MINIMIZING COMMUNICATION THROUGH COMPUTATIONAL REDUNDANCY IN PARALLEL ITERATIVE SOLVERS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

FAHREDDİN ŞÜKRÜ TORUN

September, 2011

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

Prof. Dr. Cevdet Aykanat(Advisor)

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

Assoc. Prof. Dr. Hakan Ferhatosmanoğlu

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

Assoc. Prof. Dr. Oya Ekin Karaşan

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of the Graduate School

ABSTRACT

MINIMIZING COMMUNICATION THROUGH COMPUTATIONAL REDUNDANCY IN PARALLEL ITERATIVE SOLVERS

FAHREDDİN ŞÜKRÜ TORUN
M.S. in Computer Engineering
Supervisor: Prof. Dr. Cevdet Aykanat
September, 2011

Sparse matrix vector multiplication (SpMxV) of the form y = Ax is a kernel operation in iterative linear solvers used in scientific applications. In these solvers, the SpMxV operation is performed repeatedly with the same sparse matrix through iterations until convergence. Depending on the matrix and its decomposition, parallel SpMxV operation necessitates communication among processors in the parallel environment. The communication can be reduced by intelligent decomposition. However, we can further decrease the communication through data replication and redundant computation. The communication occurs due to the transfer of x-vector entries in row-parallel SpMxV computation. The input vector x of the next iteration is computed from the output vector of the current iteration through linear vector operations. Hence, a processor may compute a y-vector entry redundantly, which leads to a x-vector entry in the following iteration, instead of receiving that x-vector entry from another processor. Thus, redundant computation of that y-vector entry may lead to reduction in communication.

In this thesis, we devise a directed-graph-based model that correctly captures the computation and communication pattern for above-mentioned iterative solvers. Moreover, we formulate the communication minimization by utilizing redundant computation of y-vector entries as a combinatorial problem on this directed graph model. We propose two heuristics to solve this combinatorial problem. Experimental results indicate that the communication reducing strategy by redundantly computing is promising.

Keywords: Sparse matrix vector multiplication, sparse matrix, parallel, replication, iterative solvers.

ÖZET

PARALEL YİNELEMELİ ÇÖZÜMLEYİCİLERDE FAZLA HESAPLAMA İLE HABERLEŞME AZALTIMI

FAHREDDİN ŞÜKRÜ TORUN
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Eylül, 2011

y=Ax biçimindeki seyrek matris-vektör çarpımı (SMxV) bilimsel uygulamalarda yinelemeli doğrusal denklem çözümleyicilerinde kullanılan bir çekirdek operasyondur. Bu çözümleyicilerde, yinelemeler vasıtasıyla yakınsayıncaya kadar aynı seyrek matris ile SMxV operasyonu tekrarlanarak uygulanır. Paralel ortamda paralel SMxV operasyonu matrise ve onun ayrışımına göre işlemciler arasında haberleşmeye ihtiyaç duyar. Bu haberleşme akıllı ayrışımlar ile azaltılabilinir. Fakat, biz veri replikasyonu ve fazla hesaplama ile bu haberleşmeyi daha da fazla azaltabiliriz. Satır-paralel SMxV hesaplamada bu haberleşme x-vektör elemanlarının transferi yüzünden oluşur. Bir sonraki yinelemenin girdi vektörü x, bazı doğrusal operasyonlar vasıtasıyla yürürlükteki yinelemenin çikti vektörü y ile hesaplanır. Bundan dolayı, bir işlemci başka bir işlemciden bir x-vektör elemanı almak yerine fazla bir y vektör elemanını, ki bu y vektör elemanı bir sonraki yinelemenin x vektör elemanına öncülük eder, hesaplayabilir. Böylece, fazla y vektör elemanı hesaplamak haberleşmenin azalmasına yol açabilir.

Bu tezde, biz yukarıda bahsedilen yinelemeli denklem çözümlayiciler için hesaplama ve haberleşme desenini doğru yakalayan yönlü çizge tabanlı model tasarladık. Bundan başka, biz fazla y-vektör elemanı hesaplaması sebebiyle haberleşme azalışını yönlü çizge modeli uzerinde bir kombinatoriyal problem olarak formülledik. Biz bu kombinatoriyal problemi çözmek için iki tane buluşsal yöntem önerdik. Deneysel sonuçlar göstermektedir ki fazla hesaplama yaparak haberleşme azaltma stratejisi gelecek vaat etmektedir.

Anahtar sözcükler: Seyrek matris vektör çarpımı, seyrek matris, paralel, replikasyon, haberlesme azaltımı, yinelemeli çözümleyiciler.

Acknowledgement

I would like to express my deepest gratitude to my supervisor Prof. Cevdet Aykanat for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis. I thank Assoc. Prof. Dr. Hakan Ferhatosmanoğlu and Assoc. Prof. Dr. Oya Ekin Karaşan for accepting to participate in my thesis committee.

I am grateful to my family for their infinite moral support and help. I owe special thanks to my friend Enver Kayaaslan. I am also grateful to my colleagues M. Abdullah Bülbül, Ahmet Çagrı Şimşek, Kadir Akbudak, Ata Türk, Emre Varol, Seher Acer, Mustafa Korkmaz, Zeynep Saka, Mücahid Kutlu, and my other friends whose names I have not mentioned.

Finally, I thank TÜBİTAK for supporting grant throughout my master program.

To my family...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Matrix-vector multiplication is an essential operation in many algorithms used in scientific applications. Repeated sparse matrix-vector multiplication (SpMxV) that involves the same sparse, large, square and rectangular matrix is a kernel operation in iterative solvers. Therefore, SpMxV plays a crucial role in scientific computing society. In iterative solvers two fundamental operations are repeatedly performed at each iteration. These operations are linear operations on dense vectors and the SpMxV operation of the form $y \leftarrow Ax$, where x and y are dense vectors, and A is an M x N sparse matrix.

In iterative methods, SpMxV is the fundamental operation that is performed at each iteration to solve linear systems. Google‘s PageRank algorithm [38], image deblurring [37], and linear system solutions [6] are some examples for application areas of iterative methods. The performance of the mentioned methods highly depends on the performance of the SpMxV operation. If the efficiency of SpMxV is improved, efficiency of these methods will be improved eventually. Due to the efficiency concerns, SpMxV is performed in a parallel fashion. Depending on the matrix and its partitioning, parallel SpMxV necessitates communication among processors in the parallel system.

The SpMxV parallelization problem is clearly defined in [51]. The distribution of computational load corresponds to the distribution of nonzeros of the matrix among the processors under the owner compute rule [31]. The distribution of nonzeros can be one dimensional(1D) or two dimensional(2D) [11, 19, 20, 27]. In 1D partitioning, each processor is ensured to own either whole rows or whole columns. In 1D row-wise distribution, rows of the input matrix are distributed among processors, the processors hold nonzeros in a row. Similarly, in 1D column-wise distribution, columns of the input matrix are distributed among processors. However, 2D partitioning techniques do not maintain any row or column integrity. Nonzeros of the input matrix can be assigned to processors without regard to the row or column coherency.

Parallel SpMxV dominates the running time of diverse applications in scientific computing. The performance of parallel SpMxV highly depends on the computational imbalance and communication overheads. The problem of communication overhead in the system stems from the task dependency and data partitioning of the input matrix. There are several graph or hyper-graph partitioning based models for sparse matrix partitioning in order to minimize task interactions among parts/processors while also aiming load balance on the parts/processors. In terms of parallel SpMxV the task interaction refers to message volume that is sent or received by processors. In this work, we further reduce the total message volume in parallel SpMxV by performing some redundant work by means of replication.

In the literature, replication is a widely used technique for various purposes in computer science, such as improving reliability, fault tolerance, accessibility, reducing processing, and communication cost. Many disciplines in computer science benefit from replication scheme through the mentioned methods. The detailed discussion is provided in Chapter 7.

We propose a new model to minimize the communication overhead of SpMxV. The proposed model employs the replication technique which is used in many areas of scientific computing. Besides, our replication methodology exploits the redundant work technique. It is not guaranteed that, replication reduces the SpMxV execution time even if communication cost is minimized considerably, because of the incurred redundant work. Hence, the expense of performed redundant work should be less than the gain from minimizing the communication overhead of SpMxV for our model to work effectively. For this reason, the replication algorithms for SpMxV must be selected wisely so as not to increase the execution time of SpMxV.

In the framework of this work, our approach is a two phase methodology that minimizes communication cost in SpMxV through utilizing data replication and the associated redundant work. In the first phase, the objective is to minimize the total message volume while keeping the computational load balance on processors. With the framework of 1D row-wise or column-wise matrix partitioning methods, the objective is achieved by means of partitioning the matrix. The partitioning acquired from the outcome of the first phase is an input to the second phase so that it determines upper-bound on the total message volume while maintaining a given balance on the computational loads of processors. In the second phase, the objective is to further reduce the total message volume by performing redundant works. As the sparse matrix partitioning problem constituting the first phase is a well-studied problem [21, 10, 54], we focus on the second phase. For this reason, our contribution can be thought as as a post processing to the partitioning methods. For the second phase, we propose a directed graph replication model on the partitioned directed graph. We extend the Fiduccia-Mattheyses (FM) heuristic [16] to achieve replication of vertices on the partitioned directed graph that is obtained from the first phase. We designate this operation as a replicated FM. The proposed method is performed on each part of the predetermined partitioned matrix individually. We consider replication problem for each part independently from each other. Thus, the obtained solution for a part does not affect the other part‘s solution. So, the proposed approach is very amenable to coarse-grain parallelization.

Our aim in this work is to reduce execution time of parallel SpMxV via minimizing communication volume by doing redundant work. For this reason we develop two replication algorithms to solve the *Min-cut Replication Problem.* The proposed replication algorithms demonstrate significant improvements on communication volume; moreover, this volume improvements induce faster parallel

SpMxV operation. In Chapter 6, it is shown that the conducted test on various matrices confirms the completion time of replicated parallel SpMxV is less than the unreplicated parallel SpMxV.

The organization of the thesis is as follows. Chapter 2 gives background material on the matrix-vector multiplies, parallel SpMxV, partitioning types for parallel SpMxV and the Min-Cut Replication problem. The proposed directed graph model for replication and Min-cut replication formulation are discussed in Chapter 3. Chapter 4 presents two methods for replication for parallel SpMxV and implementation details of these methods. Application that the experiments conducted on is explained in Chapter 5. The experimental results are demonstrated and discussed in Chapter 6. Finally, the related works are investigated in Chapter 7.

Chapter 2

Background

2.1 Sparse matrix vector multiplication

Sparse matrix vector multiplication (SpMxV) of the form y = Ax is a kernel operation in iterative solvers used in solving linear system of equations. So, we concentrate on the SpMxV on square matrices, because our problem in this work require square matrix to meet the exact necessities of iterative SpMxV.

Given an $n \times n$ sparse square matrix A, a dense input vector x of size n, in the SpMxV of the form $y = Ax$, the dense output vector y of size n is computed as

$$y_i = \sum_{j=1}^{n} A_{ij} \times x_j, \quad \text{for } 1 \le i \le n \tag{2.1}$$

where y is the dense output vector of size n. Figure 2.1 shows the basic matrix-vector multiplication, i.e., $y_6 = A_{6,4} \times x_4 + A_{6,5} \times x_5 + A_{6,7} \times x_7 + A_{6,16} \times x_{16}$. In this section, we discuss the most used data storage formats and give the details for a baseline parallel implementation of the SpMxV operation.

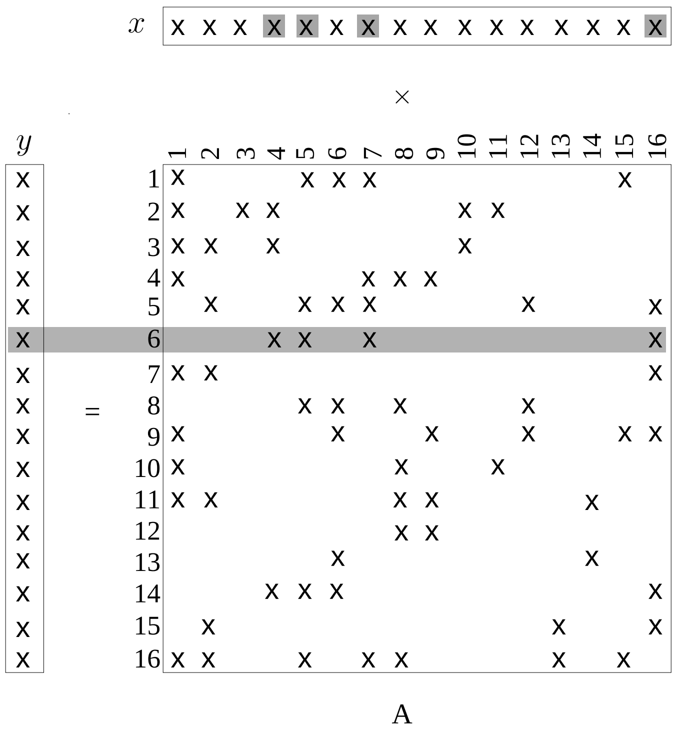


Figure 2.1: Basic Matrix Vector Multiplication

2.1.1 Data storage formats

The data storage format plays a crucial role in the performance of SpMxV operation. The standard dense matrix structures consume large amounts of memory and computation time when applied to sparse matrices. There are two commonly used storage formats: the compressed sparse row (CSR) and the compressed sparse column (CSC) format [42, 4].

The CSR and CSC schemes are used to reduce storage complexity of a sparse matrix by using special data structures. These schemes contain three one-dimensional arrays in order to store the matrix efficiently. The two of those arrays are in the size of the number of nonzeros on the matrix, while the other vector is in the size of one plus either the number of rows or columns in the matrix. The CSC scheme organizes the nonzeros of the matrix according to the

$$A = \begin{bmatrix} 4 & 0 & 3 & 0 & 6 \\ 0 & 0 & 2 & 2 & 0 \\ 9 & 8 & 5 & 0 & 0 \\ 5 & 2 & 0 & 0 & 1 \\ 1 & 0 & 3 & 5 & 0 \end{bmatrix}$$

$$\texttt{val} = \begin{bmatrix} 4 & 3 & 6 & 2 & 2 & 9 & 8 & 5 & 5 & 2 & 1 & 1 & 3 & 5 \end{bmatrix}$$

$$\texttt{colptr} = \begin{bmatrix} 1 & 3 & 5 & 3 & 4 & 1 & 2 & 3 & 1 & 2 & 5 & 1 & 3 & 4 \end{bmatrix}$$

$$\texttt{rowind} = \begin{bmatrix} 1 & 4 & 6 & 9 & 12 & 15 \end{bmatrix}$$

Figure 2.2: Example of CSR storage format.

columnwise order, whereas the CSR scheme organizes the nonzeros according to the rowwise ordering.

The CSR format is reported to be the most common data structure used to store a sparse matrix for the SpMxV operation [42]. So we focus on the parallel SpMxV operation utilizing the CSR storage format. In this format, a sparse matrix is represented by three arrays: `val`, `colptr`, `rowind`. Nonzeros in each row of a sparse matrix are kept contiguously in a dense array `val`. The `colptr` array holds column index of each nonzero. The `rowind` array stores the starting point of each row of the sparse matrix in `val` and `colptr`. The `rowind` array is used to access both `val` and `col-ptr` arrays. Algorithm 1 shows basic SpMxV operation $y \leftarrow Ax$, where A is hold in the CSR format.

Algorithm 1 SpMxV using CSR format

```
Require: val, colptr, rowind arrays of $A$, input vector $x$, output vector $y$
1: for $i \leftarrow 1$ to $n$ do
2:     $y[i] \leftarrow 0$
3:     for $j \leftarrow$ rowind[$i$] to rowind[$i$+1]$-1$ do
4:         $y[i] \leftarrow y[i] + $ val[$j$] $\times x$[colptr[$j$]]
   return y
```

2.1.2 Parallel implementation

In the literature there are various matrix partitioning schemes for parallel SpMxV; namely 1D rowwise [10], 1D columnwise , 2D checkerboard and 2D jagged [11]. In this work, we focus on parallel SpMxV based on 1D rowwise partitioning. The parallel SpMxV algorithm based on 1D rowwise partitioning is referred to as the *row parallel* SpMxV algorithm [53, 52]. The communication requirement of the row-parallel SpMxV algorithm can be explained by considering the $K \times K$ block structure induced by a K-way rowwise partition of matrix A. A K-way rowwise partition of a given matrix can be divided as inducing a symmetric partial permutation on the rows and columns of matrix A to induce a K-way block structure as follows [53, 52].

All rows assigned to part k are permuted, after the rows assigned to $k-1$, where the permutation of rows in a part are arbitrary. A matrix partition/permutation is said to be symmetric, if column permutation conform the row permutation. So, both $y[i]$ and $x[i]$ are assigned to the same part, for $1 \leq i \leq n$. The symmetric partition is preferred in order to avoid communication during linear vector operations, so the input vector x and the output vector y are partitioned conformably with the partitions on rows. Figure 2.3 illustrates such a partition. In the figure, the block $A_{k\ell}$ holds the size of $n_k \times n_\ell$, where $\sum_{k=1}^{K} n_k = n$.

$$PAP^T = A_{BL} = \begin{bmatrix} A_{11} & A_{12} & ... & A_{1K} \\ A_{21} & A_{22} & ... & A_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A_{K1} & A_{K2} & ... & A_{KK} \end{bmatrix}$$

Figure 2.3: $K \times K$ block structure of A

Let y_k and x_k respectively denote the output and input vector parts corresponding to the row stripe A_{k*} and A_{*k} of A_{BL} In the row parallel algorithm, each processor P_k holds the row stripe A_{k*} of A_{BL} and performs the computation $y_k = A_{k*} \times x$. Suppose that A is a row-wise partitioned matrix which is multiplied by a input vector x of the form $y \leftarrow Ax$. In a rowwise partitioning, the processor

P_k is responsible for computing the subvector y_k of size m_k, i.e $y = [t_1^T, y_2^T ... y_K^T]^T$. The processor P_k keeps the subvector x_k of size n_k conforming to the partition on the input vector $x = [x_1^T, x_2^T ... x_K^T]^T$. According to these partitions the matrix A is permuted into the block structure. Then the following steps are executed at P_k for the row-parallel $y \leftarrow Ax$:

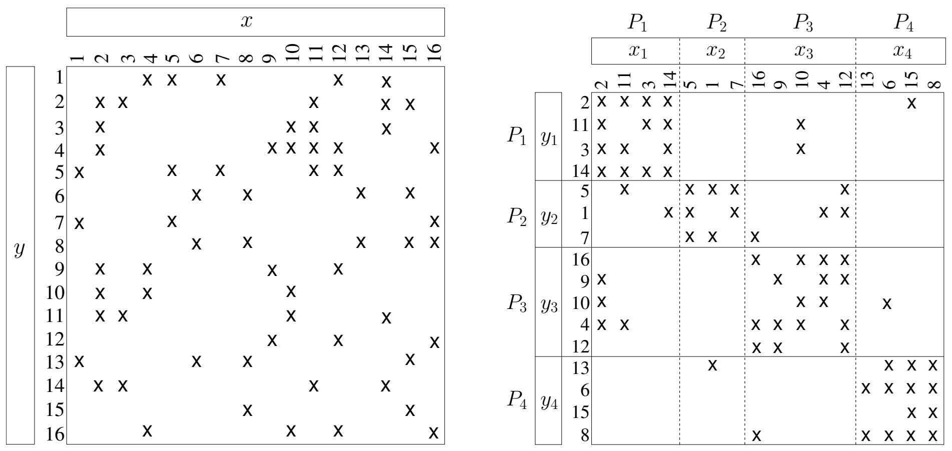
1. For each nonzero off-diagonal block $A_{\ell k}$, send sparse vector $\hat{x}_k^{\ell}$ to processor P_ℓ, where $\hat{x}_k^{\ell}$ contains only those entries of x_k corresponding to the nonzero columns in $A_{\ell k}$.
2. Compute the diagonal block product $y_k^k = A_{kk} \times x_k$, and set $y_k = y_k^k$.
3. For each nonzero off-diagonal block $A_{k\ell}$, and receive $\hat{x}_\ell^k$ from processor P_ℓ, then compute $y_k^\ell = A_{k\ell} \times \hat{x}_\ell^k$, and update $y_k = y_k + y_k^\ell$.

In row parallel SpMxV the first step of algorithm called as expand phase. Before the scalar products at the second step, expand operations are done in each processors. In the expand phase, all x-vector entries to be sent by a given processor to another processors are sent in a single message.

Figure 2.4(b) illustrates the sample square matrix A with 4×4 block structure of the size 16×16. The horizontal solid lines separate row partitions of A and the dashed lines separate column subvectors. In Figure 2.4(b), P_1 sends $\hat{x}_1^2 = (x_{11}, x_{14})$ to P_2 due to the nonzero columns in A_{21}. Those x-vector entries are needed by P_2 to compute y_5 and y_1. Similarly, P_1 sends $\hat{x}_1^3 = (x_1, x_{11})$ to P_3 to compute y_9, y_{10} and y_4. As a result, P_1 sends two messages with the total communication volume of four.

2.2 Message Passing Systems

Interprocessor communication is done via message-passing procedures in parallel applications. Since the communication overhead, which arises when the parallel application operates, stems from the the message passing performance of the



(a) The initial $n \times n$ matrix A

(b) The permuted $n \times n$ matrix A after k-way symmetric partition

Figure 2.4: Initial and permuted matrix A

application and it is generally evaluated in units of time. The equation of message size for interprocessor communication [51], the message transfer time T_{comm}, is given as

$$T_{comm} = t_{su} + t_{tr} \times m \tag{2.2}$$

where t_{su} is the start-up time (latency), t_{tr} is the per word transfer time between the message transmits from the processor sending a message to another processor receiving the message, m is the message size in terms of bytes. In the framework of this thesis, we try to minimize the total message volume in parallel system. In other words, we try to reduce the package size of transmitting data m.

2.3 Min-cut replication problem

Min-cut replication (MCR) problem is first defined by Hwang and El Gamal [23] for the formulation of replicating logic with the aim of reducing the pin count and the wiring density in partitioned logic networks. Here, we redefine the problem conforming the needs of our formulation of symmetric matrix partitioning with replication. First of all, we give the K-way replicated partition definition as follows.

Definition 1 (K-way replicated partition)**.** *Given a directed graph $\mathcal{G}$, $\Gamma(\mathcal{V}) = \{\mathcal{U}_k \subseteq \mathcal{V} : 1 \leq k \leq K\}$ defines a K-way replicated partition of $\mathcal{G}$, if the union of parts $\mathcal{U}_k$ is equal to the set $\mathcal{V}$ of all vertices, i.e., $\bigcup_{k=1}^{K} \mathcal{U}_k = \mathcal{V}$.*

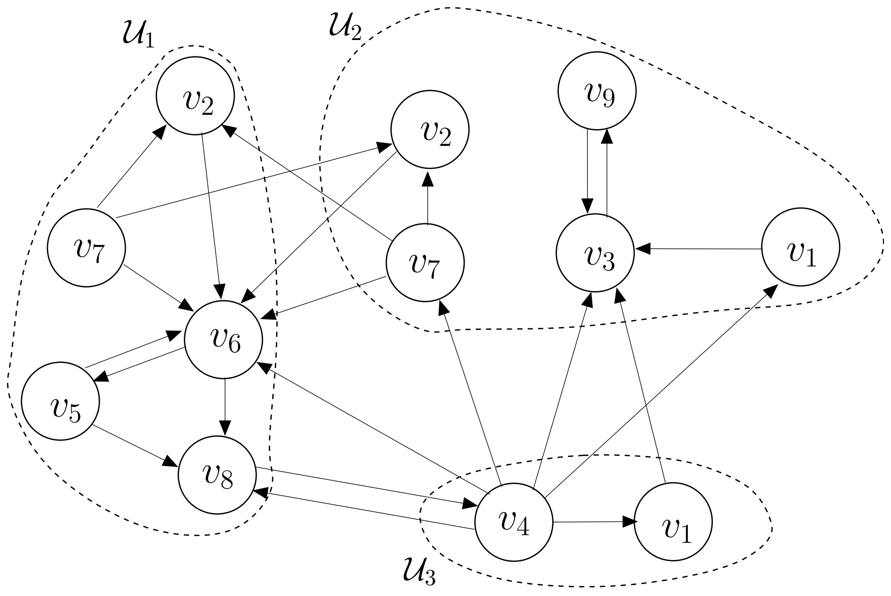


Figure 2.5: 3-way Replicated Partitioned Directed Graph

The basic replicated partitioned directed graph example is illustrated in Figure 2.3. The directed graph is partitioned into 3 parts where parts are not disjoint, some vertices can be found in multiple parts. In the figure v_2, v_7, and v_1 are replicated in different parts. The incoming and outgoing edges of the replicated vertices are also replicated.

Note that any K-way partition of vertices can also be considered as a K-way replicated partition where the parts are pairwise disjoint. However, in partitioning with replication the parts do not have to be pairwise disjoint. In an MCR problem instance, each vertex $v_i \in \mathcal{V}$ is associated with a weight w_i. The total weight W_k of a part $\mathcal{V}_k$ is computed as the sum over the weights of the vertices in $\mathcal{V}_k$, i.e.,

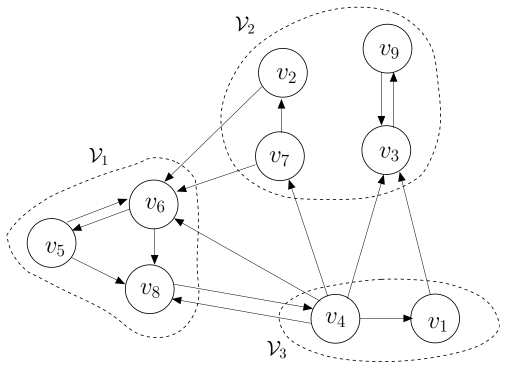
$$W_k = \sum_{v_i \in \mathcal{V}_k} w_i, \quad \forall\ 1 \leq k \leq K.$$

Moreover, let $\widehat{W}$ denote the upper bound on the total weight that any part can have. Given these, the MCR problem can formally be defined as follows.

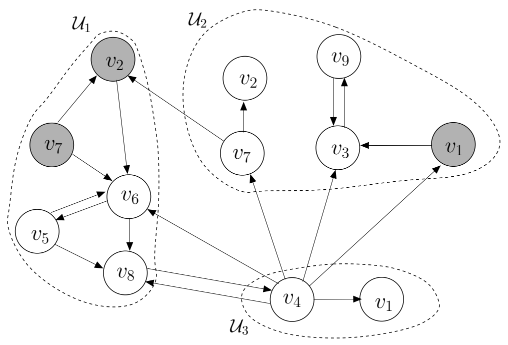
Problem 1 (Min-cut replication (MCR) problem [23])**.** *Given a directed graph $\mathcal{G}$, a part count K, an upperbound $\widehat{W}$, and a vertex partition without replication $\Pi = \{\mathcal{V}_k \subseteq \mathcal{V} : 1 \leq k \leq K\}$, find a collection of sets of vertices, $\{\mathcal{S}_k \subseteq \mathcal{V}–\mathcal{V}_k : 1 \leq k \leq K\}$ such that the total weight of no part of the resultant replicated partition Γ exceeds $\widehat{W}$ while minimizing the cutsize $\zeta(\Gamma)$, where*

$$\Gamma = \{\mathcal{V}_k \cup \mathcal{S}_k : 1 \leq k \leq K\}.$$

Here, $\mathcal{S}_k$ denotes the set of vertices to be replicated in part $\mathcal{U}_k$ of $\Gamma(\mathcal{V})$ such that $\mathcal{U}_k = \mathcal{V}_k \cup \mathcal{S}_k$.



(a) The directed graph G with partition Π



(b) The directed graph G with replicated partition Γ after Min-Cut replication

Figure 2.6: 3-way Replicated Partition

The initial directed graph is shown in Figure 2.6 where the graph is partitioned into 3 parts. Figure 2.6(b) shows a 3 way replicated partitioned graph after Min-Cut replication is performed. v_2 and v_7 are replicated to $\mathcal{U}_1$ and v_1 is replicated to $\mathcal{U}_2$.

Chapter 3

Min-cut Replication Formulation in Row Parallel SpMxV

3.1 Replication in row parallel SpMxV

Typically, the rows are partitioned and corresponding columns are placed symmetrically among K processors each of which is responsible for computing a disjoint subset of y-vector entries and holds corresponding subset of x-vector entries in row parallel SpMxV. In this thesis, our concern is parallel iterative solvers, where x-vector entries that a processor holds are computed using corresponding y-vector entries in the previous iteration. The symmetric replication implies that rows and columns of a processor are selectively and symmetrically replicated to other processors. A partitioned matrix with symmetric replication is said to be a symmetric replicated partitioned matrix. We can permute such a matrix using a $(n + r) \times n$ replicated partitioning matrix Q, where r reflects to the number of replicated rows/columns, as follows:

$$QAP^T = A^r_{BL} = \begin{bmatrix} A^r_{11} & A^r_{12} & \dots & A^r_{1K} \\ A^r_{21} & A^r_{22} & \dots & A^r_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A^r_{K1} & A^r_{K2} & \dots & A^r_{KK} \end{bmatrix}, \tag{3.1}$$

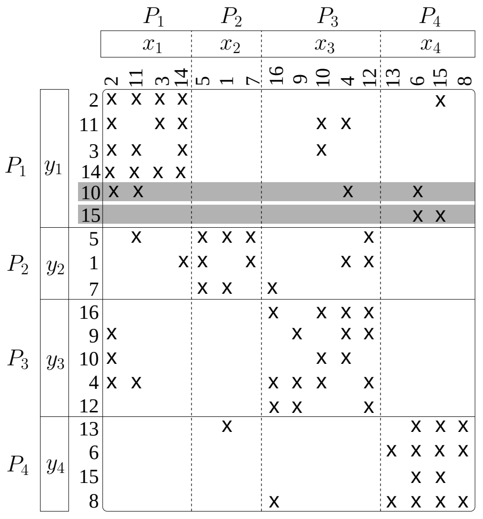
$$QAQ^T = A^\gamma_{BL} = \begin{bmatrix} A^\gamma_{11} & A^\gamma_{12} & \dots & A^\gamma_{1K} \\ A^\gamma_{21} & A^\gamma_{22} & \dots & A^\gamma_{2K} \\ \vdots & \vdots & \ddots & \vdots \\ A^\gamma_{K1} & A^\gamma_{K2} & \dots & A^\gamma_{KK} \end{bmatrix}. \tag{3.2}$$

Here, A^r_{BL} refers to the scenario that only the rows are replicated, whereas A^γ_{BL} refers to that both rows and columns are replicated. Let y^γ_k and x^γ_k denote the output and input subvector corresponding to the row stripe A^γ_{k*} and the column stripe A^γ_{*k} of A^γ_{BL}, respectively. Note that the input subvector x_k still denotes the subvector corresponding to the column stripe A_{*k} of A_{BL} as well as to the column stripe A^r_{*k} of A^r_{BL}. The row parallel SpMxV operation with row replicated performs as follows.

1. For each nonzero off-diagonal block $A^r_{\ell k}$, send sparse vector $\hat{x}^\ell_k$ to processor P_ℓ, where $\hat{x}^\ell_k$ contains only those entries of x_k that are not replicated in x^γ_ℓ and corresponding to a nonzero column in $A^r_{\ell k}$.
2. Compute the diagonal block product $y^k_k = A^\gamma_{kk} \times x^\gamma_k$, and set $y^\gamma_k = y^k_k$.
3. For each nonzero off-diagonal block $A^r_{k\ell}$, receive $\hat{x}^k_\ell$ from processor P_ℓ, then compute $y^\ell_k = A^r_{k\ell} \times \hat{x}^k_\ell$, and update $y^\gamma_k = y^\gamma_k + y^\ell_k$.

Figure 3.1 provides an illustration of symmetric replication scenarios. Figure 3.1(a) shows the illustration of A^r_{BL} where only rows row_{10} and row_{15} are replicated, whereas 3.1(b) demonstrates the scenario of A^r_{BL} where both rows row_{10} and row_{15} and columns col_{10} and col_{15} are replicated.

The modified multiplication scheme entails a reduction in communication volume due to transfer of x-vector entries at the expense of a redundant computation


(a) A^{r}_{BL}

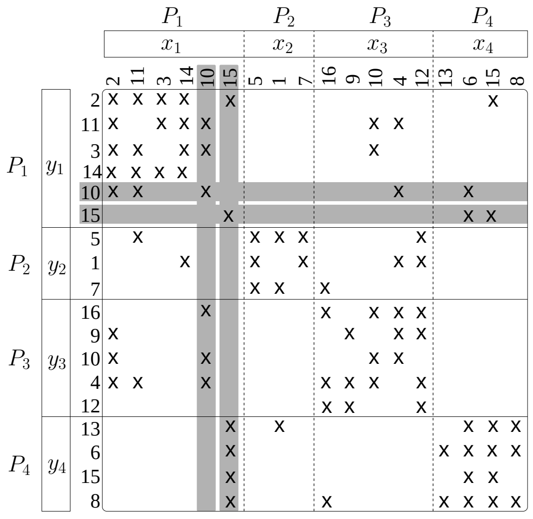

(b) A^{γ}_{BL}

Figure 3.1: Two Scenarios for Replicated Partitioned Matrix

of some y-vector entries. In particular, if we replicate row r_i such that $y_i \in y^{\pi}_{\ell}$ to a processor P_k such that $k \neq \ell$, we may obtain a reduction in the communication volume, as P_ℓ does not need to send the corresponding x-vector entry $x_i \in x^{\pi}_{\ell}$ to processor P_k. Figure 3.1(b) provides an illustrative example to clarify the reduction in communication volume. In the figure, the replication of x_{10} and x_{15} incur the data replication of row_{10} and row_{15} and redundant computations of y_{10} and y_{15}. Before replication (Figure 3.1(a)), P_1 incurs communication for three x-vector entries x_{10}, x_4, x_{15} to compute y-vector entries y_2, y_{11}, y_3, y_{14}. Although the replication of x_{10} seems to reduce the communication by one, it incurs extra communication of x_6 due to redundant computation of y_{10}. So, after the replication of x_{10}, P_1 requires three x-vector entries x_4, x_{15}, x_6 to compute y-vector entries y_2, y_{11}, y_3, y_{14} and x_{10}. Hence, the total communication is not changed after the replication of only x_{10}. If P_1 replicates x_{15} too, the communicational gain is reduced by one, since the computation of y_{15} does not incur extra communication according to the situation after the replication of x_{10}. Henceforth P_1 requires two x-vector entries x_4, x_6 to compute y-vector entries y_2, y_{11}, y_3, y_{14}, y_{10} and y_{15}.

The problem that we target in this thesis is to find a replication pattern of rows/columns such that the communication volume among the processors is minimized while maintaining that the workload of no processor exceeds a given workload capacity.

3.2 Min-cut replication formulation

We model a given square matrix A by a directed graph $\mathcal{G}(A)$ and formulate the symmetric replication problem as a combinatorial optimization problem on $\mathcal{G}(A)$. We construct the directed graph $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ as follows. For each row/column r_i/c_i in A, we introduce a vertex v_i in $\mathcal{V}$. Similarly, for each nonzero a_{ij}, we introduce a directed edge $e_{ij} = (v_i, v_j)$ in $\mathcal{E}$. In the directed graph, each vertex v_i is associated with a weight w_i which represents the workload, i.e., the number of nonzeros at row r_i, that is $w_i = nnz(r_i)$. Figure 3.2(b) illustrates the directed graph $\mathcal{G}(A)$ of a given A.

We formulate the symmetric replication problem as min-cut replication problem (see Section 2.3) where the cutsize $\zeta(\Gamma)$ of a K-way replicated partition $\Gamma = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is defined as

$$\zeta(\Gamma) = \sum_{i=1}^{K} |\beta(\mathcal{V}_i)| \tag{3.3}$$

where $\beta(\mathcal{V}_i)$ refers to the border of $\mathcal{V}_i$ as

$$\beta(\mathcal{V}_i) = \{u \in \mathcal{V} - \mathcal{V}_i : Adj(u) \cap \mathcal{V}_i \neq \emptyset\} \tag{3.4}$$

The following figures demonstrate fundamentally all phases of our approach. The given matrix A and its directed graph representation $\mathcal{G}$ are given in Figures 3.2(a) and 3.2(b), respectively. The partitioned matrix A and the illustration of $\mathcal{G}$ with a partition $\Pi = \{V_1, V_2, V_3\}$ are stated in Figures 3.2(c) and 3.2(d), respectively. The cutsize of partition Π is calculated as follows;

$$\beta(\mathcal{V}_1) = \{v_4, v_{10}\},\ \beta(\mathcal{V}_2) = \{v_{11}, v_{12}\},\ \beta(\mathcal{V}_3) = \{v_2, v_8, v_9, v_{11}\}$$

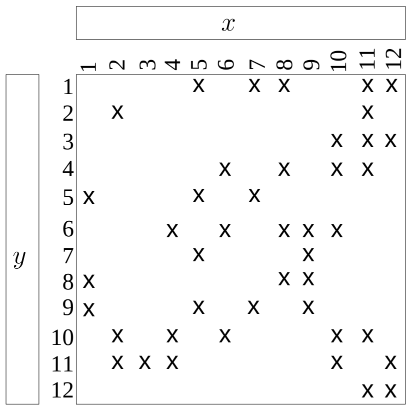
$$\zeta(\Pi) = \sum_{i=1}^{3} |\beta(V_i)| = 8 \tag{3.5}$$

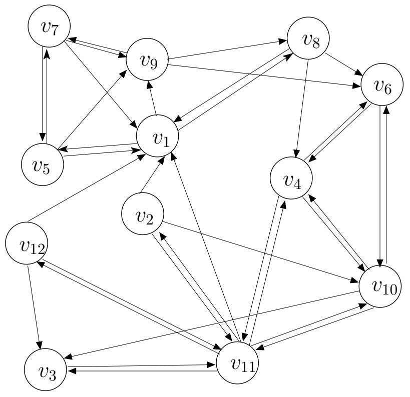
The cutsize value corresponds to total communication volume in row parallel SpMxV operation. In this manner, the communication pattern of matrix A with a partition Π will be; P_1 receives two x-vector entries x_4 and x_{10}, similarly P_2 and P_3 receives two and four x-vector entries, respectively. Thus, the total communication volume is eight. In parallel systems, the total received message volume is equal to the total sent message volume, naturally. Hence, if the total received message is minimized, the total sent message volume is also minimized. We discuss the volume improvement in terms of receiving messages volume in order to refer cutsize exactly.

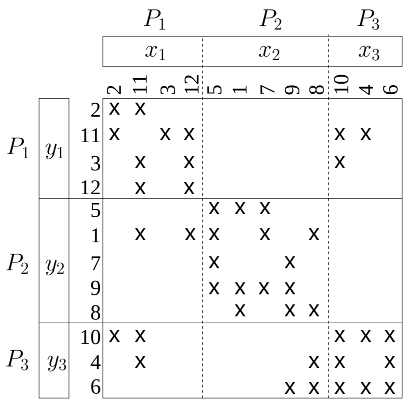
Finally, the directed graph $\mathcal{G}$ with the replicated partition $\Gamma = \{\mathcal{U}_1, \mathcal{U}_2, \mathcal{U}_3\}$ and it's matrix representation are given in Figures 3.2(d) and 3.2(f). With replication of three vertices, the cutsize of replicated partition Γ become six. So, the cutsize of the graph with partition Π is reduced from eight to six via symmetric replication. Similarly, the total communication volume will be eight messages along with the communication pattern of matrix A with a partition Γ; P_1 receives two messages v_4 and v_{10}, similarly P_2 and P_3 receives one and three messages, respectively.

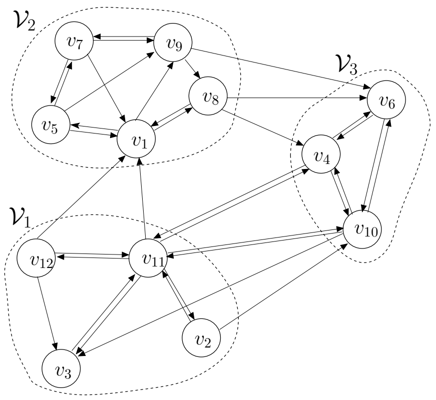
$$\beta(\mathcal{U}_1) = \{v_4, v_{10}\},\ \beta(\mathcal{U}_2) = \{v_{11}\},\ \beta(\mathcal{U}_3) = \{v_1, v_9, v_{11}\}$$

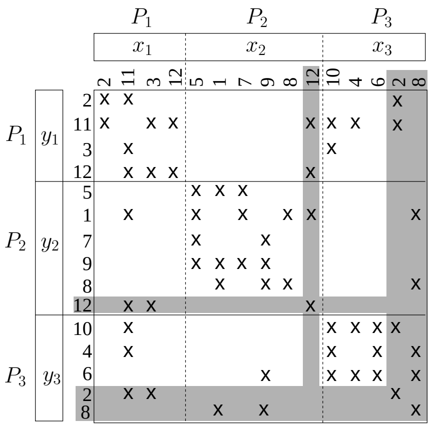
$$\zeta(\Gamma) = \sum_{i=1}^{3} |\beta(\mathcal{U}_i)| = 6 \tag{3.6}$$


(a) Initial Matrix A


(b) Directed Graph Representation of A $\mathcal{G}$


(c) Partitioned Matrix A


(d) Directed Graph $\mathcal{G}$ with a partition Π


(e) Replicated Partitioned Matrix

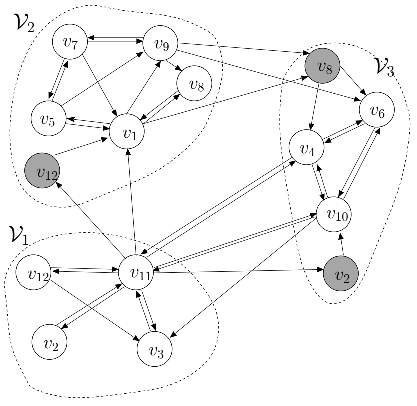

(f) Directed Graph $\mathcal{G}$ with a replicated partition Γ

Figure 3.2: Example of All Operations Applied on a Matrix

Chapter 4

Min-cut Replication Heuristics

4.1 Solution framework

We solve a given min-cut replication (MCR) problem instance using K independent minimum border subset (MBS) problem instances. Prior to the definition of the MBS problem, we define the border size $b(\mathcal{V}')$ of a vertex subset $\mathcal{V}' \subseteq \mathcal{V}$ of a given directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as

$$b(\mathcal{V}') = |\{v_i \in \mathcal{V} - \mathcal{V}' : Adj(v_i) \cap \mathcal{V}' \neq \emptyset\}|.$$

Problem 2 (Minimum Border Subset (MBS) Problem)**.** *Given a directed graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, a capacity C, and a fixed vertex subset $\mathcal{F} \subseteq \mathcal{V}$, find a vertex subset $\mathcal{S} \subseteq \mathcal{V} - \mathcal{F}$ such that $\sum_{v_i \in \mathcal{S}} w_i \leq C$, while minimizing the border size $b(\mathcal{F} \cup \mathcal{S})$.*

In an MCR problem instance, we are given a directed graph $\mathcal{G}$, an upper-bound $\widehat{W}$on part sizes, and a vertex partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$. For each part $\mathcal{V}_k \in \Pi$, we construct an MBS problem instance $(\mathcal{G}, C^k, \mathcal{F}^k)$ where $C^k = \widehat{W} - \sum_{v_i \in \mathcal{V}_k} w_i$ and $\mathcal{F}^k = \mathcal{V}_k$. After solving K MBS instances separately, we obtain a solution $\mathcal{S}^k$ for the instance corresponding to each part $\mathcal{V}_k \in \Pi$, where $\mathcal{S}^k$ refers to replication set to the k^{th} part. As a result, we construct the replicated partition Γ as

$$\Gamma = \{\mathcal{V}_k \cup \mathcal{S}^k : 1 \leq k \leq K\}.$$

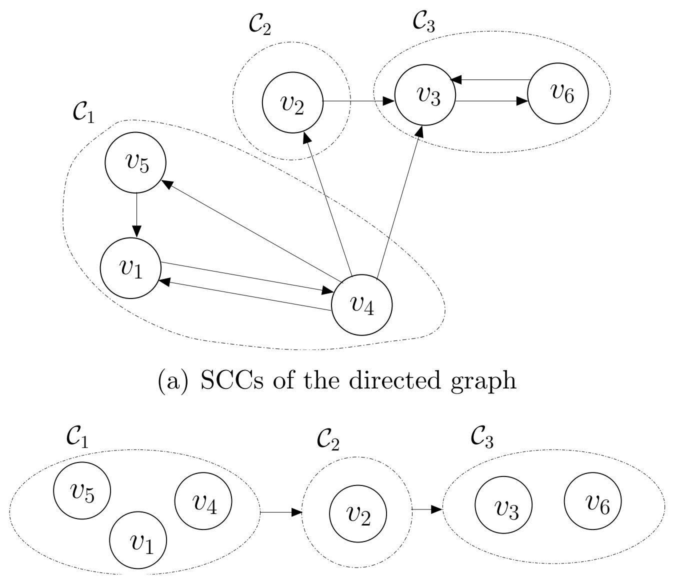
We develop two replication algorithms to solve the *MBS* problem. First algorithm *SCC-based* is more simple than second algorithm *FM-Based*. FM-based algorithm works more wisely than SCC-based algorithm. The proposed algorithms demonstrate improvements on communication volume, moreover this volume improvements induce faster parallel SpMxV operations.

4.2 SCC-based coarse-grain heuristic

This algorithm utilizes a three phase approach for solving the *MBS problem*. In the first phase, we exclude the fixed vertices in $\mathcal{F}$ from the given directed graph $\mathcal{G}$ and form the vertex-induced subgraph $\mathcal{G}[\mathcal{V}-\mathcal{F}]$. In the second phase, we find SCC's of $\mathcal{G}[\mathcal{V}-\mathcal{F}]$ and construct the component graph whose vertices correspond to the SCCs of $\mathcal{G}[\mathcal{V}-\mathcal{F}]$. Note that the component graph of $\mathcal{G}[\mathcal{V}-\mathcal{F}]$ is a directed acyclic graph (DAG) [12]. Thus, we run a topological sort algorithm on the component graph. Finally, in the third phase, we replicate SCCs according to the topological sort order in $\mathcal{F}$. Details of the algorithm and the technical terms are expressed in this section.

4.2.1 Background

A *Strongly connected component (SCC)* is a maximal subset of mutually reachable vertices[13]. Figure 4.1(a) shows the SCCs on the directed graph. A *topological ordering* of a directed graph is a linear sequence of its vertices such that the following condition is hold; for every edge (u, v), u comes before v in the ordering [13]. A topological ordering is available if and only if directed graph has no any cycle, in other words directed graph should be DAG [12, 47, 5]. It is possible that there can be multiple solution for the topological ordering of a single DAG.



(a) SCCs of the directed graph

(b) Possible Topological Order of the directed graph

Figure 4.1: Construction of SCCs and Topological Ordering of Components Graph

4.2.2 Algorithm

The MBS Problem is solved for each part of the directed graph partition independently, as stated in Chapter 2.

Algorithm 2 shows the steps of the SCC-based algorithm. The first step of this algorithm is to remove the fixed vertices subset $\mathcal{F}$ and all its edges from the directed graph $\mathcal{G}$. $\mathcal{F}$ refers to the part $\mathcal{V}_k$, which vertices to be replicated. In line 2, *strongly connected components (SCC)* are constructed in the vertex induced subgraph $\mathcal{G}[\mathcal{V}-\mathcal{F}]$. After the SCCs are constructed, the directed graph is transformed into components graph G_C which is coarser than the initial graph. In line 3, this components graph $\mathcal{G}_C$ has the properties of a *DAG*. The resulted graph $\mathcal{G}_C$ is ordered by the *topological ordering* in order to sequence the components graph $\mathcal{G}_C$. The first two phases is done up to this step.

As a third phase of the algorithm, the replication subset $\mathcal{S}$ is chosen according to the topological order, in line 6. When replicating a component, the component

Algorithm 2 SCC-Based$((G, \widehat{W}, \mathcal{F}))$

```
1: $\mathcal{G}' \leftarrow \mathcal{G}[\mathcal{V} - \mathcal{F}]$
2: $\mathcal{G}_C \leftarrow$ConstructSCCs$(\mathcal{G}')$
3: $List \leftarrow$TopologicalSort$(\mathcal{G}_C)$
4: $W(\mathcal{S}) \leftarrow 0$
5: repeat
6:     $\mathcal{S} \leftarrow \mathcal{S} \cup C_i$, where $C_i \in List[top]$
7:     Remove $C_i$ from $List$
8:     $W(\mathcal{S}) \leftarrow W(\mathcal{S}) + W(C_i)$
9: until $(W(\mathcal{F}) + W(\mathcal{S})) < \widehat{W}$
10: $P_k \leftarrow P_k \cup \mathcal{S}$
11: return $\mathcal{S}$
```

which has no incoming edge from any components is selected automatically in the way of the topological order. So, the component with the in-degree value is equals to zero is replicated at each iteration. In this way the algorithm guarantee that replicating a component cannot cause increasing of the *border*, namely does not incur any additional incoming edge. Moreover, minimization on the border size is expected eventually. The weight of the replication subset is calculated after a component is replicated in line 8. The replication procedure is repeated until the the summation of weight of fixed vertices and replication subset $(W(\mathcal{F}) + W(\mathcal{S}))$ are higher than the $\widehat{W}$. Finally, we find the replication subset $\mathcal{S}$ that possibly minimize the *border* of $\mathcal{F}$.

4.2.3 Implementation

We implement a linked list data structure to store topological order of components. The weights of each component is equal to the sum of the weight of all vertices in the component. The experimental results of this algorithm will be given in Chapter 5.

4.3 Replicated Fiduccia - Mattheyses Algorithm

In this algorithm, we modify the widely used iterative heuristic Fiduccia - Mattheyses [16] (FM). The FM heuristics is used generally for graph and hypergraph partitioning. The FM algorithm provides an efficient solution to the problem of task or network partitioning [21, 44, 10] by minimizing the inter-part connections of the partition. The FM algorithm runs in linear time in general with proper choice of data structure, such as bucket list and vertex locking. The proposed algorithm is expressed in details in the following sections.

4.3.1 Background

Iterative improvement heuristics based algorithms are widely used in graph and hypergraph partitioning. The quality of a random initial partition is improved iteratively with using these heuristics. The survey [2] discusses some of these heuristics in detail. Most of the algorithms exploit Kerninghan-Lin (KL) [28] and Fiduccia-Mattheyses (FM) [16] heuristics which are used to improve the quality of a given bipartition. The KL-based algorithms proceed in a series of passes and swap the vertex pairs in different parts of the bipartition at each pass. However, FM-based algorithms use single vertex moves from one part to the part. As an iterative improvement heuristic, FM is widely used due to its better running time performance. FM heuristics operate with multiple passes over all vertices. Each pass consists of multiple iteration of vertex moves. All vertices are *unlocked* at the beginning. The vertex is *locked* after moving to the other side and the gain value of its unlocked neighbours are updated. The improvement in the cutsize is stored at the end of each iteration. When all vertices are locked or there is no feasible move, the pass terminates and the bipartition is selected for the next pass by choosing the iteration with the highest improvement in the cut size. The passes continue until the improvement attained in a pass drops below a predefined threshold.

There are some techniques that improve the quality of FM heuristics further. The work [30] introduces the look-ahead feature to select vertex for the situation when the tie-break occurs, in other words if there are more than one vertex with highest gain value. The quality improvements of different tie-breaking strategies and data structures are investigated by [18]. The probabilistic gain computation, which is capable of estimating the future implications of moving a vertex, is proposed by [15]. The techniques proposed by [1] can make FM heuristic run faster. Two of these techniques are; using only boundary vertices to move and stops when no further improvement achieved in a pass.

In our algorithm we adopt the FM heuristics for directed graph. Our aim is to minimize the *border* as mentioned in the *MBS problem*. We also have two different part but we called them as *inside* part $\mathcal{I}$, and *outside*. The gain of each vertex is calculated according to the improvement in the cutsize to be obtained if that vertex is replicated to $\mathcal{I}$. Note that $\mathcal{I}$ is equals to $\mathcal{F}$ at the beginnig and it grows with the replicated vertices. Outside part holds the initially unlocked vertices that can be replicated to $\mathcal{I}$. As mentioned earlier, we solve the problem for each part of matrix separately and independently.

4.3.2 Algorithm

The following data structures are introduced for the proper execution of our algorithm. There are 2 color of array types of data structures, $BLACK$ and RED, to define the gain of each vertex in FM procedure.

$BLACK$ value of vertex states that the vertex is in *border*. In other words, if $BLACK$ value is greater than zero $BLACK[v_i] > 0$, vertex v_i has a outgoing edge into part $\mathcal{I}$ and the vertex v_i has a potential to reduce the border size with replication.

Black set of a vertex v:

$$BLACK(v) = \{(v, w) \in E : w \in \mathcal{I}\} \tag{4.1}$$

RED value of vertex represents the number of vertices that increase the border size. In other words, RED value of v_i is the number of new vertices that become the boundary vertex after replication of vertex v_i. If $RED[v_i] > 0$ then the replication of v_i can cause increase in the border size and brings extra communicational cost.

Red set of a vertex v:

$$RED(v) = \{(w, v) \in E : w \notin \mathcal{I}, BLACK(w) \subseteq \{v\}\} \tag{4.2}$$

Delta function is needed because even if a vertex v_y has more than one edge to v_z where $v_z \in \mathcal{I}$, the communication gain is not greater than one after replication of v_y. Moreover, only $BLACK$ value of vertices is exposed to delta function, not Red values.

Delta function for a set S:

$$\delta(S) = \begin{cases} 1 & S \neq \emptyset \\ 0 & otherwise. \end{cases} \tag{4.3}$$

Primary Gain of a vertex v:

$$g(v) = \begin{cases} \delta(BLACK(v)) - RED(v) & v \notin \mathcal{I} \\ RED(v) - \delta(BLACK(v)) & otherwise \end{cases} \tag{4.4}$$

Primary gain of a vertex meets the exact difference in the border size after replication that vertex. Positive gain value of a vertex means that border size reduces by the respective replication. When replicating a vertex, primary gain can get a value in interval $[-N, 1]$, where N is the number of columns or rows in a square matrix. Similarly, when unreplicating a vertex, which is already replicated in one of the previous phases of FM, primary value can get a value in interval $[N, -1]$.

In our FM algorithm, we implement two priority queues which are keyed with respect to primary gain values. These queues hold vertices with primary gain values for two possible moves which are replicating and unreplicating moves.

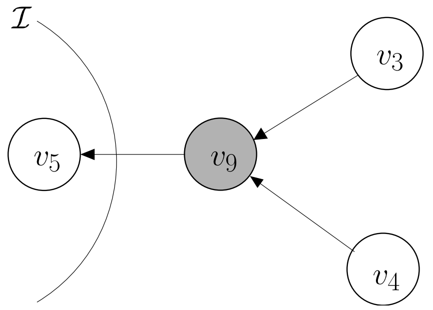
Secondary gain value is exploited because of the solution of tie-breaking problem. For secondary gain we implement a special priority queues which are keyed with 2 cascaded values, first step is primary gain and the second step is secondary gain. Suppose that two vertices have the same primary gain in the priority queue, in that situation the vertex that has a higher secondary gain will be selected for replication. Thus, the tie breaking problem in the priority queues are minimized, as increases in the effectiveness of the FM algorithm are anticipated. In equation 4.5, delta function is not applied on the $BLACK$ values of vertices. In this way the vertices that have more outgoing edges on the border are selected instead of the vertices that has the same primary gain but less outgoing edges to the other part.

Secondary Gain of a vertex v:

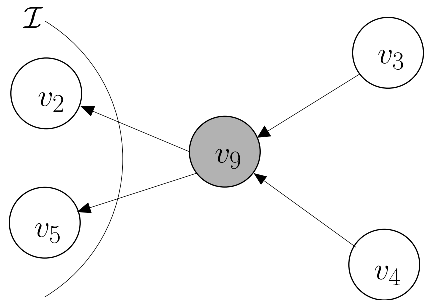
$$h(v) = \begin{cases} BLACK(v) - RED(v) & v \notin \mathcal{I} \\ RED(v) - BLACK(v) & otherwise \end{cases} \tag{4.5}$$

Algorithm 3 is repeated in the beginning of each passes of the our FM. The algorithm traverses each vertex only once and initializes the BOUND array with to NIL. In line 4, every vertex w that has an incoming edge from v is examined. If the vertex w is in part($\mathcal{I}$), the $BLACK[v]$ is increased by one. Then, $BOUND[v]$ is adjusted according to the value of $BLACK[v]$. At last, in line 12 *IncreaseReds* procedure is called for each vertex that is not inside the part. With completion of Algorithm 3, it is very easy to calculate the gains of vertices via $BLACK$ and RED data structures as in equations 4.4 and 4.5.

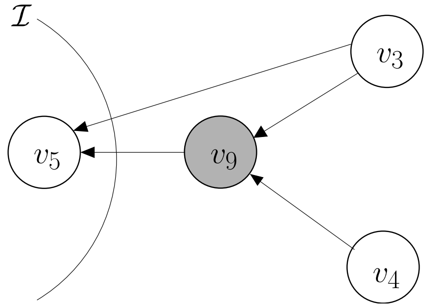
Algorithm 4 consists of two procedures, *IncreaseReds* and *DecreaseReds*, which are the auxiliary procedures that are reused multiple times in the gain calculation of our FM algorithm. *IncreaseReds* procedure increases the RED value of each vertex according to its neighbourhood with the parameter vertex v. When the $BLACK[v]$ is zero, this means that v has no edge to the other part, if there is a vertex w that has an incoming edge from v, that vertex may cause increase in the border size after replication w. Because, replication of w incurs extra vertex on the border of $\mathcal{I}$ due to v.



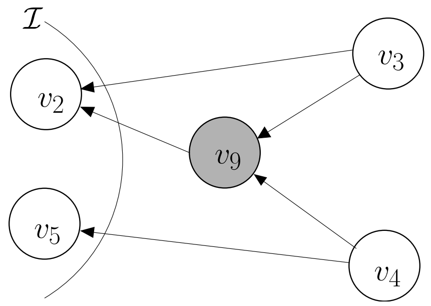
(a) B(v_9)=1, R(v_9)=2, Gain(v_9)=-1



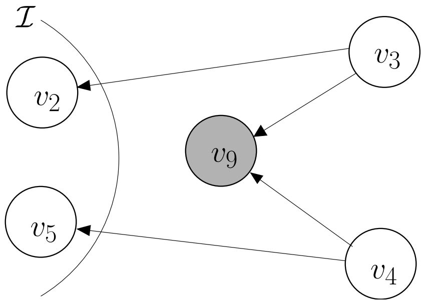
(b) B(v_9)=2, R(v_9)=2, Gain(v_9)=-1



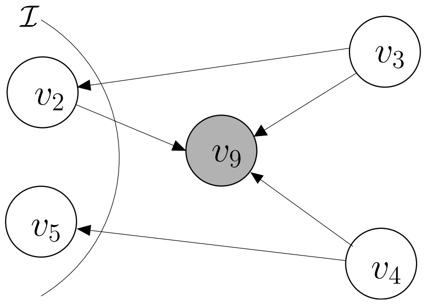
(c) B(v_9)=1, R(v_9)=1, Gain(v_9)=0



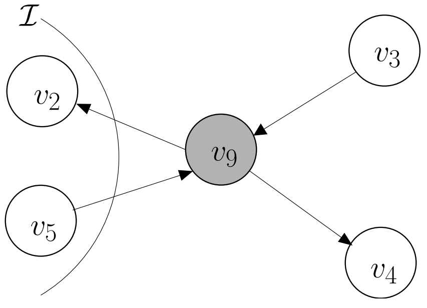
(d) B(v_9)=1, R(v_9)=0, Gain(v_9)=1



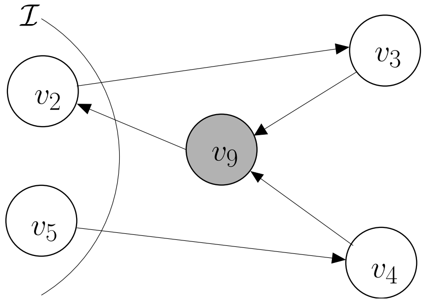
(e) B(v_9)=0, R(v_9)=0, Gain(v_9)=0



(f) B(v_9)=0, R(v_9)=0, Gain(v_9)=0



(g) B(v_9)=1, R(v_9)=1, Gain(v_9)=0



(h) B(v_9)=1, R(v_9)=2, Gain(v_9)=-1

Figure 4.2: Black, Red and Gain values of v_9 for different situations.

Algorithm 3 Initializing the Gains

```
procedure InitGains
    for each $v \in V$ do
        BOUND[$v$] $\leftarrow$ NIL
        for each $(v, w) \in E$ do
            if $w \in \mathcal{I}$ then
                BLACK[$v$] = BLACK[$v$] + 1
                if BLACK[$v$] = 1 then
                    BOUND[$v$] $\leftarrow w$
                else
                    BOUND[$v$] $\leftarrow$ NIL
        if $v \notin \mathcal{I}$ then
            IncreaseReds ($v$)
```

The data structure of $BOUND$ is meaningful when $BLACK[v] = 1$. At that time v is bounded with a vertex w, $w \in \mathcal{I}$, related to the border size reduction. Unreplication of w may change the gain value of v. Suppose $BLACK[v] = 1$ and $BOUND[v] = w$ and w must be in $\mathcal{I}$, $w \in \mathcal{I}$, and the gain of v is equals to zero. In this situation, v is in the set of $RED[w]$, after replication of v, v will be in $\mathcal{I}$, $v \in \mathcal{I}$, and the $RED[w]$ is decreased by one in according to the $BOUND[v] = w$.

Algorithm 4 Auxiliary Procedures

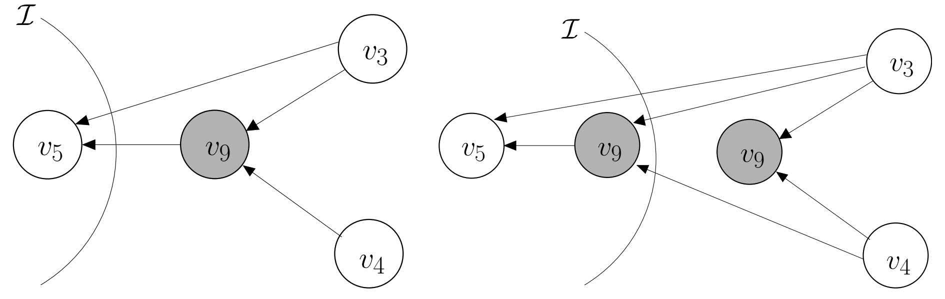
```
procedure IncreaseReds($v$)
    if BLACK[$v$] = 0 then
        for each $(v, w) \in E$ do
            RED[$w$] = RED[$w$] +1
    if BLACK[$v$] = 1 then
        RED[ BOUND[$v$] ] = RED[ BOUND[$v$] ] + 1

procedure DecreaseReds($v$)
    if BLACK[$v$] = 0 then
        for each $(v, w) \in E$ do
            RED[$w$] = RED[$w$] -1
    if BLACK[$v$] = 1 then
        RED[ BOUND[$v$] ] = RED[ BOUND[$v$] ] -1
```

MoveOut2In refers to replication move of a vertex from outside of the part to inside of part $\mathcal{I}$, whereas *MoveIn2Out* refers to unreplication move of the replicated vertex from inside of the part to the outside. The basic steps of our algorithm are given in Algorithm 5. The algorithm continues until the obtained gain is below some threshold value after a pass. We create two priority queues for the moves for *MoveOut2In* and *MoveIn2Out*, called as *unreplicated* and *replicated*, respectively.

Algorithm 5 Basic Steps of Our Algorithm

```
1: while there are passes to perform do
2:     Initialize gains queues for replicated and non-replicated
3:     while there is any valid operation do
4:         if MoveOut2In not feasible then
5:             MoveIn2Out
6:         else if MoveIn2Out and MoveOut2In feasible then
7:             if $unreplicated.val > replicated.val$ then
8:                 MoveOut2In
9:             else
10:                MoveIn2Out
11:        else if MoveIn2Out not feasible then
12:            MoveOut2In
13:        else
14:            break
```

If we look at Algorithm 6 in detail, the *MoveOut2In* procedure begins with the addition of the vertex v into the inside area $\mathcal{I}$. The *DecreaseReds* procedure makes proper adjustment on the RED values of the neighbour vertices of v. In line 4, for each vertex u that has outgoing edges to v is traversed because some vertices may reside at the border after v is replicated. Figure 4.3.2 illustrates this condition; before replication of v_9 vertex v_4 does not reside at the border however after v_9 is replicated, v_4 becomes a border vertex. If such a situation exists, the RED value of v_9 is increased due to increasing the border size because of v_4. Since v_4 resides at the border, the outgoing edges from v_4 does not incur any increase on the border size, the *DecreaseReds* performed for v_4 in order to decrease proper red values of its neighbours. In the rest of lines $BLACK$ and BOUND values are assigned. The BOUND array is necessary because suppose the BLACK value of a vertex v_i is equal to 1 and its Bound value is v_j, v_j stays at $\mathcal{I}$, when v_j is unreplicated then v_i does not stayed at the Border anymore. Thus, the outgoing edges from the vertex v_i causes increases the border size.



(a) B(v_9)=1, R(v_9)=1, Gain(v_9)=0 (b) B(v_9)=1, R(v_9)=1, Gain(v_9)=0

Figure 4.3: After Replication of v_9 into inside area $\mathcal{I}$

Algorithm 6

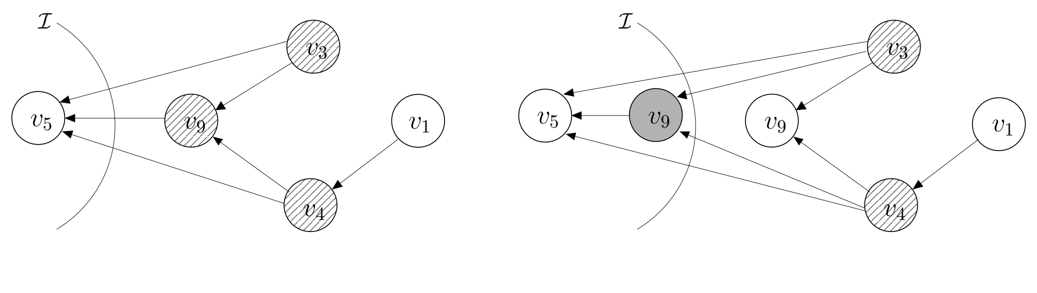
```
1: procedure MoveOut2In($v$)
2:     $\mathcal{I} \leftarrow \mathcal{I} \cup \{v\}$
3:     DecreaseReds($v$)
4:     for each $(u,v) \in E$ do
5:         if $u \notin \mathcal{I}$ then
6:             if BLACK[$u$] $= 0$ then
7:                 RED[$v$] = RED[$v$] + 1
8:             DecreaseReds($u$)
9:         if BLACK[$u$] $= 0$ then
10:            BOUND[$u$] $\leftarrow v$
11:        if BLACK[$u$] $= 1$ then
12:            BOUND[$u$] $\leftarrow$ NIL
13:        BLACK[$u$] = BLACK[$u$] +1
```

The unreplication procedure, called as *MoveIn2Out* in our algorithm, is described in the algorithm 4.3.2. It is a dual procedure of *MoveOut2In*. First we exclude the unreplicated vertex v from the inside area $\mathcal{I}$. Red values of neighbours of the unreplicated vertex is updated properly in *IncreaseReds* procedure. Then the vertices that has outgoing edge to replicated vertex v is scanned. For each vertex u that $(u,v) \in E$, the black values of u is decreased. In line 6, if the vertex u has no outgoing edge to $\mathcal{I}$, which means $BLACK[u] = 0$, the vertex u does not hold any boundary vertex in $\mathcal{I}$, $BOUND[u] = NIL$. If $BLACK$ value of u decreased to one $BLACK[u] = 1$, then the vertex u has a critical boundary vertex. The $BOUND[u]$ is scanned, and assigned in line 9-12. Updating gains

Algorithm 7

```
procedure MoveIn2Out($v$)
    $\mathcal{I} \leftarrow \mathcal{I} - \{v\}$
    IncreaseReds($v$)
    for each $(u, v) \in E$ do
        BLACK[$u$] = BLACK[$u$] -1
        if BLACK[$u$] = 0 then
            BOUND[$u$] ← NIL
        if BLACK[$u$] = 1 then
            for each $(u, w) \in E$ do
                if $w \in \mathcal{I}$ then
                    BOUND[$u$] ← $w$
                    break
        if $u \notin \mathcal{I}$ then
            if BLACK[$u$] = 0 then
                RED[$v$] = RED[$v$] -1
            IncreaseReds($u$)
```

of the neighbour vertices are completed with the line 13-16. Basic illustration of directed graph is shown in Figure 4.3.2. The gain of v_9 is equal to one before replication, this means that, after the replication of v_9 the border size is reduced by one. The patterned vertices demonstrate the border of the part $\mathcal{I}$. With the replication of v_9, the border size is reduced from three to two. The updated gain of v_9 is equals to -1, since the unreplication of v_9 cause increase in the border size by one.



(a) B(v_9)=1, R(v_9)=0, Gain(v_9)=1 (b) B(v_9)=1, R(v_9)=0, Gain(v_9)=-1

Figure 4.4: After Replication of v_9 to inside part $\mathcal{I}$

4.3.3 Implementation

We maintain two priority queues which are keyed with respect to the gain values of the vertices. For efficiency objectives, the priority queues are implemented as buckets lists . With bucket lists data structure, we can reduce the time complexity of a single pass of the FM algorithm to linear in the size of the directed graph.

Three different techniques are applied when selecting vertices for the solution of tie-breaking problem. These techniques are last-In first-out (LIFO), first-in first-out (FIFO) and random selection. For proper execution bucket based of priority queue implementation is needed [18]. In LIFO methods, the neighbour vertices of the moved vertex stay in front of the other vertices in the same bucket. This method works clustering-like move and replication possibility of the neighbour vertices of the moved vertex is increased. FIFO method is a dual operation of LIFO. The neighbour vertices of the moved vertex locate at the end of the queue in the same bucket. In random selection technique, after updating the gain values of the neighbour vertices of the moved, the location of neighbour vertices remain unchanged.

In well-known move based FM algorithms, the vertex is locked after moving to the other part in order to avoid futile moves. In our algorithm we also use locking mechanism; each vertex can move only once during a pass of the replication selection algorithm. After a vertex is moved from outside of the part to inside of the part or from inside of the part to outside of the part, that vertex is locked until the current pass of FM procedure is completed. (We have tried two or more move allowing locking methodologies but these experiments did not give us better results.)

It is clear that the smaller number of replica for the same results is better since replicas refer to the redundant work to be performed by the respective processor. Unlikely the typical FM heuristic, our algorithm obtains the iteration with the maximum gain and also the minimum number of replicated vertices at the end of phases. We called that technique as *Min-Rep*. This technique has several advantages for our problem. First, we try to minimize the redundant works while the border size remains identical. The second advantage is, the algorithm can minimize the border size further by minimizing the total weight of replica subset S, which means, reducing the weight of S permits doing more replication for the same *capacity* threshold.

The algorithms that are developed for the proposed model can not encode the increase in the number of messages. In some problems the number of message is an essential problem and may be more costly than the message volume. Thus, we implement an auxiliary variable which stores that after replication that vertex, whether the number messages is increased. If replication procedure cause an increases in the number number of messages, the replication operation of that vertex is ignored.

Chapter 5

Row Parallel SpMxV with Row Replication

The problem of maximizing the efficiency of Parallel SpMxV, is one of the problem that numerous scientists work on to improve the efficiency further. For this reason, parallel SpMxV libraries in the literature are investigated according to their compatibility, ease to use, and most importantly effectiveness. We are inspired from the parallel SpMxV library of [53]. The modifications and additions to this library are given the following section.

5.1 Data storage modifications

Typically part vector of partitioning tool for 1D row-wise partitioning is a one dimensional array where the order of array identifies corresponding row and value in each entry represents which part to hold this row. In our algorithm, we proposes a new partition vector structure for row replicated SpMxV, called *replicated part vector*. This vector can store two dimensional array which stores the default partition in the first columns. The other columns are used for the part where the corresponding row is replicated.

Due to redundant works, multiple computation of y vector elements, in order to reach true result the output vector of SpMxV should be processed. While the output vector y of unreplicated SpMxV is of size N, the result vector y of replicated SpMxV is a vector that is greater than unreplicated result vector, $(N + rep)$ because of the redundant computation of y elements. Similarly, the input vector x entries are replicated to some processors. The same x vector is stored in different processors. The distribution of x vector and gathering the y vector elements are added to the library [53].

5.2 Implementation details

Row replicated row parallel SpMxV is implemented using the library for parallel SpMxVs [53]. This library implements both row and column-parallel SpMxV algorithms. To cast our model on SpMxV we have modified the library efficiently. The modified algorithm is capable of computing the same y-vector entry in multiple processor and capturing the communication minimization exactly in parallel SpMxV. The necessary modifications for this requirements is briefly expressed in the following.

1. Providing partitioning indicators on x and y vector. The partitioning indicators are read by the master processor that broadcast them to the other processors. Each processor gets two arrays, one for input part vector and another for output part vector of sizes M and N, respectively. In this algorithm the vector that indicates the input partitioning called as inpart vector, similarly output part vector called as outpart vector. In our modification, the inpart and outpart vectors are the same for a processor since we just pertain square matrices with symmetric partitioning. However, there is no global inpart vector as in the original algorithm because of the replication. Each processor gets different inpart vector according to the replicated rows through redundant works. So the master processor configure inpart vector of each processor according to the replication vector and sends appropriate inpart vector to corresponding processor.

2. Providing matrix nonzeros and x vector component. A master processor reads the matrix file and scatters the matrix rows or columns according to the inpart vector and partitioning. In our implementation, we adopt 1D rowwise partitioning as mentioned before, therefore the master processor sends the x vector entries and rows of matrix to processors according to the replicated partition. Hence, the same x vector entries and conforming rows of matrix can be stored in multiple processors because of the data replication.

3. Determining the communication pattern. The work [53] describes the communication patterns for unreplicated SpMxV. For row replicated SpMxV, we do not need to modify this procedure, because the inpart of each part of matrix is filled according to replicated partition vector. After determining the communication pattern, each processor knows that how much data to be sent to which processor and how much data that to be received from which processor. We can see the communication volume minimization by looking the size of send or receive vector of a processor.

4. Determining local indices, setting local indices for the vector components to be sent and to be received, and assembling the local sparse matrix procedures [53] are adjusted automatically through the preliminary configuration at step 1, 2 and 3 for replicated parallel SpMxV. During assembling the local sparse matrix, the matrix is split into two parts *Aloc and Acpl* [50]. Here, *Aloc* contains the nonzeros in the diagonal area of the global matrix where holds nonzero a_{ij} in which $x[j]$ belong to the associated processor. *Acpl* matrix contains the nonzeros a_{ik} in which $x[k]$ belongs to another processor.

Chapter 6

Experimental Results

We have tested the performance of the proposed model through running our row-replicated row-parallel SpMxV algorithm according to the results of our replication algorithms.

6.1 Data Set

In experiments, we use sparse matrix collection of Florida University [14]. This collection is widely used by the linear algebra community. Its matrices cover a wide spectrum of domains, such as structural engineering, computational fluid dynamics, model reduction, electromagnetics, semiconductor devices, thermodynamics, materials, acoustics, computer graphics/vision, robotics/kinematic, etc. We select appropriate matrices from the dataset to observe rational results when solving the proposed problem. These matrices are large, sparse, structurally square.

To illustrate meaningful tables we eliminate some matrices with using following rules. Firstly the matrices that have less than 15000 rows are eliminated due to avoiding inconsistent results. The second rule is we eliminate some matrices according to the speed-up value of unreplicated matrix vector multiplication results. If speed-up value of a matrix is less than one, this means that the parallel execution time of the matrix takes more time than the serial multiplication of the matrix. Under third rule, we eliminate those matrices with speed-up values which are significantly higher. This kind of matrices have considerably less total message volume than other matrices and therefore they perform parallel multiplication pretty efficiently. For this reason, the replication algorithms cannot improve their total message volumes considerably. For eliminating these matrices, the matrices whose speed-up values are less than 8 in 16 way parallel multiplication is chosen. The final rule is that one representative matrix is selected randomly from each group of matrices since the matrices in the same group have similar sparsity pattern in the dataset [14]. Moreover, their results of both replication and parallel execution are almost identical for each matrix in the same group. The test matrices are shown in Table 6.1.

In parallel computing, speed-up refers to how much a parallel algorithm is faster than a conforming serial algorithm. The speed-up value is calculated by dividing the serial execution time with parallel execution. Speed-up is formulated as $S = T_s/T_p$, where T_s is the sequential execution time, T_p is the parallel execution time. Speed-up value may take value between $0 < S < K$, K is the number of processor in parallel system.

6.2 Setup

Aforementioned, our approach has two phases. In the first phase we partition the matrices into predefined number of parts. The matrices are partitioned into 16 parts through one dimensional rowwise decomposition by using PaToH (a multilevel hypergraph partitioning tool) [10] with default parameters. The partitioned matrix is used as an input partition to the second phase. Then, the replication vectors are obtained after running the replication algorithms on these matrices. Finally the matrices are multiplied according to the replication vector and completion times are acquired.

Matrix Name	**Row Count**	**NNZ**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**	**Speed Up**
af23560	23560	460598	6319	496	491	4	4.66
Andrews	60000	760154	27199	3013	2973	15	5.52
appu	14000	1853104	205173	12996	13110	15	1.57
av41092	41092	1683902	51306	5270	10454	15	2.94
bcsstk29	13992	619488	4557	384	342	9	5.30
cage12	130228	2032536	110521	10946	10020	15	3.42
case39	40216	1042160	79726	19965	6380	15	7.61
cit-HepTh	27770	352807	31446	2689	3140	15	2.03
conf6_0-8x8-80	49152	1916928	66828	4440	4338	7	4.15
crplat2	18010	960946	5232	444	432	6	6.66
crystm03	24696	583770	4826	369	368	5	5.59
denormal	89400	1156224	7024	608	614	8	7.89
Dubcova2	65025	1030225	5847	618	497	7	8.00
e40r0100	17281	553562	4698	444	417	6	5.08
EAT_SR	23219	325589	75995	5445	6901	15	1.01
ex35	19716	227872	1044	94	94	3	7.67
FEM_3D_thermal1	17880	430740	5980	535	541	5	4.28
g7jac080sc	23670	259648	6424	566	535	15	7.25
garon2	13535	373235	3597	299	306	8	4.23
gupta2	62064	4248286	106503	10305	13420	15	4.49
helm3d01	32226	428444	11035	939	985	12	4.78
ibm_matrix_2	51448	537038	20914	4487	2256	10	3.74
inlet	11730	328323	1772	136	141	3	3.39
lhr10c	10672	232633	9142	760	839	14	1.96
light_in_tissue	29282	406084	3108	270	276	7	4.97
mixtank_new	29957	1990919	19275	1458	1643	10	7.60
net150	43520	3121200	102506	16819	9782	15	3.26
pkustk02	10800	810000	4644	492	474	10	6.47
pli	22695	1350309	18502	1576	1638	9	5.76
poisson3Da	13514	352762	9894	714	720	12	2.98
Pres_Poisson	14822	715804	4344	408	406	6	6.17
qa8fm	66127	1660579	16338	1242	1284	9	7.47
skirt	12598	196520	1316	143	143	4	2.53
ted_B	10605	144579	472	60	50	3	2.11
TSOPF_RS_b39_c7	14098	252446	670	80	210	15	2.30
tube2	21498	897056	3893	420	438	9	6.84
vfem	93476	1434636	18246	1327	1279	9	7.15
viscoplastic2	32769	381326	10884	1215	1279	6	3.31

Table 6.1: Matrix Properties and Communication Characteristics After Partitioning

Table 6.1 shows the characteristics of the partitioned matrices that are used in the experiments. The test matrices are partitioned into 16 parts using 1D rowwise decomposition as a first phase of the our approach. Total volume, maximum receive message volume, maximum send message volume, and the maximum send message count, in other words the maximum number of send message packages in the partition, are shown in the table.

The parallel program that we implemented uses LAM/MPI [8] message passing library in parallel environment. Tests are performed on Beowulf [46] type PC cluster with 16 nodes. Each node has 3.00 Ghz Pentium 4 processor and 512 MB memory. The interconnection network is comprised of a 3COM SuperStack II 3900 managed switch connected to Intel Ethernet Pro 100 Fast Ethernet network interface cars at each node. The system runs the Linus kernel 2.4.14 and the debian GNU/Linus distribution.

In our algorithms, we define the capacity threshold according to the maximum loaded part in the partitioned graph at the first phase. Replication ratio is a number in term of percentage related to the maximum loaded part‘s load. For instance, if the replication ratio is 10%, it means that all parts can replicate the vertices until the load of replicated vertices reaches the capacity.

$$Capacity = Ratio \times MaxLoad \tag{6.1}$$

The imbalance values show the ratio between Maximum loaded part and average load of all parts. Formally unreplicated imbalance are calulated as follows;

$$Imbalance = Max_{Load}/Average_{Load}.$$

Similarly the replicated imbalance value is calculated as folows;

$$Imbalance_{Rep} = Max_{Load(Rep)}/Average_{Load}.$$

The same $average_{Load}$ value is used in two formulas in order to compare increases on the $Max_{Load(Rep)}$.

6.3 Results

Tables 6.2, 6.3, 6.4, and 6.5 show the normalized values of SCC-based algorithm results to the unreplicated partitioning state. Imbalance ratio before replication and after replication, total message volume, maximum receive and send message volume in the system, and the maximum send message count are illustrated in these tables. According to the tables the best volume improvement is acquired when the replication ratio is 0.15. When the replication ratio is increased, the total volume decreases as expected in some matrices, naturally. However, most of the matrices can not improve their total volume. The causes for this problem is that the constructed SCCs are too big to be replicated because of the capacity constraint.

At the second phase, one of the proposed replication algorithm is run over test matrices with using following properties;

1. Vertices are locked after one move
2. Min-Rep is applied at each pass
3. LIFO type bucket list data structure is used
4. Pass are continued until it converged (convergence value= (*Row Count*)/10000)

In Figure 6.6, we compare different replica selection techniques at priority queue implementation of FM algorithm. The LIFO scheme give slightly better improvement over random and FIFO schemes. Thus, the algorithm selects the vertices according to the LIFO order scheme. Necessary modifications is added our priority queue data structure, bucket list.

In our application, the parallel execution time is calculated according to following criteria;

1. Each multiplication is repeated 100 times

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**
af23560	1.02	1.02	1.00	1.00	1.00	1.00
Andrews	1.07	1.07	1.00	1.00	1.00	1.00
appu	1.02	1.02	1.00	1.00	1.00	1.00
av41092	1.04	1.04	1.00	1.00	1.00	1.00
bcsstk29	1.03	1.03	1.00	1.00	1.00	1.00
cage12	1.07	1.07	1.00	1.00	1.00	1.00
case39	1.05	1.05	1.00	1.00	1.00	1.00
cit-HepTh	1.08	1.08	0.89	0.97	0.97	1.00
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00
crplat2	1.01	1.01	1.00	1.00	1.00	1.00
crystm03	1.04	1.04	1.00	1.00	1.00	1.00
denormal	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.00	1.00	1.00	1.00	1.00
EAT_SR	1.05	1.05	0.68	0.64	0.54	1.00
ex35	1.01	1.01	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.03	1.00	1.00	1.00	1.00
g7jac080sc	1.06	1.06	0.94	0.99	0.87	1.00
garon2	1.00	1.00	1.00	1.00	1.00	1.00
gupta2	1.05	1.05	1.00	1.00	1.00	1.00
helm3d01	1.03	1.03	1.00	1.00	1.00	1.00
ibm_matrix_2	1.04	1.04	1.00	1.00	1.00	1.00
inlet	1.03	1.03	0.98	0.98	0.94	0.67
lhr10c	1.04	1.04	1.00	1.00	1.00	1.00
light_in_tissue	1.02	1.02	1.00	1.00	1.00	1.00
mixtank_new	1.00	1.00	1.00	1.00	1.00	1.00
net150	1.05	1.05	1.00	1.00	1.00	1.00
pkustk02	1.02	1.02	1.00	1.00	1.00	1.00
pli	1.04	1.04	1.00	1.00	1.00	1.00
poisson3Da	1.00	1.00	1.00	1.00	1.00	1.00
Pres_Poisson	1.05	1.05	1.00	1.00	1.00	1.00
qa8fm	1.03	1.03	1.00	1.00	1.00	1.00
skirt	1.06	1.06	1.00	1.00	1.00	1.00
ted_B	1.00	1.00	1.00	1.00	1.00	1.00
TSOPF_RS_b39_c7	1.00	1.00	1.00	1.00	1.00	1.00
tube2	1.02	1.02	1.00	1.00	1.00	1.00
vfem	1.02	1.02	1.00	1.00	1.00	1.00
viscoplastic2	1.05	1.05	1.00	0.99	0.99	1.00
Average	**1.03**	**1.03**	**0.98**	**0.99**	**0.99**	**0.98**

Table 6.2: SCC algorithm with replication ratio 0.00

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**
af23560	1.02	1.02	1.00	1.00	1.00	1.00
Andrews	1.07	1.07	1.00	1.00	1.00	1.00
appu	1.02	1.02	1.00	1.00	1.00	1.00
av41092	1.04	1.05	1.00	1.00	1.00	1.00
bcsstk29	1.03	1.06	1.00	1.00	1.00	1.00
cage12	1.07	1.07	1.00	1.00	1.00	1.00
case39	1.05	1.05	1.00	1.00	1.00	1.00
cit-HepTh	1.08	1.13	0.88	0.96	0.95	1.00
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00
crplat2	1.01	1.01	1.00	1.00	1.00	1.00
crystm03	1.04	1.04	1.00	1.00	1.00	1.00
denormal	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.05	0.96	1.00	1.00	1.00
EAT_SR	1.05	1.05	0.68	0.64	0.54	1.00
ex35	1.01	1.06	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.03	1.00	1.00	1.00	1.00
g7jac080sc	1.06	1.11	0.92	0.99	0.87	1.00
garon2	1.00	1.00	1.00	1.00	1.00	1.00
gupta2	1.05	1.06	1.00	0.99	1.00	1.00
helm3d01	1.03	1.03	1.00	1.00	1.00	1.00
ibm_matrix_2	1.04	1.04	1.00	1.00	1.00	1.00
inlet	1.03	1.04	0.97	0.98	0.94	0.67
lhr10c	1.04	1.08	1.00	1.00	1.00	1.00
light_in_tissue	1.02	1.02	1.00	1.00	1.00	1.00
mixtank_new	1.00	1.00	1.00	1.00	1.00	1.00
net150	1.05	1.05	1.00	1.00	1.00	1.00
pkustk02	1.02	1.02	1.00	1.00	1.00	1.00
pli	1.04	1.04	1.00	1.00	1.00	1.00
poisson3Da	1.00	1.00	1.00	1.00	1.00	1.00
Pres_Poisson	1.05	1.05	1.00	1.00	1.00	1.00
qa8fm	1.03	1.03	1.00	1.00	1.00	1.00
skirt	1.06	1.06	1.00	1.00	1.00	1.00
ted_B	1.00	1.05	1.00	1.00	1.00	1.00
TSOPF_RS_b39_c7	1.00	1.05	0.66	0.88	0.23	0.13
tube2	1.02	1.02	1.00	1.00	1.00	1.00
vfem	1.02	1.02	1.00	1.00	1.00	1.00
viscoplastic2	1.05	1.05	1.00	0.99	0.99	1.00
Average	**1.03**	**1.04**	**0.97**	**0.98**	**0.94**	**0.94**

Table 6.3: SCC algorithm with replication ratio 0.05

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**
af23560	1.02	1.02	1.00	1.00	1.00	1.00
Andrews	1.07	1.07	1.00	1.00	1.00	1.00
appu	1.02	1.02	1.00	1.00	1.00	1.00
av41092	1.04	1.05	1.00	1.00	1.00	1.00
bcsstk29	1.03	1.06	1.00	1.00	1.00	1.00
cage12	1.07	1.07	1.00	1.00	1.00	1.00
case39	1.05	1.05	1.00	1.00	1.00	1.00
cit-HepTh	1.08	1.18	0.87	0.95	0.94	1.00
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00
crplat2	1.01	1.01	1.00	1.00	1.00	1.00
crystm03	1.04	1.04	1.00	1.00	1.00	1.00
denormal	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.09	0.95	1.00	1.00	1.00
EAT_SR	1.05	1.05	0.68	0.64	0.54	1.00
ex35	1.01	1.06	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.03	1.00	1.00	1.00	1.00
g7jac080sc	1.06	1.17	0.91	0.99	0.87	1.00
garon2	1.00	1.00	1.00	1.00	1.00	1.00
gupta2	1.05	1.06	1.00	0.99	1.00	1.00
helm3d01	1.03	1.03	1.00	1.00	1.00	1.00
ibm_matrix_2	1.04	1.04	1.00	1.00	1.00	1.00
inlet	1.03	1.04	0.97	0.98	0.94	0.67
lhr10c	1.04	1.08	1.00	1.00	1.00	1.00
light_in_tissue	1.02	1.02	1.00	1.00	1.00	1.00
mixtank_new	1.00	1.00	1.00	1.00	1.00	1.00
net150	1.05	1.05	1.00	1.00	1.00	1.00
pkustk02	1.02	1.02	1.00	1.00	1.00	1.00
pli	1.04	1.04	1.00	1.00	1.00	1.00
poisson3Da	1.00	1.00	1.00	1.00	1.00	1.00
Pres_Poisson	1.05	1.05	1.00	1.00	1.00	1.00
qa8fm	1.03	1.03	1.00	1.00	1.00	1.00
skirt	1.06	1.14	0.98	1.00	1.00	1.00
ted_B	1.00	1.10	1.00	1.00	1.00	1.00
TSOPF_RS_b39_c7	1.00	1.10	0.64	0.88	0.23	0.13
tube2	1.02	1.02	1.00	1.00	1.00	1.00
vfem	1.02	1.02	1.00	1.00	1.00	1.00
viscoplastic2	1.05	1.05	1.00	0.99	0.99	1.00
Average	**1.03**	**1.05**	**0.97**	**0.98**	**0.94**	**0.94**

Table 6.4: SCC algorithm with replication ratio 0.10

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**
af23560	1.02	1.02	1.00	1.00	1.00	1.00
Andrews	1.07	1.07	1.00	1.00	1.00	1.00
appu	1.02	1.02	1.00	1.00	1.00	1.00
av41092	1.04	1.05	1.00	1.00	1.00	1.00
bcsstk29	1.03	1.06	1.00	1.00	1.00	1.00
cage12	1.07	1.07	1.00	1.00	1.00	1.00
case39	1.05	1.05	1.00	1.00	1.00	1.00
cit-HepTh	1.08	1.24	0.87	0.95	0.94	1.00
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00
crplat2	1.01	1.01	1.00	1.00	1.00	1.00
crystm03	1.04	1.04	1.00	1.00	1.00	1.00
denormal	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.09	0.95	1.00	1.00	1.00
EAT_SR	1.05	1.05	0.68	0.64	0.54	1.00
ex35	1.01	1.06	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.03	1.00	1.00	1.00	1.00
g7jac080sc	1.06	1.21	0.91	0.99	0.87	1.00
garon2	1.00	1.00	1.00	1.00	1.00	1.00
gupta2	1.05	1.06	1.00	0.99	1.00	1.00
helm3d01	1.03	1.03	1.00	1.00	1.00	1.00
ibm_matrix_2	1.04	1.04	1.00	1.00	1.00	1.00
inlet	1.03	1.04	0.97	0.98	0.94	0.67
lhr10c	1.04	1.08	1.00	1.00	1.00	1.00
light_in_tissue	1.02	1.02	1.00	1.00	1.00	1.00
mixtank_new	1.00	1.00	1.00	1.00	1.00	1.00
net150	1.05	1.05	1.00	1.00	1.00	1.00
pkustk02	1.02	1.02	1.00	1.00	1.00	1.00
pli	1.04	1.04	1.00	1.00	1.00	1.00
poisson3Da	1.00	1.00	1.00	1.00	1.00	1.00
Pres_Poisson	1.05	1.05	1.00	1.00	1.00	1.00
qa8fm	1.03	1.03	1.00	1.00	1.00	1.00
skirt	1.06	1.14	0.98	1.00	1.00	1.00
ted_B	1.00	1.15	1.00	1.00	1.00	1.00
TSOPF_RS_b39_c7	1.00	1.15	0.64	0.88	0.23	0.13
tube2	1.02	1.02	1.00	1.00	1.00	1.00
vfem	1.02	1.02	1.00	1.00	1.00	1.00
viscoplastic2	1.05	1.05	1.00	0.99	0.99	1.00
Average	**1.03**	**1.05**	**0.97**	**0.98**	**0.94**	**0.94**

Table 6.5: SCC algorithm with replication ratio 0.15

2. After each multiplication one of the basic iterative solver Jacobi method are simulated

3. For completion time, only the execution time of 100 iterations is taken.

The completion time results given in Tables 6.7, 6.8, 6.9, and 6.10 are the geometric mean of ten different SpMxV operation in which 100 repeated SpMxV occur. This process is required because of attaining more consistent results, since in the parallel environment there are several factors that affect the execution time of the parallel SpMxV. However, in our experiments the results are almost the same and the average standard deviation of completion times is quite small 0.001684.

Tables 6.7, 6.8, 6.9, and 6.10 show imbalance ratios before and after replication, total message volume, the maximum receive and send message volume, the number of maximum send messages per processor , and completion time of replicated parallel SpMxV. The values on the tables are normalized with respect to the unreplicated values given in the table 6.1.

If we analyze at the matrices individually, with replication ratio 0.00 in Table 6.7, The highest message volume improvement in the table occurred in *EAT_SR* . Furthermore, this volume improvement induces 26% time reduction on SpMxV completion time. However, if we consider at the *viscoplastic2* matrix, although the total volume is improved significantly, the completion time is not improved that much. This is because, the total message volume is not a single indicator that affects the completion time of SpMxV. The maximum send message count is another factor that affects the completion time. In the same table, the maximum send message count of the matrix *inlet* is decreased to 67%, this reduction incurs 18% faster parallel SpMxV.

Table 6.11 shows geometrical mean of the completion time of test matrices with different replication ratios. Even if the more replication ratio improves the total message volume, along with the increases on the redundant works cause worse completion times. If we increase the replication ratio further, the system tends to serial multiplication instead of parallel. Furthermore, when we apply the full replication on 16-processor system parallel system, we need to perform 16 serial execution of the same matrix concurrently. This is totally meaningless in the framework of this thesis. So, the replication ratio is selected wisely for the optimum reduction in completion time. If we look at Table 6.11, the preferable replication ratios are 0.05 and 0.10 for the test matrices for the fastest parallel SpMxV on average.

According to the results, our proposed model for row replicated SpMxV, using FM-like replication algorithm, can reduce the completion time of a matrix by up to 27% with replication ratio 0.10 and can improve the total message volume of a matrix by up to 62% with a replication ratio of 0.15. On the average, the completion time of replicated parallel SpMxV with replication ratio 0.10 is approximately 4% faster than the unreplicated SpMxV.

Strategy Replication	**Imbalance Volume**	**Total Volume**	**Max Recv Volume**	**Max Send Count**	**Max Send**
LIFO	1.14	0.85	0.87	0.85	0.99
Random	1.16	0.86	0.87	0.85	0.99
FIFO	1.16	0.86	0.87	0.86	0.99

Table 6.6: Different Bucket Strategies for Replication Ratio= 0.15

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**	**Completion Time**
af23560	1.02	1.02	1.00	1.00	1.00	1.00	1.00
Andrews	1.07	1.07	0.97	0.97	0.96	1.00	0.99
appu	1.02	1.02	1.00	1.00	1.00	1.00	1.00
av41092	1.04	1.04	0.99	0.96	0.99	1.00	1.00
bcsstk29	1.03	1.03	1.00	1.00	1.00	1.00	0.99
cage12	1.07	1.07	0.97	0.96	0.98	1.00	0.98
case39	1.05	1.05	1.00	1.00	1.00	1.00	1.00
cit-HepTh	1.08	1.08	0.86	0.96	0.92	1.00	0.95
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00	1.00
crplat2	1.01	1.01	1.00	1.00	1.00	1.00	1.00
crystm03	1.04	1.04	1.00	1.00	1.00	1.00	1.00
denormal	1.00	1.00	1.00	1.00	1.00	1.00	1.01
Dubcova2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
e40r0100	1.00	1.00	1.00	1.00	1.00	1.00	1.00
EAT_SR	1.05	1.05	0.67	0.64	0.53	1.00	0.74
ex35	1.01	1.01	1.00	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.03	1.00	1.00	1.00	1.00	1.00
g7jac080sc	1.06	1.06	0.85	0.87	0.82	1.00	0.98
garon2	1.00	1.00	1.00	1.00	1.00	1.00	1.01
gupta2	1.05	1.05	0.99	1.00	1.00	1.00	1.00
helm3d01	1.03	1.03	0.91	0.94	0.89	1.00	0.97
ibm_matrix_2	1.04	1.04	0.98	1.00	1.00	1.00	1.01
inlet	1.03	1.03	0.92	0.97	0.94	0.67	0.82
lhr10c	1.04	1.04	1.00	1.00	1.00	1.00	0.95
light_in_tissue	1.02	1.02	0.98	0.96	0.96	1.00	1.00
mixtank_new	1.00	1.00	0.99	0.99	0.97	1.00	1.01
net150	1.05	1.05	0.97	0.97	0.90	1.00	0.98
pkustk02	1.02	1.02	0.97	0.98	1.00	1.00	1.00
pli	1.04	1.04	1.00	1.00	1.00	1.00	1.01
poisson3Da	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Pres_Poisson	1.05	1.05	1.00	1.00	1.00	1.00	0.99
qa8fm	1.03	1.03	1.00	1.00	1.00	1.00	1.01
skirt	1.06	1.06	0.99	1.00	1.00	1.00	0.92
ted_B	1.00	1.00	1.00	1.00	1.00	1.00	0.93
TSOPF_RS_b39_c7	1.00	1.00	0.97	1.00	0.99	1.00	0.95
tube2	1.02	1.02	1.00	1.00	0.99	1.00	0.99
vfem	1.02	1.02	0.92	0.96	0.92	1.00	0.99
viscoplastic2	1.05	1.05	0.77	0.91	0.71	1.00	0.99
Average	**1.03**	**1.03**	**0.96**	**0.97**	**0.95**	**0.99**	**0.98**

Table 6.7: FM algorithm with replication ratio 0.00

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**	**Completion Time**
af23560	1.02	1.02	1.00	1.00	1.00	1.00	1.01
Andrews	1.07	1.11	0.96	0.97	0.96	1.00	1.00
appu	1.02	1.07	0.99	1.00	0.99	1.00	1.00
av41092	1.04	1.06	0.98	0.96	0.98	1.00	0.99
bcsstk29	1.03	1.09	0.98	0.95	1.00	1.00	1.00
cage12	1.07	1.12	0.95	0.95	0.97	1.00	0.98
case39	1.05	1.10	1.00	1.00	1.00	1.00	1.01
cit-HepTh	1.08	1.13	0.82	0.91	0.89	1.00	0.92
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00	1.00
crplat2	1.01	1.05	1.00	1.00	1.00	1.00	0.99
crystm03	1.04	1.09	0.99	1.00	1.00	1.00	0.93
denormal	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.05	0.66	0.60	0.75	1.00	0.94
e40r0100	1.00	1.05	0.75	0.83	0.84	1.00	0.97
EAT_SR	1.05	1.07	0.67	0.64	0.53	1.00	0.74
ex35	1.01	1.01	1.00	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.06	0.99	0.98	1.00	1.00	1.01
g7jac080sc	1.06	1.08	0.82	0.83	0.79	1.00	0.98
garon2	1.00	1.05	0.77	0.81	0.79	1.00	1.02
gupta2	1.05	1.10	0.97	0.98	0.99	1.00	0.99
helm3d01	1.03	1.08	0.84	0.84	0.83	1.00	0.94
ibm_matrix_2	1.04	1.09	0.96	0.96	1.00	1.00	0.98
inlet	1.03	1.07	0.86	0.85	0.82	0.67	0.82
lhr10c	1.04	1.09	0.99	1.00	1.00	1.00	0.92
light_in_tissue	1.02	1.07	0.97	0.96	0.95	1.00	1.00
mixtank_new	1.00	1.05	0.85	0.88	0.80	1.00	0.97
net150	1.05	1.10	0.93	0.85	0.85	1.00	0.90
pkustk02	1.02	1.07	0.94	0.98	1.00	1.00	1.01
pli	1.04	1.10	0.99	0.99	1.00	1.00	1.03
poisson3Da	1.00	1.05	0.92	0.93	0.87	1.00	0.99
Pres_Poisson	1.05	1.09	1.00	1.00	1.00	1.00	1.00
qa8fm	1.03	1.08	0.99	1.00	1.00	1.00	1.03
skirt	1.06	1.11	0.98	0.99	1.00	1.00	0.91
ted_B	1.00	1.05	0.52	0.58	0.54	1.00	0.94
TSOPF_RS_b39_c7	1.00	1.05	0.75	0.88	0.79	0.93	0.87
tube2	1.02	1.07	0.99	1.00	0.97	1.00	0.98
vfem	1.02	1.06	0.81	0.83	0.83	1.00	0.99
viscoplastic2	1.05	1.10	0.58	0.78	0.53	1.00	0.98
Average	**1.03**	**1.07**	**0.89**	**0.90**	**0.89**	**0.99**	**0.96**

Table 6.8: FM algorithm with replication ratio 0.05

Matrix Name	**Imbalance No-Replica**	**Imbalance Replication**	**Total Volume**	**Max Recv Volume**	**Max Send Volume**	**Max Send Count**	**Completion Time**
af23560	1.02	1.02	1.00	1.00	1.00	1.00	1.01
Andrews	1.07	1.16	0.96	0.97	0.96	1.00	0.99
appu	1.02	1.13	0.99	0.99	0.99	1.00	1.00
av41092	1.04	1.06	0.98	0.96	0.98	1.00	0.99
bcsstk29	1.03	1.14	0.97	0.95	1.00	1.00	1.01
cage12	1.07	1.16	0.95	0.95	0.98	1.00	0.99
case39	1.05	1.15	1.00	1.00	1.00	1.00	1.04
cit-HepTh	1.08	1.18	0.78	0.88	0.84	1.00	0.90
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00	1.00
crplat2	1.01	1.05	1.00	1.00	1.00	1.00	1.00
crystm03	1.04	1.12	0.99	1.00	0.96	1.00	0.98
denormal	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.10	0.66	0.59	0.74	1.00	0.96
e40r0100	1.00	1.10	0.59	0.67	0.75	1.00	0.96
EAT_SR	1.05	1.10	0.67	0.63	0.53	1.00	0.73
ex35	1.01	1.01	1.00	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.13	0.99	0.98	0.99	1.00	1.01
g7jac080sc	1.06	1.08	0.82	0.83	0.79	1.00	0.97
garon2	1.00	1.08	0.73	0.69	0.70	1.00	1.01
gupta2	1.05	1.15	0.96	0.95	0.99	1.00	0.99
helm3d01	1.03	1.11	0.83	0.83	0.81	1.00	0.94
ibm_matrix_2	1.04	1.14	0.94	0.91	0.98	1.10	0.98
inlet	1.03	1.07	0.85	0.85	0.82	0.67	0.82
lhr10c	1.04	1.14	0.99	1.00	1.00	1.00	0.94
light_in_tissue	1.02	1.11	0.97	0.96	0.95	1.00	0.95
mixtank_new	1.00	1.10	0.76	0.77	0.70	1.00	0.94
net150	1.05	1.15	0.91	0.77	0.82	1.00	0.85
pkustk02	1.02	1.11	0.93	0.98	1.00	1.00	1.01
pli	1.04	1.13	0.99	0.99	0.99	1.00	1.03
poisson3Da	1.00	1.10	0.84	0.86	0.86	1.00	0.96
Pres_Poisson	1.05	1.13	0.99	1.00	1.00	1.00	1.00
qa8fm	1.03	1.10	0.99	1.00	1.00	1.00	1.00
skirt	1.06	1.12	0.98	0.97	1.00	1.00	0.93
ted_B	1.00	1.08	0.49	0.35	0.48	1.00	0.95
TSOPF_RS_b39_c7	1.00	1.06	0.73	0.88	0.71	0.80	0.83
tube2	1.02	1.07	0.99	1.00	0.97	1.00	0.98
vfem	1.02	1.06	0.81	0.83	0.83	1.00	0.94
viscoplastic2	1.05	1.16	0.44	0.66	0.48	1.00	0.92
Average	**1.03**	**1.10**	**0.86**	**0.87**	**0.87**	**0.99**	**0.96**

Table 6.9: FM algorithm with replication ratio 0.10

Matrix Name	Imbalance No-Replica	Imbalance Replication	Total Volume	Max Recv Volume	Max Send Volume	Max Send Count	Completion Time
af23560	1.02	1.02	1.00	1.00	1.00	1.00	1.01
Andrews	1.07	1.20	0.96	0.97	0.96	1.00	1.06
appu	1.02	1.18	0.99	0.99	0.98	1.00	1.04
av41092	1.04	1.06	0.98	0.96	0.98	1.00	0.99
bcsstk29	1.03	1.18	0.96	0.94	1.00	1.00	1.04
cage12	1.07	1.21	0.95	0.95	0.98	1.00	1.00
case39	1.05	1.20	0.99	0.99	1.00	1.00	1.06
cit-HepTh	1.08	1.24	0.76	0.87	0.81	1.00	0.94
conf6_0-8x8-80	1.02	1.02	1.00	1.00	1.00	1.00	1.00
crplat2	1.01	1.12	0.99	1.00	1.00	1.00	0.97
crystm03	1.04	1.16	0.98	1.00	0.96	1.00	0.99
denormal	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Dubcova2	1.00	1.15	0.66	0.59	0.74	1.00	0.97
e40r0100	1.00	1.14	0.54	0.52	0.58	1.00	1.00
EAT_SR	1.05	1.11	0.67	0.63	0.53	1.00	0.78
ex35	1.01	1.01	1.00	1.00	1.00	1.00	1.00
FEM_3D_thermal1	1.03	1.17	0.99	0.98	0.99	1.00	1.01
g7jac080sc	1.06	1.12	0.81	0.83	0.79	1.00	0.97
garon2	1.00	1.08	0.73	0.69	0.70	1.00	1.01
gupta2	1.05	1.20	0.95	0.95	0.99	1.00	1.02
helm3d01	1.03	1.12	0.83	0.83	0.81	1.00	0.94
ibm_matrix_2	1.04	1.19	0.92	0.87	0.85	1.10	0.98
inlet	1.03	1.07	0.85	0.85	0.82	0.67	0.85
lhr10c	1.04	1.19	0.99	1.00	1.00	1.00	0.99
light_in_tissue	1.02	1.16	0.97	0.96	0.95	1.00	0.96
mixtank_new	1.00	1.15	0.73	0.75	0.68	1.00	0.95
net150	1.05	1.21	0.89	0.76	0.82	1.00	0.91
pkustk02	1.02	1.11	0.93	0.98	1.00	1.00	1.02
pli	1.04	1.20	0.99	0.99	0.99	1.00	1.02
poisson3Da	1.00	1.15	0.77	0.80	0.76	1.00	0.94
Pres_Poisson	1.05	1.20	0.99	1.00	1.00	1.00	1.04
qa8fm	1.03	1.10	0.99	1.00	1.00	1.00	1.02
skirt	1.06	1.19	0.97	0.95	1.00	1.00	0.94
ted_B	1.00	1.08	0.49	0.35	0.48	1.00	0.97
TSOPF_RS_b39_c7	1.00	1.06	0.73	0.88	0.71	0.80	0.88
tube2	1.02	1.14	0.99	1.00	0.97	1.00	0.98
vfem	1.02	1.06	0.81	0.83	0.83	1.00	0.93
viscoplastic2	1.05	1.21	0.38	0.54	0.40	1.00	0.99
Average	**1.03**	**1.14**	**0.85**	**0.85**	**0.85**	**0.99**	**0.98**

Table 6.10: FM algorithm with replication ratio 0.15

Replication Ratio	Imbalance Replication	Total Volume	Max Recv Volume	Max Send Volume	Max Send Count	Completion Time
0.00	1.03	0.96	0.97	0.95	0.99	0.98
0.05	1.07	0.89	0.90	0.89	0.99	0.96
0.10	1.10	0.86	0.87	0.87	0.99	0.96
0.15	1.14	0.85	0.85	0.85	0.99	0.98

Table 6.11: Average Results for different Replication Ratios

Chapter 7

Related Works

Sparse matrix vector multiplication (SpMxV) of the form y = Ax is a kernel operation in iterative solvers used in scientific applications. Sparse matrix-vector multiplication is a fundamental operation in many of the problems requiring the solution of least squares problems and eigenvalue problems [17, 3, 39], as well as the image reconstruction in medical imaging [40, 26]. In some iterative methods for linear solvers, the output vector y of a matrix-vector multiplication is transformed into a new input vector x. This transformation can be done directly or with some modification onto y vector. Some example of methods that involve large number of sparse matrix-vector multiplication in a successive manner in their algorithm are Jacobi, Gauss-Seidel, SOR [43]. Besides, Jacobi-Davidson iterative methods is used in computation of low-lying eigenstates of the Hamilton matrices, [45]. Moreover, sparse matrix-vector multiplication is the most time consuming procedure in that algorithm.

In the literature, replication is a widely used technique for various purposes in computer science, such as improving reliability, fault tolerance, accessibility, reducing processing and communication cost. Many discipline in computer science benefit from replication schemes through the mentioned methods. In parallel information retrieval systems document partitioning and term partitioning are the types of index partitioning schemes [48, 25, 9, 33, 35]. Replication is used to improve query throughput and fault tolerance in parallel information retrieval system for these schemes. Tomasic et al.[49] make comparison between different index distributions in their work and state that replication is necessary to improve query throughput. The work of [7] introduce an overview of the clustering architecture deployed at Google in which they use replication to improve query throughput.

We investigate replication in VLSI literature especially. Generally, VLSI community exploits various replication techniques to minimize the needed communication in network partitioning. A first discussion about logic replication is stated by the work [41]. Later, some other heuristics that extend the Fiduccia-Mattheyses (FM) algorithm to duplicate vertices during partitioning is published [29, 36]. The min-cut replication problem is defined and the optimal solution for finding the min-cut replication sets of a K-way partitioned directed graph is given by Hwang and El Gamal [22, 24, 23]. They present an efficient algorithm for determining an optimal min-cut replication set for a k-partitioned graph and implement a tool called TAPIR which is a software tool incorporating min-cut replication and presents the results of applying min-cut replication. The problem is modelled as a directed graph and their aim is to minimize cut edges with logic replication. [55] optimally solves the min-area min-cut replication problem on digraphs, which is to find min-cut replication set with the minimum sizes. The work [34] also proposes a network flow based algorithm to determine an optimum replication min-cut partitioning that requires minimum replication. [32] introduces an FM-based algorithm for optimum partitioning with replication and without size constraints. The survey [2] is presented about circuit partitioning and existing logic replication design.

Chapter 8

Conclusion

In this thesis, we proposed a model to reduce the communication of the repeated row-parallel sparse matrix vector multiplication (SpMxV) operations in iterative solvers through row replication. In iterative solvers, output vector of the SpMxV operation performed in the current iteration becomes the input vector of the SpMxV operation of the next iteration. Hence, a processor may compute an output-vector entry redundantly in the current iteration, which leads to an input-vector entry in the following iteration, instead of receiving that input-vector entry from another processor. Thus, redundant computation of selected output-vector entries through replicating the respective rows were utilized in reducing the total communication volume in row-parallel SpMxV. We proposed a graph theoretical formulation, namely the Min-Cut Replication (MCR) problem which can be tackled as K independent Minimum Border Subset (MBS) problems, for solving the target row replication problem for row-parallel SpMxV. Two algorithms were proposed to solve the MBS problem. In most of the test matrices, we achieved considerably better results than the unreplicated SpMxV in terms of execution time and communication volume improvement.

Bibliography

[1] C.J. Alpert, J.H. Huang, and A.B. Kahng. Multilevel circuit partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(8):655–667, 1998.

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

[3] J. Baglama, D. Calvetti, and L. Reichel. Iterative methods for the computation of a few eigenvalues of a large symmetric matrix. *BIT Numerical Mathematics*, 36(3):400–421, 1996.

[4] Z. Bai. *Templates for the solution of algebraic eigenvalue problems*, volume 11. Society for Industrial Mathematics, 2000.

[5] J. Bang-Jensen and G.Z. Gutin. *Digraphs: theory, algorithms and applications*. Springer Verlag, 2010.

[6] R. Barrett. *Templates for the solution of linear systems: building blocks for iterative methods*. Number 43. Society for Industrial Mathematics, 1994.

[7] L.A. Barroso, J. Dean, and U. Holzle. Web search for a planet: The google cluster architecture. *Micro, IEEE*, 23(2):22–28, 2003.

[8] G. Burns, R. Daoud, and J. Vaigl. Lam: An open cluster environment for mpi. In *Proceedings of supercomputing symposium*, volume 94, pages 379–386, 1994.

[9] B. Cahoon and K.S. McKinley. Performance evaluation of a distributed architecture for information retrieval. In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 110–118. ACM, 1996.

[10] Ü.V. Çatalyürek and C. Aykanat. Patoh: Partitioning tool for hypergraphs. Technical report, Department of Computer Engineering, Bilkent University, 1999.

[11] Ü.V. Çatalyürek and C. Aykanat. A fine-grain hypergraph model for 2d decomposition of sparse matrices. In *Proceedings of 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA*. Citeseer, 2001.

[12] N. Christofides. *Graph theory: An algorithmic approach*, volume 8. Academic press New York, 1975.

[13] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[14] T. Davis. University of florida sparse matrix collection. *NA digest*, 97(23):7, 1997.

[15] S. Dutt and W. Deng. A probability-based approach to vlsi circuit partitioning. In *Proceedings of the 33rd annual Design Automation Conference*, pages 100–105. ACM, 1996.

[16] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Papers on Twenty-five years of electronic design automation*, pages 241–247. ACM, 1988.

[17] G. GOLUB and M. KENT. Estimates of eigenvalues for iterative methods. *Mathematics of computation*, 53(188):619–626, 1989.

[18] L.W. Hagen, D.J.H. Huang, and A.B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(10):1199–1205, 1997.

[19] B. Hendrickson and T.G. Kolda. Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel processing. *SIAM Journal on Scientific Computing*, 21(6):2048–2072, 2000.

[20] B. Hendrickson and R. Leland. The chaco users guide. version 1.0. Technical report, Sandia National Labs., Albuquerque, NM (United States), 1993.

[21] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, pages 28–es. ACM, 1995.

[22] J. Hwang. Min-cut replication for improved partitions.

[23] J. Hwang and A. El Gamal. Optimal replication for min-cut partitioning. In *Proceedings of the 1992 IEEE/ACM international conference on Computer-aided design*, pages 432–435. IEEE Computer Society Press, 1992.

[24] L.J. Hwang and A. El Gamal. Min-cut replication in partitioned networks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(1):96–106, 1995.

[25] B.S. Jeong and E. Omiecinski. Inverted file partitioning schemes in multiple disk systems. *Parallel and Distributed Systems, IEEE Transactions on*, 6(2):142–153, 1995.

[26] Y.M. Kadah. New solution to the gridding problem. In *Proc. SPIE*, volume 4684, page 2. Citeseer.

[27] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359, 1999.

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

[29] C. Kring and A.R. Newton. A cell-replicating approach to minicut-based circuit partitioning. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 2–5. IEEE, 1991.

[30] B. Krishnamurthy. An improved min-cut algonthm for partitioning vlsi networks. *Computers, IEEE Transactions on*, 100(5):438–446, 1984.

[31] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing*, volume 110. Benjamin/Cummings, 1994.

[32] L.T. Liu, M.T. Kuo, C.K. Cheng, and TC Hu. A replication cut for two-way partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(5):623–630, 1995.

[33] A. MacFarlane, J.A. McCann, and S.E. Robertson. Parallel search using partitioned inverted files. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 209–220. IEEE, 2000.

[34] W.K. Mak and DF Wong. Minimum replication min-cut partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(10):1221–1227, 1997.

[35] A. Moffat, W. Webber, J. Zobel, and R. Baeza-Yates. A pipelined architecture for distributed text query evaluation. *Information Retrieval*, 10(3):205–231, 2007.

[36] R. Murgai, R.K. Brayton, and A. Sangiovanni-Vincentelli. On clustering for minimum delay/ara. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 6–9. IEEE, 1991.

[37] J.G. Nagy, K. Palmer, and L. Perrone. Iterative methods for image deblurring: a matlab object-oriented approach. *Numerical Algorithms*, 36(1):73–93, 2004.

[38] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[39] W.H. Press. *Numerical recipes in C: the art of scientific computing C*. Cambridge University Press, 1992.

[40] L.F. Romero and E.L. Zapata. Data distributions for sparse matrix vector multiplication* 1. *Parallel Computing*, 21(4):583–605, 1995.

[41] R.L. Russo, P.H. Oden, and PK Wolff Sr. A heuristic procedure for the partitioning and mapping of computer logic graphs. *Computers, IEEE Transactions on*, 100(12):1455–1462, 1971.

[42] Y. Saad and Research Institute for Advanced Computer Science (US). *SPARSKIT: A basic toolkit for sparse matrix computations.* Citeseer, 1990.

[43] Y. Saad and Y. Saad. *Iterative methods for sparse linear systems.* PWS Pub. Co., 1996.

[44] L.A. Sanchis. Multiple-way network partitioning. *Computers, IEEE Transactions on*, 38(1):62–81, 1989.

[45] G. Schubert, G. Hager, H. Fehske, and G. Wellein. Parallel sparse matrix-vector multiplication as a test case for hybrid mpi+ openmp programming. *Arxiv preprint arXiv:1101.0091*, 2010.

[46] T. Sterling, D.J. Becker, D. Savarese, J.E. Dorband, U.A. Ranawake, and C.V. Packer. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing.* Citeseer, 1995.

[47] K. Thulasiraman and M.N.S. Swamy. *Graphs: theory and algorithms.* Wiley Online Library, 1992.

[48] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Parallel and Distributed Information Systems, 1993., Proceedings of the Second International Conference on*, pages 8–17. IEEE, 1993.

[49] A. Tomasic and H. Garcia-Molina. Performance issues in distributed shared-nothing information-retrieval systems* 1. *Information processing & management*, 32(6):647–665, 1996.

[50] R.S. Tuminaro, J.N. Shadid, and S.A. Hutchinson. Parallel sparse matrix vector multiply software for matrices with data locality. *Concurrency - Practice and Experience*, 10(3):229–247, 1998.

[51] B. Uçar. *Parallel Sparse Matrix-Vector Multiplies and Iterative Solvers*. PhD thesis, Bilkent University, 2005.

[52] B. Uçar and C. Aykanat. Encapsulating multiple communication-cost metrics in partitioning sparse rectangular matrices for parallel matrix-vector multiplies. 2004.

[53] B. Ucar and C. Aykanat. A library for parallel sparse matrix-vector multiplies. Technical report, Tech. Rep. BU-CE-0506, Department of Computer Engineering, Bilkent University, Ankara, Turkey, 2005.

[54] B. Uçar and C. Aykanat. Partitioning sparse matrices for parallel preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 29(4):1683–1709, 2008.

[55] H.H. Yang and DF Wong. New algorithms for min-cut replication in partitioned circuits. In *Proceedings of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 216–222. IEEE Computer Society, 1995.