\}|. \tag{4.3}$$

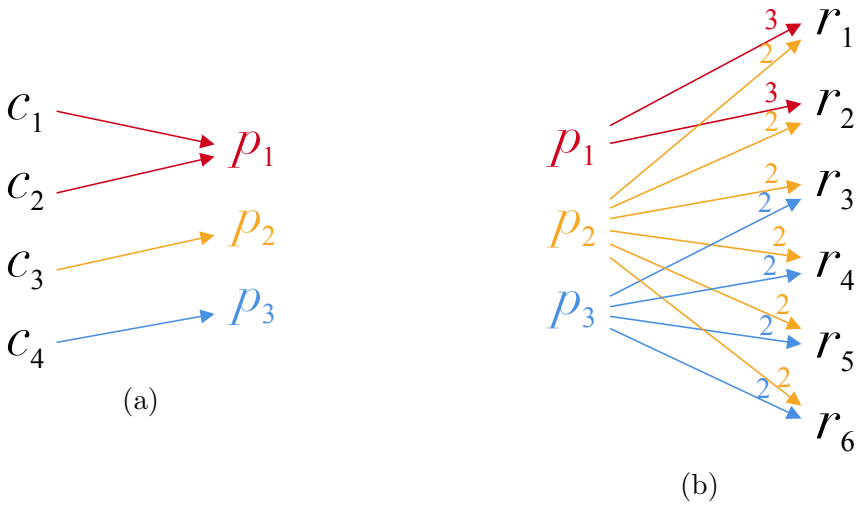

Figure 4.1: (a) Four computational tasks assigned to three processors and (b) Three processors that produce partial results to six reduce communication tasks

Then, total number of messages in this system becomes

$$msg_{total} = \sum_{k=1}^{K} msg(P_k). \tag{4.4}$$

The reduce communication task assignment problem is defined as follows:

Reduce Communication Task Assignment Problem. *Given a set of reduce communication tasks* $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$*, together with the partial results* $partials(P_k, r_i)$ *generated by each processor* P_k *for each reduce communication task* r_i*; find a* K*-way partition* $\Pi^{\mathcal{R}} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_K\}$ *of the reduce communication tasks among the processors, that minimizes the send volume load of the maximally loaded processor, i.e.,* $\max_k vol(P_k)$.

A 3-way partition of reduce communication tasks in Figure 4.1b is illustrated in Figure 4.2. In the figure, $\mathcal{R}_k$ is assigned to processor P_k for $k = 1, 2, 3$. Dashed lines represent the reduce communication tasks that the assigned processor *owns*, i.e., they do not incur communication, whereas solid lines represent partial results produced for a reduce communication task that is assigned to another processor, i.e, they do incur communication. In this example partition, $vol(P_2) = \sum_{r_i \in \{results(P_2) - \mathcal{R}_2\}} |partials(P_2, r_i)| = |partials(P_2, r_1)| +$

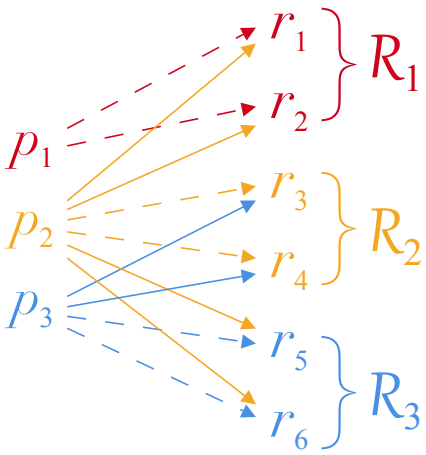

Figure 4.2: A 3-way partition of reduce communication tasks in Figure 4.1b

$|partials(P_2, r_2)| + |partials(P_2, r_5)| + |partials(P_2, r_6)| = 8$. Similarly, $vol(P_1) = 0$, and $vol(P_3) = 4$. Hence, $vol_{total} = 0 + 8 + 4 = 12$. Additionally, $msg(P_2) = |\{\mathcal{R}_1, \mathcal{R}_3\}| = 2$. Similarly, $msg(P_1) = 0$ and $msg(P_3) = 1$. Hence, $msg_{total} = 0 + 2 + 1 = 3$.

4.2 Bipartite Graph Model for Task Assignments

The computational-task-to-processor mapping and reduce communication task assignment problem can be modeled by a bipartite graph $\mathcal{G} = (\mathcal{V}^{\mathcal{C}} \cup \mathcal{V}^{\mathcal{R}}, \mathcal{E})$ where each vertex $v_i^{\mathcal{C}} \in \mathcal{V}^{\mathcal{C}}$ represents a computational task c_i and each vertex $v_j^{\mathcal{R}} : r_j \in \mathcal{R}$ represents a reduce communication task. Vertices $v_i^{\mathcal{C}}$ and $v_j^{\mathcal{R}}$ are connected by an edge $e_{i,j}$ if, as a result of the computational task c_i, at least one partial result for the reduce communication task r_j is produced. Therefore, the neighbors of $v_i^{\mathcal{C}}$ are reduce communication tasks for which partial results are produced as a result of the computational task c_i and the neighbors of $v_j^{\mathcal{R}}$ are computational tasks that produce at least one partial result for that reduce communication task.

Let $partials(v_i^{\mathcal{C}}, v_j^{\mathcal{R}})$ denote the partial results generated as a result of a computational task c_i represented by the vertex $v_i^{\mathcal{C}}$ for a reduce communication task r_j represented by the vertex $v_j^{\mathcal{R}}$. The cost of the edge $e_{i,j}$ connecting $v_i^{\mathcal{C}}$ and $v_j^{\mathcal{R}}$ is

$$c(e_{i,j}) = |partials(v_i^{\mathcal{C}}, v_j^{\mathcal{R}})|. \tag{4.5}$$

The weights of the vertices in $\mathcal{V}^{\mathcal{C}}$ are equal to the total number of partial results generated by the computational task it represents. That is,

$$w(v_i^{\mathcal{C}}) = \sum_{v_j^{\mathcal{R}} \in \mathcal{V}^{\mathcal{R}}} |partials(v_i^{\mathcal{C}}, v_j^{\mathcal{R}})| \text{ where } |partials(v_i^{\mathcal{C}}, v_j^{\mathcal{R}})| \geq 1, \ \forall v_i^{\mathcal{C}} \in \mathcal{V}^{\mathcal{C}}. \tag{4.6}$$

The weights of the vertices in $\mathcal{V}^{\mathcal{R}}$ are set to zero since they do not signify any computation. That is,

$$w(v_j^{\mathcal{R}}) = 0, \ \forall v_j^{\mathcal{R}} \in \mathcal{V}^{\mathcal{R}}. \tag{4.7}$$

Figure 4.3 displays the bipartite graph for the assignment given Figure 4.1. Vertices that represent the computational tasks, i.e., $v_i^{\mathcal{C}} \in V^{\mathcal{C}}$ are represented by circles and vertices that represent the reduce communication tasks, i.e., $v_j^{\mathcal{R}} \in V^{\mathcal{R}}$ are represented by triangles. The numbers on the side of the vertices represent their weights and the numbers above the edges represent their costs.

A K-way partition $\Pi = \Pi^{\mathcal{C}} \cup \Pi^{\mathcal{R}} = \{\mathcal{V}_1^{\mathcal{C}}, \mathcal{V}_2^{\mathcal{C}}, \ldots, \mathcal{V}_K^{\mathcal{C}}\} \cup \{\mathcal{V}_1^{\mathcal{R}}, \mathcal{V}_2^{\mathcal{R}}, \ldots, \mathcal{V}_K^{\mathcal{R}}\}$ induces the computational-task-to-processor mapping by $\Pi^{\mathcal{C}}$ and the reduce communication task assignment by $\Pi^{\mathcal{R}}$. That is, $\Pi^{\mathcal{C}}$ assigns computational tasks to processors and $\Pi^{\mathcal{R}}$ assigns reduce communication tasks to processors that will be responsible for the final results. To obtain this K-way partition, we use METIS: a graph partitioning tool by Karypis and Kumar [8]. To improve the communication balance by minimizing the load of the maximally loaded processor (as defined in 4.1), our proposed models in Chapter 5 replaces $\Pi^{\mathcal{R}}$.

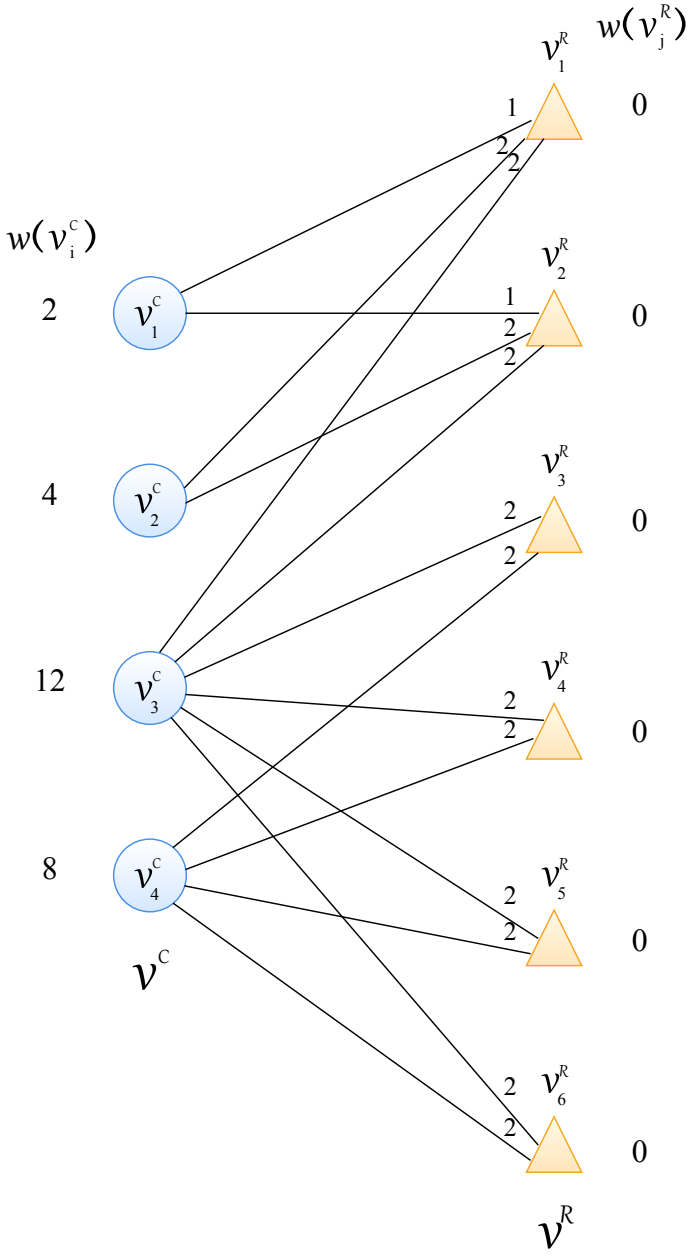

Figure 4.3: The bipartite graph for the assignment given Figure 4.1

4.3 Example Applications

In this section, we discuss outer-product-parallel SpGEMM and outer-product-parallel SpMM as example applications of the bipartite graph model for task assignments.

4.3.1 Outer-Product-Parallel SpGEMM

We consider the outer-product-parallel SpGEMM of the form $C = A \times B$ in a distributed memory setting. In this scheme, there are two types of atomic tasks: the outer product of the xth column of A and the xth row of B (i.e., $a_{*,x} \otimes b_{x,*}$) and the reduction of partial results for nonzero $c_{i,j} \in C$.

The parallelization is achieved through conformable columnwise and rowwise partitioning of the input matrices A and B. The processor that *owns* column x of A, *owns* row x of B as well. The outer product $a_{*,x} \otimes b_{x,*}$ produces a partial result for $c_{i,j} \in C$ if $a_{i,x} \in A$ and $b_{x,j} \in B$. Hence, the outer product $a_{*,x} \otimes b_{x,*}$ corresponds to a computational task.

A and B matrices are conformally partitioned columnwise and rowwise, respectively. However, the conformal partition of A and B does not yield a natural partition of C. We partition the C matrix rowwise; and to obtain this partition, we amalgamate the nonzeros $c_{i,j}$ at the same row of C into $c_{i,*}$. Figure 4.4 represents this amalgamation process. For instance, the computational task $a_{*,x} \otimes b_{x,*}$ produces partial results $c^x_{i,j}$ and $c^x_{i,k}$ for the nonzeros $c_{i,j}$ and $c_{i,k}$, respectively. Since $c_{i,j}$ and $c_{i,k}$ are both in the i-th row of C, we amalgamate them into $c_{i,*}$.

After this amalgamation, we assign the rows of C to processors to make them responsible for the final results of the nonzeros in the rows. In other words, if processor P_k produces a partial result for the row $c_{i,*}$ and if it is not assigned to P_k, P_k needs to send its partial results to the processor responsible for the row's final result. Hence, the row $c_{i,*} \in C$ corresponds to a reduce communication task.

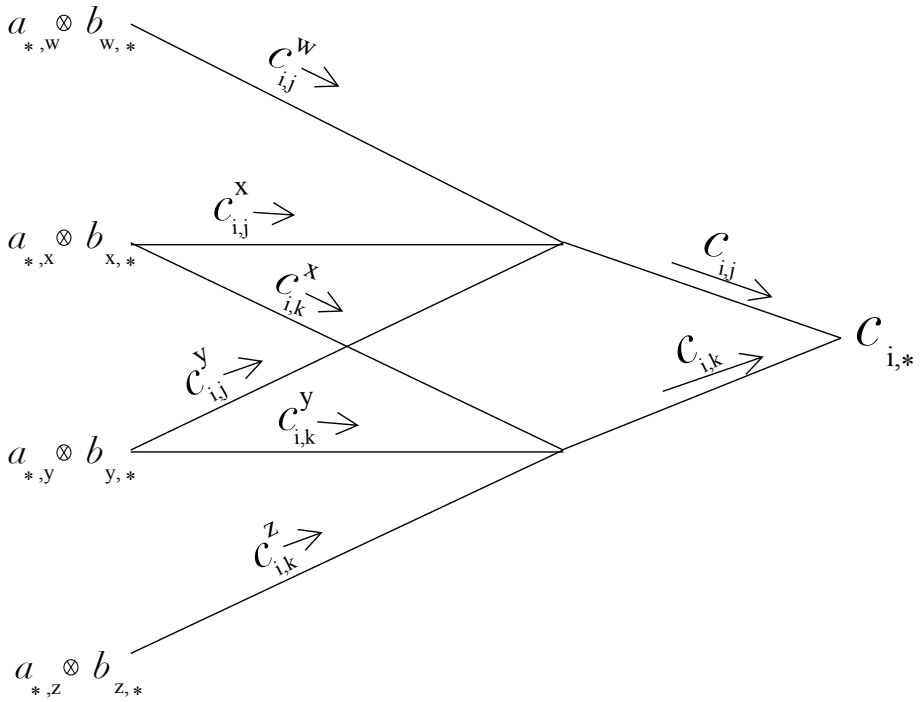

Figure 4.4: Amalgamation of nonzeros of reduce communication tasks

We model outer-product-parallel SpGEMM with a bipartite graph $\mathcal{G} = \{\mathcal{V}^{AB} \cup \mathcal{V}^{C}, \mathcal{E}\}$ where each vertex $v_x^{AB} \in \mathcal{V}^{AB}$ represents the computational task of $a_{*,x} \otimes b_{x,*}$ and each vertex v_i^C represents the reduce communication task of the i-th row of C, i.e., $c_{i,*}$. Note that v_i^C here is different than $v_i^{\mathcal{C}}$ in 4.2 (which represents a computational task c_i).

There is an edge $e_{x,i}$ connecting v_x^{AB} and v_i^C if $a_{*,x} \otimes b_{x,*}$ produces a partial result for $c_{i,*}$, and the number of partial results it produces is assigned as the cost of this edge. That is,

$$c(e_{x,i}) = nnz(b_{x,*}), \tag{4.8}$$

where nnz is the number of nonzeros. The weights of the vertices that represent the computational tasks are equal to the number of multiply-and-add operations performed by that computational task. This $w(v_x^{AB})$ value is also equal to the number of partial results produced by the respective computational task. That is,

$$w(v_x^{AB}) = nnz(a_{*,x}) \times nnz(b_{x,*}), \ \ \forall v_x^{AB} \in \mathcal{V}^{AB}. \tag{4.9}$$

The weights of the vertices that represent the reduce communication tasks are

zero. That is,

$$w(v_i^C) = 0, \ \ \forall v_i^C \in \mathcal{V}^C. \tag{4.10}$$

An example SpGEMM computation is given in Figure 4.5. In the figure, an example computational task, $a_{*,1} \otimes b_{1,*}$ is highlighted in A and B matrices. Also in the figure, reduce communication tasks, rows of C, are colored.

The bipartite graph for the example computation in Figure 4.5 is given in Figure 4.6. Vertices that represent the computational tasks, i.e., $v_x^{AB} \in V^{AB}$ are represented by circles and vertices that represent the reduce communication tasks, i.e., $v_i^C \in V^C$ are represented by triangles. The numbers on the side of the vertices represent their weights and the numbers above the edges represent their costs.

4.3.2 Outer-Product-Parallel SpMM

The parallelization and the bipartite graph representation of outer-product-parallel SpGEMM described in 4.3.1 holds for outer-product-parallel SpMM as well. An example SpMM computation is given in Figure 4.7. In the figure, an example computational task, $a_{*,1} \otimes b_{1,*}$ is highlighted in A and B matrices. Also in the figure, reduce communication tasks, rows of C, are colored.

The bipartite graph for the example computation in Figure 4.7 is given in Figure 4.8. Vertices that represent the computational tasks, i.e., $v_x^{AB} \in V^{AB}$ are represented by circles and vertices that represent the reduce communication tasks, i.e., $v_i^C \in V^C$ are represented by triangles. The numbers on the side of the vertices represent their weights and the numbers above the edges represent their costs.

The main difference between outer-product-parallel SpGEMM and SpMM operations is: each atomic task of outer-product computation in SpMM produces the same number of partial results for a C-matrix row, which is equal to the number of columns in the B matrix. This corresponds to each edge in the bipartite

graph representation of SpMM having the same cost. For example, in Figure 4.8, each edge has a cost of 2 for a 4×2 B matrix; whereas in Figure 4.6, edges have different costs depending on the number of nonzeros in the respective B-matrix rows.

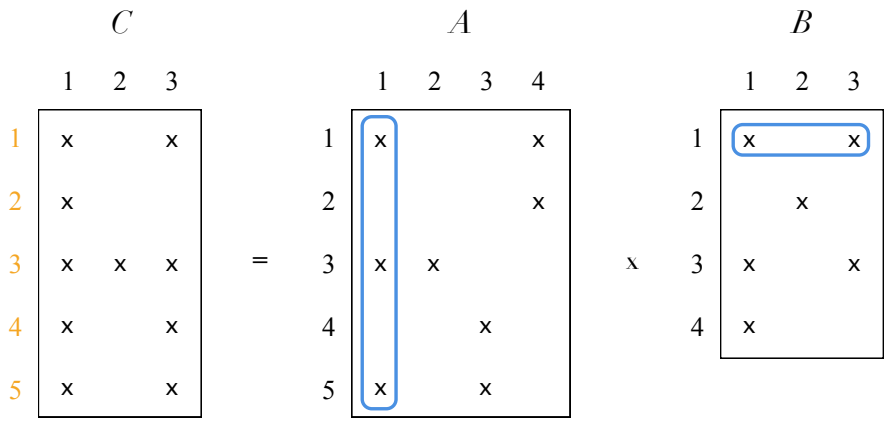

Figure 4.5: An example $C = A \times B$ SpGEMM computation

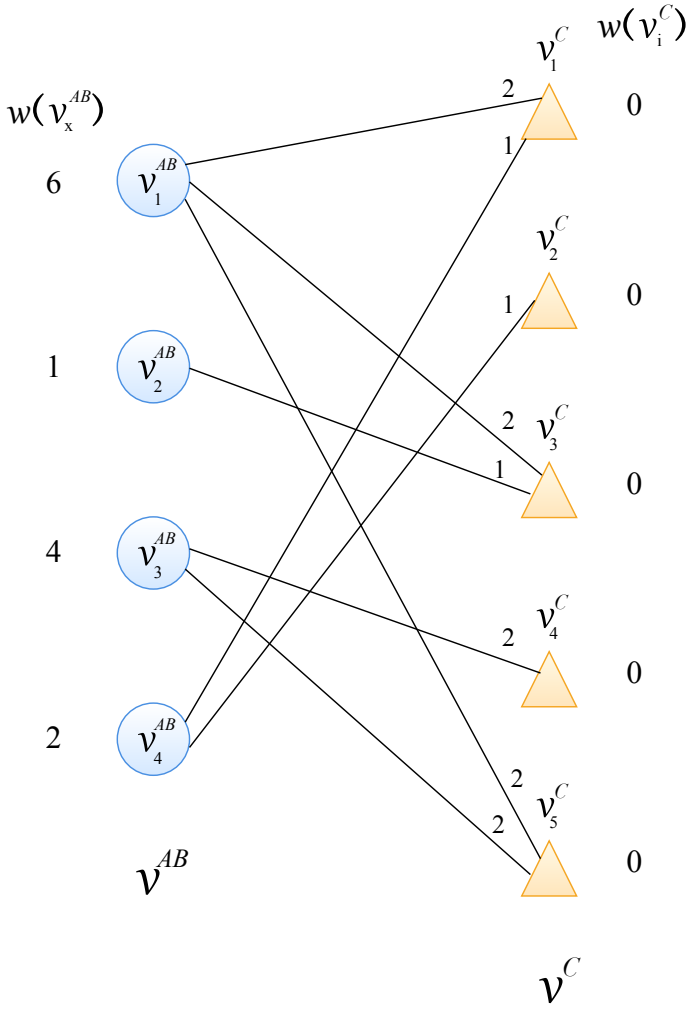

Figure 4.6: The bipartite graph for the SpGEMM computation in Figure 4.5

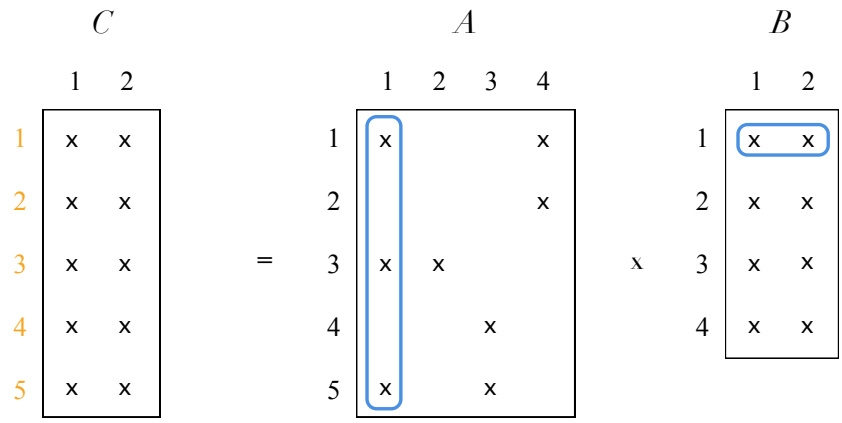

Figure 4.7: An example $C = A \times B$ SpMM computation

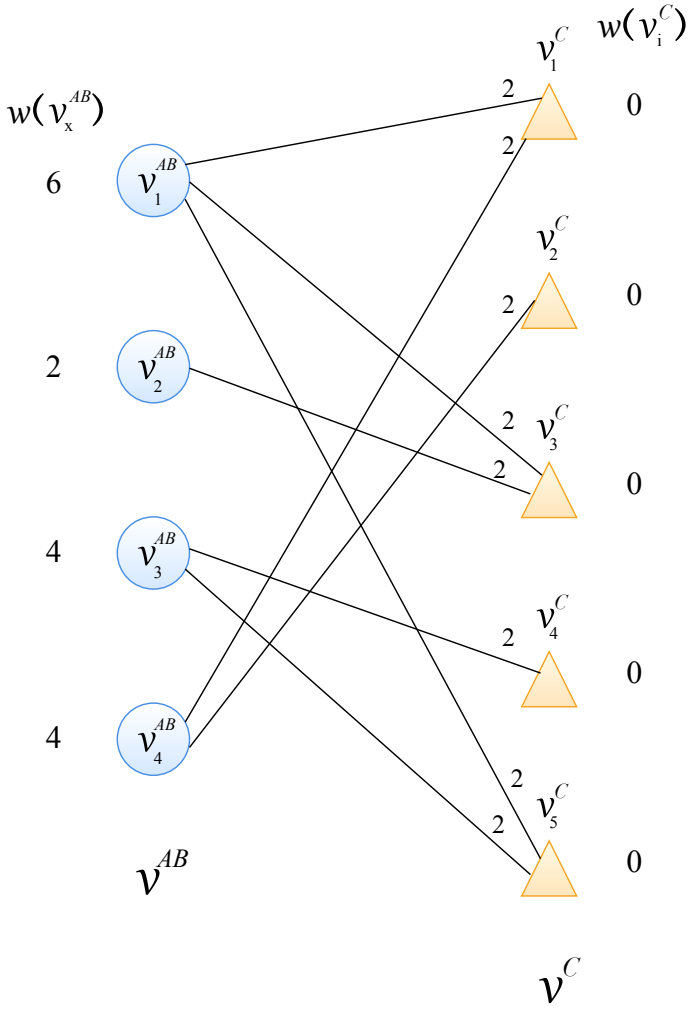

Figure 4.8: The bipartite graph for the SpMM computation in Figure 4.7

Chapter 5

Proposed Methods

We propose a new independent-task-assignment-based algorithm called MaxMinPQ and four new bin-packing-based algorithms for the reduce communication task assignment problem defined in 4.1.

5.1 MaxMinPQ

As mentioned earlier, in the conventional independent task assignment problem, assigning r_i to P_k increases the load of P_k by $x_{i,k}$. However, in the reduce communication task assignment problem, assigning r_i to P_k incurs increases in the send volume loads of the processors in the set $processors(r_i) - \{P_k\}$. To model this reduce communication task assignment problem as a variant of the independent task assignment problem, we propose the following novel scheme.

The proposed formulation requires the initial load vector $\mathbf{l} = (l_k)$ of size K in addition to the $N \times K$ communication volume matrix $\mathbf{X} = (x_{i,k})$. Initially, the load of a processor P_k is set to the total number of partial results generated, i.e., $l_k = \sum_{r_i \in \{results(P_k)\}} |partials(P_k, r_i)|$. As we assign reduce communication tasks to P_k, we subtract the send loads since they no longer incur any communication (see Equation 4.1). To model this, communication volume matrix values $x_{i,k} \in \mathbf{X}$

for the reduce communication task r_i and the processor P_k are set to $-1 \times |partials(P_k, r_i)|$.

The proposed MaxMinPQ algorithm is given in Algorithm 1.

Algorithm 1: MaxMinPQ(**l**, **X**, K, N)

```
1 for k ← 1 to K do
2 |   PQ_k ← Build(k, X)
  |   // PQ_k contains records of ⟨x_{i,k}, i⟩
3 for i ← 1 to N do
4 |   ⟨i', k'⟩ ← MaxMinPQSelect(PQ, l, K)
5 |   A[i'] ← k'
6 |   l_{k'} ← l_{k'} + x_{i',k'}
7 |   for k ← 1 to K do
8 |   |   Delete(PQ_k, i')
9 return A
```

Algorithm 2: MaxMinPQSelect(PQ, **l**, K)

```
1 max ← −∞
2 for k ← 1 to K do
3 |   ⟨x_{i,k}, i⟩ ← Min(PQ_k)
4 |   if l_k + x_{i,k} > max then
5 |   |   max ← l_k + x_{i,k}
6 |   |   i' ← i
7 |   |   k' ← k
8 return ⟨i', k'⟩
```

In MaxMinPQ; the negative send volumes for the reduce communication tasks associated with each processor are separately maintained using priority queues. That is, each processor P_k has a priority queue PQ_k to maintain the negative send volumes for the reduce communication tasks for which P_k produces partial results. More specifically, each reduce communication task r_i is maintained in the priority queue of each processor that produces partial results for that reduce communication task, keyed by their $x_{i,k}$ values.

Each priority queue is built using the Build operation (lines 1-2) which stores

the negative send volumes for the reduce communication task for which the processor produces partial results. The following loop (lines 3-8) performs N iterations, assigning a reduce communication task to a processor using the MaxMinPQSelect function (Algorithm 2) at each iteration.

The MaxMinPQSelect function invokes a Min operation on each priority queue PQ_k to find a candidate task for processor P_k. The candidate task r_i selected for processor P_k is effectively the task that will decrease the current send load of P_k, i.e., l_k, by the largest amount if r_i is assigned to P_k. For each processor P_k, the negative send load of the candidate task r_i is added to l_k to consider the updated load for P_k if r_i is assigned to P_k. A running-max operation over these values (lines 4-7) gives the assignment $\langle i', k' \rangle$ for the current iteration. This selection policy tries to find assignments that yield larger decreases of the load of the current maximally loaded processor in earlier iterations.

After the selected reduce communication is assigned to the selected processor in Algorithm 1 (line 5), and the current load of the selected processor is updated (line 6); the assigned reduce communication task is deleted from all priority queues (lines 7-8). For simplicity, we represent a deletion loop that iterates over all processors in Algorithm 1 (line 7); however, in our implementation, we maintain a list of processors that produce partial results for a reduce communication task to iterate over only those processors for deletion.

5.2 Bin Packing Algorithms

We also propose four bin-packing-based algorithms for the reduce communication task assignment problem. These algorithms utilize the heuristics used in solving the K-feasible bin packing problem [9]. These algorithms differ by the order of the reduce task assignments, as well as the best-fit criterion used in the assignments.

Two different task assignment orders investigated are decreasing order of maximum communication volume (MaxVol) and total communication volume (TotVol)

incurred by the reduce communication task. Two different best-fit criteria investigated are MaxMin and Sum-of-Squares (SumSqrs).

5.2.1 BP-MaxVol-MaxMin

The proposed bin packing variation BP-MaxVol-MaxMin is given in Algorithm 3. The K bins correspond to the send volume loads of the K processors. The algorithm takes a communication volume matrix $\mathbf{X} = (x_{i,k})$ where $x_{i,k}$ for the reduce communication task r_i and the processor P_k is $|partials(P_k, r_i)|$.

Algorithm 3: BP-MaxVol-MaxMin($\mathbf{X}$, K)

```
1  for k ← 1 to K do
2  |  l_k ← 0
3  foreach reduce task r_i in decreasing order of MaxVol do
4  |  k' ← BP-MaxMinSelect(X, r_i, l, K)
5  |  A[r_i] ← k'
6  |  l_k' ← l_k' − x_i,k'
7  return A
```

Algorithm 4: BP-MaxMinSelect($\mathbf{X}$, r_i, $\mathbf{l}$, K)

```
1  min ← ∞
2  for k ← 1 to K do
3  |  if x_i,k > 0 then
4  |  |  l_k ← l_k + x_i,k
5  for k ← 1 to K do
6  |  if x_i,k > 0 then
7  |  |  l_k ← l_k − x_i,k // Update the load as if r_i is assigned to P_k
8  |  |  max ← Max(l)
9  |  |  l_k ← l_k + x_i,k // Update the load to the original value
10 |  |  if max < min then
11 |  |  |  min ← max
12 |  |  |  k' ← k
13 return k'
```

Loads of the K processors are initialized as zero (lines 1-2). For the sake of simplicity of representation, the rows of $\mathbf{X}$ are assumed to be sorted according

to their maximum volumes (MaxVol) in decreasing order. Due to this order, the reduce communication tasks with higher maximum volumes ($max(x_{i,*})$) are assigned in earlier iterations. The following loop (lines 3-6) assigns each reduce communication task to a processor using the BP-MaxMinSelect function (Algorithm 4).

The first loop in BP-MaxMinSelect (lines 2-4) updates the send loads of the processors that produce partial results for the current reduce communication task, i.e., $x_{i,k} > 0$ for r_i and P_k; as if all these processors need to send their partial results to some other processor. The second loop is represented to iterate over all processors and consider the processors producing partial results (lines 5-6) for simplicity; however, in our implementation, we maintain a list of processors that produce partial results for a reduce communication r_i to iterate over only those processors. The second loop (lines 5-12) considers the possible scenarios where the current task r_i has been assigned to each of the processors that produce partial results to r_i. That is, it models what the send loads would be if it assigned r_i to processor P_k, and repeats this for every processor that produce partial results for r_i. It first updates the load of P_k (line 7) as if r_i is assigned to P_k. Then it calculates the maximum send load $max(\mathbf{l})$ if r_i was assigned to P_k (line 8), and reverses the load update (line 9) since the assignment has not yet been finalized. A running-min operation over these K max values constitutes the MaxMin selection criterion. This selection criterion tries to minimize the load of the maximally loaded processor by selecting an assignment that yields the minimum max send volume.

Finally, in Algorithm 3, the reduce communication task is assigned to the selected processor (line 5) and the load of the selected processor is updated (line 6).

5.2.2 BP-TotVol-MaxMin

The proposed bin packing variation BP-TotVol-MaxMin is given in Algorithm 5. BP-TotVol-MaxMin differs from BP-MaxVol-MaxMin only in the order

of assignment. Here, the rows of the $\mathbf{X}$ matrix are assumed to be sorted according to their total volumes (TotVol) in decreasing order. Due to this order, the reduce communication tasks with higher total volumes ($sum(x_{i,*})$) are assigned in earlier iterations.

Algorithm 5: BP-TotVol-MaxMin($\mathbf{X}$, K)

```
1 for k ← 1 to K do
2 |   l_k ← 0
3 foreach reduce task r_i in decreasing order of TotVol do
4 |   k' ← BP-MaxMinSelect(X, r_i, l, K)
5 |   A[r_i] ← k'
6 |   l_k' ← l_k' − x_i,k'
7 return A
```

5.2.3 BP-MaxVol-SumSqrs

The Sum-of-Squares Algorithm for bin packing [7] is adapted here to obtain BP-MaxVol-SumSqrs. BP-MaxVol-SumSqrs (Algorithm 6) differs from BP-MaxVol-MaxMin in the best-fit criterion. Here, the algorithm tries to select assignments that yield minimum sum-of-squares of the send loads (line 8 in Algorithm 7).

Algorithm 6: BP-MaxVol-SumSqrs($\mathbf{X}$, K)

```
1 for k ← 1 to K do
2 |   l_k ← 0
3 foreach reduce task r_i in decreasing order of MaxVol do
4 |   k' ← BP-SumSqrsSelect(X, r_i, l, K)
5 |   A[r_i] ← k'
6 |   l_k' ← l_k' − x_i,k'
7 return A
```

Algorithm 7: BP-SumSqrsSelect($\mathbf{X}$, r_i, $\mathbf{l}$, K)

```
1  min ← ∞
2  for k ← 1 to K do
3  |  if x_{i,k} > 0 then
4  |  |  l_k ← l_k + x_{i,k}
5  for k ← 1 to K do
6  |  if x_{i,k} > 0 then
7  |  |  l_k ← l_k − x_{i,k} // Update the load as if r_i is assigned to P_k
8  |  |  sumsqrs ← SumSqrs(l)
9  |  |  l_k ← l_k + x_{i,k} // Update the load to the original value
10 |  |  if sumsqrs < min then
11 |  |  |  min ← sumsqrs
12 |  |  |  k' ← k
13 return k'
```

5.2.4 BP-TotVol-SumSqrs

The proposed bin packing variation BP-TotVol-SumSqrs is given in Algorithm 8. BP-TotVol-SumSqrs differs from BP-MaxVol-SumSqrs only in the order of assignment. Here, the rows of the $\mathbf{X}$ matrix are assumed to be sorted according to their total volumes (TotVol) in decreasing order. Due to this order, the reduce communication tasks with higher total volumes ($sum(x_{i,*})$) are assigned in earlier iterations.

Algorithm 8: BP-TotVol-SumSqrs($\mathbf{X}$, K)

```
1 for k ← 1 to K do
2 |  l_k ← 0
3 foreach reduce task r_i in decreasing order of TotVol do
4 |  k' ← BP-SumSqrsSelect(X, r_i, l, K)
5 |  A[r_i] ← k'
6 |  l_{k'} ← l_{k'} − x_{i,k'}
7 return A
```

Chapter 6

Experiments and Results

We validate our proposed methods for the reduce communication task assignment problem on outer-product-parallel SpGEMM and SpMM kernels. In this section, we discuss our experiment setup, our test data, and our results.

6.1 Setup and Data

To obtain a K-way partition for the bipartite graph model (as in Chapter 4.2), we use Metis [8] with the partitioning objective of minimizing the edgecut. We set K to 1024, and the maximum load imbalance to 10%.

We tested the validity of our proposed algorithms on a set of sparse test matrices arising from real-world applications. These matrices are from the University of Florida Sparse Matrix Collection [10]. Table 6.1 presents the properties of these test matrices in alphabetical order (their names are under the "Matrix" column). Their number of rows, columns, and nonzeros are given in "rows", "cols", and "nnz's" columns, respectively. Also in the table; their minimum, average, and maximum nonzero counts of rows and columns are given under "nnz's in a row" and "nnz's in a col", respectively.

Our SpMM experiments are in the form of $C = A \times B$ where the A matrix is from Table 6.1, and the B matrix is a generated dense matrix. The generated B matrix's number of rows is equal to the number of columns of A, and its number of columns is set to 16.

6.2 Results

In our experiments, the results produced by the reduce communication task assignment by METIS are considered as the baseline results. We report the following four metrics for communication performance comparison: maximum communication volume (send load of the maximally loaded processor), total communication volume (see Equation 4.2), maximum number of messages, and total number of messages (see Equation 4.4) for the baseline and our algorithms.

Results for the outer-product-parallel SpMM problem of the form $C = A \times B$ is given in Table 6.2. Matrices listed under the "Matrix" column are the A matrices and the B matrices are generated dense matrices. In Table 6.2, we report the actual values for the baseline and report normalized values for the five proposed methods with respect to the baseline. Also in Table 6.2, we report the geometric mean ("Geomean" row) for the normalized metrics of our proposed methods for the 15 SpMM instances.

From Table 6.2; we observe that, on average, MAXMINPQ improves the maximum volume by 12%, BP-MAXVOL-SUMSQRS improves the maximum volume by 11%, and BP-TOTVOL-SUMSQRS improves the maximum volume by 8%. For this problem, on average, BP-MAXVOL-MAXMIN and BP-TOTVOL-MAXMIN seem to not improve maximum volume. In the bin-packing-based methods, the best-fit criterion SumSqrs performs considerably better than MaxMin. For the assignment order, decreasing order of MaxVol produces slightly better results than TotVol. Based on the average maximum volume metric values of the five proposed methods; we can remark that MAXMINPQ is the best candidate method for this application.

Results for the outer-product-parallel SpGEMM problem are given in Table 6.3. 10 of these SpGEMM instances are in the form $C = A \times A$, and 5 of them are in the form $C = A \times A^T$ where A matrices are the matrices whose names are given under "Matrix" column. As in the SpMM problem; actual metric values are reported for the baseline in Table 6.3, and the metric values for the proposed methods are normalized with respect to baseline. We report the geometric mean ("Geomean" row) for the normalized metrics of our proposed methods for the 15 SpGEMM instances.

From Table 6.3; we observe that, on average, MaxMinPQ improves maximum volume by 20%, BP-MaxVol-MaxMin by 12%, BP-TotVol-MaxMin by 10%, BP-MaxVol-SumSqrs by 22%, and BP-TotVol-SumSqrs by 23%. Decreasing MaxVol and TotVol assignment orders produce comparable performance. On the other hand, for the best-fit criterion, SumSqrs perform significantly better than MaxMin. Based on the average maximum volume metric values of the five proposed methods, we can remark that BP-TotVol-SumSqrs is the best candidate method for this application.

Table 6.1: Properties of the test matrices

Matrix	Number of rows	Number of cols	Number of nnz's	nnz's in a row min	nnz's in a row avg	nnz's in a row max	nnz's in a col min	nnz's in a col avg	nnz's in a col max
bcsstk32	44,609	44,609	2,014,701	2	45	216	2	45	216
bmwcra_1	148,770	148,770	10,644,002	24	72	351	24	72	351
brack2	62,631	62,631	733,118	3	12	32	3	12	32
fe_rotor	99,617	99,617	1,324,862	5	13	125	5	13	125
finance256	37,376	37,376	298,496	3	8	55	3	8	55
Ga19As19H42	133,123	133,123	8,884,839	15	67	697	15	67	697
Ga41As41H72	268,096	268,096	18,488,476	18	69	702	18	69	702
ia2010	216,007	216,007	1,021,170	1	5	49	1	5	49
net4-1	88,343	88,343	2,441,727	2	28	4,791	2	28	4,791
pkustk03	63,336	63,336	3,130,416	12	49	90	12	49	90
sgpf5y6	246,077	312,540	831,976	2	3	61	1	3	12
srb1	54,924	54,924	2,962,152	36	54	270	36	54	270
watson_1	201,155	386,992	1,055,093	1	5	93	1	3	9
watson_2	352,013	677,224	1,846,391	1	5	93	1	3	15
xenon1	48,600	48,600	1,181,120	1	24	27	1	24	27

Table 6.2: Outer-product-parallel SpMM results for $K = 1024$ processors

	Baseline (actual values)				**Proposed Methods (normalized w.r.t. baseline)**																			
	Metis				**MaxMinPQ**				**BP-MaxVol-MaxMin**				**BP-TotVol-MaxMin**				**BP-MaxVol-SumSqrs**				**BP-TotVol-SumSqrs**			
	volume		**message**		**volume**		**message**		**volume**		**message**		**volume**		**message**		**volume**		**message**		**volume**		**message**	
Matrix	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**	**max**	**avg**
bcsstk32	25,056	14,333	24	8.29	0.87	1.18	1.17	1.26	0.96	1.40	1.04	1.29	1.08	1.45	1.17	1.18	0.96	1.01	0.96	1.00	0.95	1.02	1.00	1.03
bmwcra_1	93,632	67,729	33	13.58	0.97	1.19	1.15	1.23	1.22	1.54	1.09	1.31	1.31	1.54	1.00	1.17	0.98	1.00	0.97	0.98	1.07	1.02	1.00	1.02
brack2	5,616	3,535	37	10.71	0.83	1.15	0.97	1.02	1.16	1.69	0.92	1.12	1.18	1.67	0.97	1.12	0.88	1.00	0.92	0.99	0.87	1.01	0.92	0.98
fe_rotor	8,400	5,793	42	14.06	0.94	1.14	1.05	1.04	1.38	1.73	0.98	1.14	1.51	1.72	1.00	1.11	0.91	1.00	0.98	0.99	0.98	1.01	0.98	0.98
finance256	3,120	1,493	26	13.90	0.71	1.30	0.85	0.93	1.03	1.71	0.92	1.03	1.07	1.71	1.00	0.92	0.89	1.01	1.00	0.98	0.95	1.02	0.88	0.97
Ga19As19H42	34,176	16,178	48	23.88	0.84	1.07	1.19	0.70	0.87	1.19	1.25	0.65	0.87	1.19	1.15	0.62	0.90	1.00	1.06	0.83	0.91	1.01	1.06	0.83
Ga41As41H72	62,416	32,050	48	17.27	0.93	1.08	1.29	0.90	0.93	1.17	1.23	0.84	0.94	1.17	1.23	0.81	0.98	1.00	1.21	0.96	0.99	1.01	1.17	0.96
ia2010	1,760	1,016	12	5.81	0.84	1.17	1.00	0.99	1.35	1.98	0.92	0.97	1.35	1.96	1.00	0.97	0.84	1.01	1.00	1.00	0.85	1.01	1.00	1.00
net4-1	77,328	15,783	193	16.08	0.73	2.08	0.30	0.65	0.65	2.04	0.82	1.32	0.66	2.08	0.97	1.33	1.00	0.98	0.96	0.98	1.00	0.98	0.95	0.97
pkustk03	28,256	19,142	18	7.12	0.92	1.18	1.44	1.25	1.12	1.43	1.33	1.25	1.29	1.47	1.22	1.21	1.00	1.01	1.00	0.97	1.03	1.02	1.00	1.03
sgpf5y6	5,968	2,004	75	12.58	0.85	1.70	1.41	1.00	0.73	1.67	1.33	0.99	0.78	1.74	1.51	0.99	0.65	1.09	1.01	0.93	0.65	1.09	1.04	0.93
srb1	28,416	19,750	18	6.76	0.89	1.16	1.00	1.22	1.08	1.40	1.00	1.25	1.30	1.48	1.00	0.97	0.99	1.01	0.94	0.99	1.05	1.04	1.00	1.00
watson_1	3,328	824	42	5.70	0.85	2.11	0.81	1.00	0.97	2.71	0.69	0.96	0.91	2.68	0.64	0.96	0.67	1.18	0.83	1.01	0.66	1.18	0.83	1.01
watson_2	2,784	981	33	6.01	1.08	1.57	1.06	1.01	1.25	2.43	0.85	0.94	1.26	2.34	0.82	0.93	0.80	1.06	0.79	1.00	0.81	1.06	0.79	1.00
xenon1	8,864	6,404	18	10.05	0.98	1.19	1.06	1.15	1.34	1.66	1.22	1.24	1.46	1.74	1.00	1.05	0.99	1.00	1.00	1.00	1.12	1.04	1.00	1.01
Geomean					**0.88**	1.32	1.00	1.01	**1.05**	1.67	1.02	1.07	**1.10**	1.69	1.03	1.01	**0.89**	1.02	0.97	0.97	**0.92**	1.03	0.97	0.98

Table 6.3: Outer-product-parallel SpGEMM results for $K = 1024$ processors

	Baseline (actual values)				Proposed Methods (normalized w.r.t. baseline)																			
	Metis				MaxMinPQ				BP-MaxVol-MaxMin				BP-TotVol-MaxMin				BP-MaxVol-SumSqrs				BP-TotVol-SumSqrs			
	volume		message		volume		message		volume		message		volume		message		volume		message		volume		message	
Matrix	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg	max	avg
$C = A \times A$																								
bcsstk32	108,300	49,115	22	8.53	0.74	1.16	1.18	1.27	0.86	1.34	1.36	1.32	0.81	1.40	1.23	1.17	0.88	1.01	1.00	1.01	0.88	1.02	1.09	1.03
bmwcra_1	653,137	333,092	30	13.46	0.72	1.17	1.07	1.21	0.87	1.50	1.20	1.39	0.93	1.52	1.13	1.17	0.84	1.00	0.97	0.98	0.83	1.02	0.97	1.02
brack2	5,349	3,270	40	10.76	0.85	1.14	0.88	1.04	1.09	1.62	0.93	1.14	1.09	1.60	0.85	1.11	0.87	1.00	0.88	0.98	0.84	1.01	0.83	0.99
fe_rotor	11,079	5,792	41	14.21	0.70	1.14	0.95	1.05	0.92	1.63	1.29	1.16	0.98	1.62	1.02	1.10	0.69	1.00	0.93	0.99	0.68	1.01	0.93	0.99
finance256	3,503	1,879	24	14.51	0.66	1.00	0.92	0.95	0.78	1.22	1.04	0.99	0.79	1.22	0.92	0.89	0.75	0.93	1.08	0.93	0.74	0.93	1.00	0.93
Ga19As19H42	280,497	174,117	279	44.19	0.85	1.00	1.05	0.45	0.86	1.00	1.03	0.46	0.87	1.01	1.04	0.44	0.90	0.99	1.16	0.59	0.90	0.99	1.16	0.59
Ga41As41H72	560,886	363,233	227	28.22	0.86	1.00	0.99	0.60	0.89	1.01	1.00	0.62	0.89	1.01	1.00	0.61	0.91	0.99	1.00	0.83	0.92	0.99	1.00	0.83
net4-1	22,953,681	611,893	131	6.55	0.61	0.75	0.24	0.59	0.62	0.75	0.27	0.68	0.63	0.75	0.27	0.77	0.61	0.74	0.37	0.85	0.64	0.74	0.36	0.83
pkustk03	104,940	65,244	18	7.08	0.87	1.17	1.33	1.27	1.01	1.41	1.22	1.27	1.18	1.44	1.00	1.16	0.94	1.01	1.00	0.98	0.91	1.02	1.06	1.02
srb1	155,286	67,595	15	6.73	0.69	1.15	1.00	1.20	0.87	1.40	1.27	1.21	0.81	1.46	0.87	0.96	0.90	1.01	1.00	1.00	0.75	1.04	0.87	1.00
$C = A \times A^T$																								
ia2010	961	391	13	5.91	0.94	1.27	1.08	1.00	0.93	1.83	0.85	0.96	0.94	1.82	0.85	0.96	0.68	1.01	1.00	1.00	0.71	1.01	1.00	1.00
sgpf5y6	1,459	543	77	13.79	0.77	1.15	0.96	1.03	0.65	1.52	0.96	1.02	0.67	1.53	1.01	1.01	0.58	1.08	0.86	0.95	0.59	1.07	0.84	0.95
watson_1	616	149	42	6.25	0.93	1.72	1.02	1.03	0.81	2.38	0.90	0.97	0.80	2.39	0.86	0.96	0.72	1.17	0.83	1.00	0.73	1.17	0.83	0.99
watson_2	509	175	38	6.56	0.91	1.37	1.03	0.98	1.01	2.18	0.82	0.93	1.01	2.15	0.82	0.93	0.65	1.07	0.84	1.00	0.63	1.07	0.87	1.00
xenon1	14,017	10,093	17	10.18	0.97	1.18	1.18	1.15	1.30	1.65	1.24	1.24	1.45	1.72	1.06	1.12	0.96	1.00	0.94	1.00	0.98	1.02	1.00	1.00
Geomean					**0.80**	1.14	0.94	0.95	**0.88**	1.44	0.97	0.98	**0.90**	1.45	0.89	0.93	**0.78**	1.00	0.90	0.93	**0.77**	1.00	0.89	0.94

Chapter 7

Conclusion and Future Work

We focused on the problem in which given a computational-task-to-processor mapping in a parallel system with K processors, the goal is to assign reduce communication tasks to processors in a way that minimizes the communication volume load of the maximally loaded processor. We proposed one new independent-task-assignment-based algorithm and four new bin-packing-based algorithms for the defined problem. We showed the validity of our algorithms on outer-product-parallel SpGEMM and SpMM kernels in terms of communication volume metrics. We will demonstrate the performance of the proposed algorithms in terms of actual parallel runtimes of these two kernels.

As future work, we are planning to utilize the proposed communication volume balancing algorithms for improving the performance of the reduce phase of the communication involved in parallel Stochastic Gradient Descent algorithm used in solving the matrix completion problem with large latent factor sizes.

Bibliography

[1] M. O. Karsavuran, S. Acer, and C. Aykanat, "Reduce operations: Send volume balancing while minimizing latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1461–1473, 2020.

[2] K. Akbudak, O. Selvitopi, and C. Aykanat, "Partitioning models for scaling parallel sparse matrix-matrix multiplication," *ACM Trans. Parallel Comput.*, vol. 4, Jan. 2018.

[3] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, p. 280–289, Apr. 1977.

[4] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810 – 837, 2001.

[5] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107 – 131, 1999.

[6] E. K. Tabak, B. B. Cambazoglu, and C. Aykanat, "Improving the performance of independenttask assignment heuristics minmin,maxmin and sufferage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, p. 1244–1256, May 2014.

[7] J. Csirik, D. S. Johnson, C. Mathieu, P. W. Shor, and R. Weber, "A self organizing bin packing heuristic," in *ALENEX*, 1999.

[8] G. Karypis and V. Kumar, "Multilevel k-way partitioning scheme for irregular graphs," *J. Parallel Distrib. Comput.*, vol. 48, p. 96–129, Jan. 1998.

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

[10] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Trans. Math. Softw.*, vol. 38, Dec. 2011.