PARTITIONING SPARSE RECTANGULAR MATRICES FOR PARALLEL COMPUTING OF AA^Tx

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING AND INFORMATION SCIENCE
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

BY

BORA UÇAR

SEPTEMBER, 1999

PARTITIONING SPARSE RECTANGULAR MATRICES FOR PARALLEL COMPUTING OF AA^TX

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER

ENGINEERING AND INFORMATION SCIENCE

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Bora Uçar

September, 1999

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. Cevdet Aykanat(Principal 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.

Asst. Prof. Dr. Attila Gürsoy

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. Kemal Efe

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Science

ABSTRACT

PARTITIONING SPARSE RECTANGULAR MATRICES FOR PARALLEL COMPUTING OF AA^Tx

Bora Uçar
M.S. in Computer Engineering and Information Science
Supervisor: Asoc. Prof. Dr. Cevdet Aykanat
Spetember, 1999

Many scientific applications involve repeated sparse matrix-vector and matrix-transpose-vector product computations. Graph and hypergraph partitioning based approaches used in the literature aim at minimizing the total communication volume while maintaining computational load balance through one dimensional partitioning of sparse matrices. In this thesis, we consider two approaches which consider minimizing both the total message count and communication volume while maintaining balance on the communication loads of the processors. Two communication schemes are investigated for the fold and expand operations needed in the parallel algorithm. For the global communication scheme, we show that the problem of minimizing concurrent communication volume can be formulated as the problem of permuting the sparse matrix into a singly-bordered block-diagonal form, where the total and concurrent message count is determined by the interconnection topology. For the personalized communication scheme, a two stage approach is proposed. In the first stage, the total communication volume is minimized while maintaining balance on the computational loads of the processors. In the second stage, a novel communication hypergraph model is proposed which enables the minimization of the total message count while maintaining balance on the communication loads of the processors through hypergraph-partitioning-like methods. The solution methods are tested on various matrices and results, which are quite attractive in terms of solution quality and running times, are obtained.

Key words: Sparse Rectangular Matrices, Computational Hypergraph Model, Communication Hypergraph Model, Hypergraph Partitioning.

ÖZET

SEYREK DİKDÖRTGENSEL MATRİSLERİN AA^Tx'İN
PARALEL İŞLEMCİLERDE HESAPLANABİLMESİ İÇİN
PARÇALANMASI

Bora Uçar
Bilgisayar ve Enformatik Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doc. Dr. Cevdet Aykanat
Eylül, 1999

Birçok bilimsel uygulama, bir seyrek dikdörtgensel matrisin bir vektör ve de aynı matrisin transpozunun bir başka vektör ile çarpılmasını içermektedir. Şimdiye kadar kullanılan çizge ve hiperçizge parçalanması algoritmalarında toplam haberleşme hacmi azaltılırken işlemciler arasındaki işlemsel denge, matrisin tek boyutlu ayrıştırılması ile yakalanmaya çalışılmıştır. Bu tezde, toplam haberleşme sayısı ve hacmi iki yaklaşımla azaltılmaya çalışılmıştır. Genel haberleşme yaklaşımındaki birleşik haberleşme hacminin azaltılması problemi, matrislerin tek sınırlı çarpraz bloklar haline getirilmesi problemine dönüştürülmüştür. Bu yaklaşımdaki birleşik ve toplam haberleşme sayısı paralel işlemcilerin bağlantısına bağlı olup değiştirilememektedir. Kişiselleştirilmiş haberleşme yaklaşımındaki toplam haberleşme sayısı ve hacmi iki aşamada azaltılmıştır. Birinci aşamada, toplam haberleşme hacmi azaltılırken işlemsel denge sağlanılmıştır. İkinci aşamada, önerilen haberleşme hiperçizgesi yardımı ile toplam haberleşme sayısı azaltılırken işlemciler arasında haberleşme işi dengesi sağlanmaya çalışılmıştır. Sunulan yöntemler birçok matris üzerinde sınanmış ve iyi yönde sonuçlar elde edilmiştir.

Anahtar kelimeler: Seyrek dikdörtgensel matrisler, hesap hiperçizgesi, haberleşme hiperçizgesi, hiperçizge parçalama.

Anneme ve Babama

ACKNOWLEDGMENTS

Öncelikle danışmanım Cevdet Aykanat'a;
Buraları daha çekilir kılan Nihan Özyürek'e;
Programlama sırasında karşılaştığım sorunları çözmeme yardımcı olan Ümit V. Çatalyürek'e;
Tezi yazmama ve düzeltmeme yardım eden aynı zamanda inceldiğim her noktada bana destek olan kardeş Mehmet Koyutürk'e;

bu tez için teşekkür ederim.

Ayrıca ufkumu açan A. Pınar Sayın'a,
Computer Science'in kapılarını gösteren Ali Pınar'a şükranlarımı sunarım.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Repeated matrix-vector and matrix-transpose-vector products of a rectangular or structurally non-symmetric sparse-matrix comprise the core of many iterative parallel algorithms such as linear programs and linear system solvers. The efficient parallelization of such kind of algorithms requires the distribution of the nonzero elements of the matrix across processors in such a way that the computational work per processor is balanced and the cost of interprocessor communication is low.

In the literature, one dimensional partitioning of rectangular matrices is phrased in terms of partitioning bipartite graphs [15], and partitioning hypergraphs [1, 6]. Although recent works on modeling partitioning of rectangular or structurally non-symmetric sparse-matrices [1, 6, 15] handle the computational balance successfully, they fail to reveal the mystery behind the communication cost while minimizing it. The communication cost that is minimized in these works is considered to be the total communication volume across all processors. In the past, the total message count is proven to have great impact on the communicational cost [10]. Therefore, another objective of a partitioning method should be trying to minimize the total message count across the processors. Yet another objective may be achieving balance on the message count and communication volume handled by a processor. This work focuses on partitioning rectangular sparse-matrices using hypergraph models in such a way that after balancing the computational work per processor, the total

communication volume as well as total message count is minimized.

Two parallel communication schemes are investigated. The first one is named the *global communication scheme* in which all processors are involved in the same communication operation. The second one is named *personalized communication scheme* in which each processor is involved in more than one communication operation with some other processors.

The problem of minimizing the communication cost in the global communication scheme is reduced to the problem of minimizing the total and concurrent communication volume, due to the fact that the total message count is totally determined by the underlying interconnection topology in this scheme. The problem of minimizing the total and concurrent communication volume in this scheme is achieved by transforming the sparse matrix into a singly bordered (SB) block-diagonal form through non-symmetric row/column permutation. By minimizing the size of the border while maintaining balance on the nonzero counts of the row stripes of the matrix in the SB form, we minimize the total communication volume while maintaining computational load balance across the processors.

The problem of minimizing the communication cost in the personalized communication scheme is modeled as a two stage process. In the first stage, total communication volume is minimized and in the second stage, total message count is minimized. The problem in the first stage is solved using existing tools. The problem in the second stage is solved by proposing a novel communication hypergraph model. After building a clear correspondence between the communication requirements and the proposed hypergraph model, the problem is solved by partitioning the mentioned hypergraph. Two solutions are proposed. The first one solves the problem in this stage with a two phase process. In the first phase, the total message count is minimized and in the second phase communication loads of the processors are determined. The second solution minimizes total message count while determining the communication loads of the processors.

The organization of this thesis is as follows: Chapter 2 presents a brief description of hypergraphs and hypergraph partitioning problem. The terminology described in this chapter will be used throughout the thesis. The chapter also includes a brief summary of communication cost in the message passing systems. Chapter 3 defines the problem of minimizing total communication volume in the global scheme in terms of computational hypergraph partitioning. This chapter also defines the problem of minimizing the total communication volume as well as total message count in the personalized communication scheme as a special hypergraph partitioning problem. In this chapter, two solutions to the mentioned hypergraph partitioning problem will be presented. In Chapter 4, experimental results will be presented along with the comments on the proposed solutions. Finally, directions for future work and a conclusion will be presented.

Chapter 2

Preliminaries

In this chapter, after a brief discussion of communication in message passing systems, background on hypergraphs and hypergraph partitioning will be built. Iterative improvement hypergraph partitioning techniques and multilevel hypergraph partitioning techniques will be reviewed in order to clarify the discussion in the following chapters.

2.1 Communication in Message Passing Systems

Parallel applications running on a message passing system communicate results and problem parameters with explicit message-passing subroutines. The message passing performance is often measured in units of time, because it is a source of overhead when executing programs in parallel. The message passing time, t_n, is usually a linear function of message size for two communicating processors. It is modeled as

$$t_n = \alpha + \beta n + (h-1)\gamma \tag{1}$$

where α is the start-up time (latency), β is the per-byte transmission cost —often referred to as transfer time, γ is the per-hop delay, n is the size of the message in bytes and h is the number of hops a message must travel.

Due to efficient "worm-hole" routing [31] the per-hop delay becomes negligible and the first two terms are used to measure the cost of sending a message of size n bytes. In their work [10], Dongarra and Dunigan show the effect of latency in the message passing performance by examining various multiprocessor message passing systems. They show that for some architectures the latency is the dominating factor for messages of length smaller than 500-bytes. Therefore, one would like to minimize message count in his/her application in order to have an efficient parallel program.

2.2 Hypergraphs and Hypergraph Partitioning

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of nodes $\mathcal{V}$ and a set of nets $\mathcal{N}$ among those nodes. Every net n_i is a subset of nodes. The nodes in a net n_i are called its *pins* and denoted as $Pins(n_i)$. In this work this operator is extended to include the pin list of a set of nets $\mathcal{N}'$, i.e., $Pins(\mathcal{N}') = \bigcup_{n_i \in \mathcal{N}'} Pins(n_i)$. The size of a net is equal to the number of its pins, i.e., $s_i = |Pins(n_i)|$. The set of nets connected to a node v_j is denoted as $Nets(v_j)$, which is also extended to a set of nodes appropriately. The degree d_j of a node v_j is equal to the number of nets it is connected to, i.e.,$d_j = |Nets(v_j)|$. Weights can be assigned to the nodes. Let w_i denote the weight of the node v_i. The total number of pins, p, denotes the size of $\mathcal{H}$ where $p = \sum_{n_i \in \mathcal{N}} s_i = \sum_{v_j \in \mathcal{V}} d_j$.

$\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way partition of $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ if and only if the following three conditions are satisfied:

- $\mathcal{V}_k \subset \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$
- $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$
- $\mathcal{V}_k \cap \mathcal{V}_l = \emptyset$ for $1 \leq k < l \leq K$

The partitioning is sometimes referred to as bisection in the case of two-way partitioning. For $K > 2$ the partitioning is called multi-way or multiple-way

partitioning by some researchers.

In a partition Π of $\mathcal{H}$, a net is said to *connect* a part if it has at least one node in that part. *Connectivity set* Λ_i of a net n_i is defined as the set of parts connected by n_i. *Connectivity* $\lambda_i = |\Lambda_i|$ of a net n_i denotes the number of parts connected by n_i.

A net n_i is said to be *internal* if $\lambda_i = 1$. $\mathcal{N}_I^\Pi$ is said to be the set of those internal nets. Accordingly, a net n_i is defined to be *external* if $\lambda_i > 1$. The set of external nets of a partition Π is denoted as $\mathcal{N}_E^\Pi$. There are two metrics widely used in VLSI community [22] to define the cost of a partition, usually referred to as *cutsize*. The first one is referred to as the *cutnet metric* and counts the number of external nets as the cost of the partition Π, i.e.,

$$cost(\Pi) = |\mathcal{N}_E^\Pi| \tag{2}$$

The second one is referred to as the *connectivity metric* and counts the number of parts connected by external nets reduced by the number of external nets:

$$cost(\Pi) = \sum_{n_i \in \mathcal{N}_E^\Pi} (\lambda_i - 1) \tag{3}$$

The weight W_k of a part $\mathcal{V}_k$ is the sum of the individual weights of nodes in that part, i.e.,$W_k = \sum_{v_i \in \mathcal{V}_k} w_i$. A partition Π is said to be balanced if:

$$\frac{W_{max} - W_{avg}}{W_{avg}} \leq \epsilon \tag{4}$$

where W_{max} is the weight of the part with maximum weight and W_{avg} is the average part weight, and ϵ is a predetermined imbalance ratio.

Hence, the K-way hypergraph partitioning problem can be stated as finding a partition Π where $cost(\Pi)$ is minimized and the balance criterion stated by Eq. 4 is satisfied.

2.3 Computational Hypergraph Models

Computational graph model is widely used in the literature for the parallelization of various scientific applications including repeated sparse-matrix-vector-products through domain partitioning [18, 23, 24, 27, 31, 32, 33]. In this model, the problem of sparse matrix partitioning for minimizing communication volume while maintaining load balance is formulated as the well known K-way graph partitioning problem, where K denotes the number of processors in the target parallel architecture. Recently, two papers addressed the flaws of this model [6, 14]. First, graph model can only be used for symmetric square matrices. Second, the cost function in the graph model is only a weak approximation to the actual volume of communication. Computational hypergraph models are proposed by Çatalyürek and Aykanat [6] to remedy these flaws. The hypergraph models enable the representation and hence partitioning of rectangular matrices as well as symmetric and non-symmetric square matrices. Additionally, the hypergraph models capture the total volume of communication accurately. Henceforth, the partitioning problem has been reduced to the well known K-way hypergraph partitioning problem. In the rest of this chapter, these computational hypergraph models and their partitioning methods will be discussed.

Two computational hypergraph models for partitioning of sparse matrices are proposed by Çatalyürek and Aykanat [6]. The first one is named *column-net* model and the second one is named *row-net* model. The first one is used for rowwise partitioning (post-communication) and the latter one is used for columnwise partitioning (pre-communication). Here, the referred paper will be used to define the first model. The definition for the latter model can easily be acquired by replacing column by row and vice versa through the following discussion.

In the column-net model, sparse matrix $\mathbf{A}$ is represented as a hypergraph $\mathcal{H} = (\mathcal{V}_\mathcal{R}, \mathcal{N}_\mathcal{C})$ for rowwise partitioning. The elements of the set $\mathcal{V}_\mathcal{R}$ correspond to the rows of $\mathbf{A}$ and, the elements of the set $\mathcal{N}_\mathcal{C}$ correspond to the columns of $\mathbf{A}$. For each row i of $\mathbf{A}$ there exists a unique node $v_i \in \mathcal{V}_\mathcal{R}$. For each column j of $\mathbf{A}$ there exists a unique net $n_j \in \mathcal{N}_\mathcal{C}$ and, $v_i \in n_j$ if and only if $a_{ij} \neq 0$, i.e.,

for each nonzero entry in $\mathbf{A}$ there is a unique pin in $\mathcal{H}$. Each node $v_i \in \mathcal{V}_\mathcal{R}$ corresponds to the atomic task associated with row i. Each net $n_j \in \mathcal{N}_\mathcal{C}$ represents the dependency of atomic tasks on the columns. Particularly, in rowwise partitioning of sparse-matrix $\mathbf{A}$ for matrix-vector product $y = \mathbf{A}x$, $v_i \in \mathcal{V}_\mathcal{R}$ represents the atomic task of computing the inner product of row i with column vector x, with the proper weight equal to the number of nonzero elements of row i —which is, in turn, equal to d_i in $\mathcal{H}$. A net $n_j \in \mathcal{N}_\mathcal{C}$ represents the set of atomic tasks that need x_j.

A K-way partition Π of $\mathcal{H}$ induces a K-way partition of rows of $\mathbf{A}$ among K processors. Minimizing the cutsize according to Eq. 3 corresponds to minimizing the total communication volume and maintaining the balance criterion according to Eq. 4 corresponds to maintaining the computational load balance. Note that weak balance conditions enlarge the domain of possible partitions, hence can enable partitions with smaller costs than those of partitions found under strong balance conditions. Particularly, by not setting a balance constraint one can find a partitioning with a *minimal* cost, albeit the problem as stated above is proven to be *NP-hard* [12].

2.4 Hypergraph Partitioning Methods

There has been vast amount of work on partitioning hypergraphs. As a result of these works lots of partitioning methods have been devised. They can be classified as move-based approaches —including iterative improvement [11, 20, 29, 30] and mean-field annealing [4, 5] approaches—, geometric representations [3, 16], and multilevel approaches [6, 9, 16, 18]. multilevel approaches have been proven to be successful both in solution quality and run-time performance. The multilevel partitioning tool PaToH [6] is used in this thesis. It includes iterative improvement techniques. Therefore, for convenience the multilevel approaches and iterative improvement techniques will be discussed in the remaining of this chapter. For the rest of the approaches and many other approaches to hypergraph partitioning, the reader is referred to the excellent survey by Alpert and Kahng [2].

2.4.1 Iterative Improvement Approaches

Iterative improvement approaches start with a given feasible solution and move to a neighboring solution iteratively. The improvement process terminates when there is no better neighboring solution. In this case, the algorithm is said to be stuck in a local optima. In order to remedy this problem, the algorithms often include some hill-climbing paradigms.

Most of the iterative improvement algorithms are based on the famous Kernighan-Lin (KL) bisection heuristic proposed for graphs [19] which is later adapted to hypergraphs by Schweikert and Kernighan [30]. The KL heuristic defines a partition $\Pi^1 = \{p_1^1, p_2^1\}$ to be in neighborhood of another partition $\Pi^2 = \{p_1^2, p_2^2\}$ if Π^2 can be obtained from Π^1 by only exchanging position of two vertices belonging two different parts. The heuristic proceeds in a series of passes. At each pass, every vertex is swapped once to reach the best neighboring solution, i.e., the swap with the highest gain is tentatively realized. The gains due to these swaps are recorded and after all vertices are swapped once, the best solution encountered in this pass is returned. The gain of a move is defined as the decrease in the cutsize due to the move. Next pass is than started from the resultant bisection. KL heuristics climb out the local optima by considering swaps even with negative gain. Additionally, the heuristic is robust. It can be modified to partition into equal sized parts, to consider some vertices special than the others. However, it can handle only graphs with identically weighted vertices. It has a complexity of $O(n^2 \lg n)$ per pass where it is experimentally verified that an average KL algorithm converges in 2 to 4 passes [19].

Fiduccia-Mattheyses' Approach

Fiduccia and Mattheyses introduced their linear-time heuristics (FM) for hypergraph bipartitioning in [11]. They modified the neighborhood definition of the KL heuristic and thus gain formulation. A partition is said to be in neighborhood of another if one can be derived from the other by only one node move, and gain associated with moving a node from one part to another is

the reduction in the cutsize. Note that in the intermediate solutions FM allows deviations from the required balance on parts by the weight of the largest node. Like its ascendent KL heuristic, FM performs passes where each node can move exactly once and returns the best solution encountered during the pass and terminates when there is no improvement throughout a pass.

The heuristic runs in $O(p)$ time where p is the number of pins of the input hypergraph. The reduction in the runtime is achieved by utilizing bucket data structure for gains, thereby providing constant-time selection of best move and fast gain updates.

Krishnamurty's Approach

Krishnamurthy suggested that intelligent tie-breaking mechanisms are required in order to avoid FM to make bad choices [20]. Later, Hagen et al. [21] emphasized the importance of tie-breaking by stating the observation that on some VLSI benchmark data at any time there are many alternatives with the highest gain. As a tie-breaking mechanism Krishnamurthy introduced look-ahead ability to the conventional FM. A gain vector of size l is associated with each node, which is a sequence of gain values corresponding to possible l moves in the future, where l is the predetermined number of levels. The first level gain is the same as FM method. The second level gain of v_i is the possible reduction in cutsize in the next move following the move of node v_i, and so on. Thus the i^{th} level gain looks i moves ahead, and the ties are broken lexicographically by considering 1^{st} level gains, then by 2^{nd} level gains, and so on.

Sanchis' Approach

Sanchis extended the conventional FM with look-ahead ability to a multi-way hypergraph partitioning that directly partitions the given hypergraph into more than two parts [28] with cost metric given by Eq. 2. All the approaches prior to Sanchis's method (SN) were originally proposed as bipartitioning algorithms. In SN heuristic, at any time during the iterative refinement, a node can move from its current part to any other part, as long as this move will not violate

```
Input:H: A hypergraph
       K: number of partitions
begin
       (m, H_list) ← Coarsen(H);
       Π ← Initial_Partitioning (H_list[m]);
       Π_H ←Uncoarsening(H_list, m, Π);
end
```

Figure 2.1: Multilevel direct K-way partitioning

the balance conditions on parts.

She later enhanced her heuristic to work with the second cost metric [29]. Her heuristic for K-way partitioning of hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ looking l-levels ahead runs in $O(lpK(\lg K + d_{max}.l))$ and $O(p(s_{max} + Kl + K^2)(\lg K + d_{max}))$ for cost metrics (2) and (3), respectively, where p is the number of pins in $\mathcal{H}$, $d_{max} = max_{v_i \in \mathcal{V}}(d_i)$ and $s_{max} = max_{n_j \in \mathcal{N}}(s_j)$.

2.4.2 Multilevel Approaches

The performance of FM-like algorithms deteriorates for large and sparse hypergraphs. They work well on hypergraphs with high node degrees and low net sizes. Another important fact is that the solution quality of FM is not predictable, i.e., the average solution's quality is significantly worse than the best solution's quality. These are the motivations directed researchers to multilevel approaches.

As seen in Fig. 2.1 multilevel approaches have three phases: *coarsening*, *initial partitioning*, and *uncoarsening*. In the first phase, the input hypergraph is coarsened by coalescing highly interacting nodes into multinodes forming a coarser hypergraph. The coarsening operation proceeds coarsening the hypergraph till the number of nodes in the coarsened hypergraph reduces below a predetermined threshold value. By the end of this phase, node degrees become larger than the original hypergraph. In the second phase, coarsest graph is

partitioned into K parts using various heuristics. In the third phase, the initial partition found in the second phase is projected back towards the input hypergraph, and refined on intermediate hypergraphs using FM-like heuristics.

Coarsening Phase

In this phase, the given hypergraph $\mathcal{H} = \mathcal{H}_0 = (\mathcal{V}_0, \mathcal{N}_0)$ is coarsened into a sequence of smaller hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1), \mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2), \ldots, \mathcal{H}_m = (\mathcal{V}_m, \mathcal{N}_m)$ satisfying $|\mathcal{V}_0| > |\mathcal{V}_1| \cdots > |\mathcal{V}_m|$. This coarsening is achieved by coalescing disjoint subsets of nodes of hypergraph $\mathcal{H}_i$ into multinodes such that each multinode in $\mathcal{H}_i$ forms a single node in $\mathcal{H}_{i+1}$. The weight of each node of $\mathcal{H}_{i+1}$ becomes equal to the sum of its constituent nodes of the respective multinode in $\mathcal{H}_i$. Accordingly, the net set of each node of $\mathcal{H}_{i+1}$ becomes equal to the union of the net sets of the constituent multinodes in $\mathcal{H}_i$.

The skeleton of the coarsening algorithm is shown in Fig. 2.2. The clustering algorithms used in coalescing can be classified as *agglomerative* and *hierarchical*. In agglomerative clustering, new clusters are formed one at a time, whereas in hierarchical clustering several new clusters may be formed simultaneously. The hierarchical clustering scheme is often called matching-based clustering. The reason is that, it matches two nodes in $\mathcal{H}_i$ according to some criterion to form a multinode in $\mathcal{H}_{i+1}$. That is, the nodes that are selected are at the same level of clustering. The algorithms in the latter category can choose a node in $\mathcal{H}_i$ to be involved in a previously formed multinode in $\mathcal{H}_{i+1}$.

Initial Partitioning Phase

The goal of this phase is to partition the coarse enough hypergraph generated in the previous phase. Researchers employed a variety of initial partitioning algorithms including random partitioning [15] and bin packing-like methods [1, 6].

Input: $\mathcal{H}_0$: A hypergraph
Output:m:number of coarsening levels
$\mathcal{H}_{list} = \mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$: hypergraph list
begin
$m \leftarrow 0$;
$\mathcal{H}_{list} \leftarrow \emptyset$;
while ($\mathcal{H}_m$ is not coarser enough) **do**
$\mathcal{H}_{list} \leftarrow \mathcal{H}_{list} \cup \mathcal{H}_m$
$m \leftarrow m + 1$;
$\mathcal{H}_m \leftarrow$ Cluster($\mathcal{H}_{m-1}$);
end
return $\langle m, \mathcal{H}_{list} \rangle$
end

Figure 2.2: Coarsening Algorithm

Uncoarsening Phase

As seen in Fig. 2.3 the uncoarsening phase proceeds as follows. At each level i (for $i = m, m-1, \ldots, 1$), partition Π_i found on $\mathcal{H}_i$ is projected back to a partition Π_{i-1} on $\mathcal{H}_{i-1}$. The constituent nodes of each multinode in $\mathcal{H}_i$ is assigned to the part of the respective node in $\mathcal{H}_i$. Naturally, the partitions Π_{i-1} and Π_i have the same cutsizes. Then, the new partitioning is refined with an FM-like algorithm.

Input: $\mathcal{H}_{list} = \mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_m$: hypergraph list
m: number of uncoarsening levels
$\Pi_{\mathcal{H}_m}$: a partitioning on $\mathcal{H}_m$
Output: $\Pi_{\mathcal{H}_0}$: a partitioning on $\mathcal{H}_0$
begin
$\Pi \leftarrow \Pi_{\mathcal{H}_m}$
while $(m \geq 0)$ **do**
DoRefinement$(\mathcal{H}_m, \Pi)$
$m \leftarrow m - 1$;
$\Pi \leftarrow$ Project$(\Pi, \mathcal{H}_m, \mathcal{H}_{m+1})$
end
return Π
end

Figure 2.3: Uncoarsening Algorithm

Chapter 3

Models and Solution Methods

In this chapter Interior Point Methods containing repeated matrix-vector and matrix-transpose-vector multiplications will be introduced. The communication cost in the parallelization of these applications depends on the communication scheme followed. The communication in the parallel implementation of these methods can take place in two schemes. In the first scheme, all of the processors are involved in a global communication on some equal sized sparse vectors. In the second scheme, the sparsity of the vectors can be exploited and communication can be restricted to some sets of processors. Note that in the first scheme all of the processors are involved in the same communication operation. Hence, this scheme is referred to here as *global communication scheme.* In the second scheme, the communication takes place among some sets of processors where each processors can be included in more than one set communicating on distinct vectors. Therefore, this communication scheme is referred to here as *personalized communication scheme.*

The communication cost of the global communication scheme is minimized in Section 3.2 by reducing the size of the communicated vector. The minimization problem regarding the cost of global communication scheme is solved by minimizing the total communication volume. The total message count can not be changed in the global communication scheme. The communication requirements in the parallelization of these algorithms are further investigated and the total communication cost is minimized by following the personalized

communication scheme where the total message count and total communication volume is minimized. The problem of minimizing the communication cost in the personalized communication scheme will be modeled as a two stage process, where total communication volume is minimized in the first stage and the total message count is minimized in the second stage. Another objective of the second stage is set to be balancing the communication loads of the processors. Although first stage of the problem is solved using an existing hypergraph partitioning tool PaToH [6], one can use any other hypergraph partitioning tool which minimizes the total communication volume. The problem in the second stage is modeled with a special hypergraph called *communication hypergraph*, and we come up with two partitioning schemes that solve the communication cost minimization problem. The first one solves the problem in this stage with a two phase process. In the first phase —trying to be a generic solution— it utilizes PaToH The tool used in this phase is not tuned to work with the communication hypergraph, it just gives the total message count and a lower bound on the total communication volume, where the problem of assigning message loads to processors is not solved. In the second phase, the scheme uses weighted maximal matching in bipartite graphs to solve the assignment problem arising in the first phase. For the second scheme, we have tuned PaToH to work on the communication hypergraph that will be defined to solve the problem.

3.1 Applications and Problem Definition

Interior point methods are widely used to solve the linear programming (LP) problem

$$\begin{aligned} &\textit{minimize } c^T x \\ &\textit{subject to } \mathbf{A}x = b, \\ &x \geq 0, \end{aligned}$$

where, c, x are real N-vectors, b is a real M-vector, and $\mathbf{A}$ is a real $M \times N$ matrix of rank M, with $M \leq N$. At each iteration of the methods, the search

direction is determined. This comprises the bulk of the work done [34]. The search direction is computed by solving the set of equations

$$\begin{bmatrix} D & \mathbf{A}^T \\ \mathbf{A} & 0 \end{bmatrix} \begin{bmatrix} \Delta e \\ \Delta f \end{bmatrix} = \begin{bmatrix} w \\ v \end{bmatrix}$$

where f is the dual variable and D is the diagonal matrix that changes at each iteration. Alternatively, the normal equations

$$(\mathbf{A}D^{-2}\mathbf{A}^T)\Delta f = r$$

can be solved at each iteration. Wang and O'Leary [34] discuss many iterative as well as direct methods for the solution of these equations.

In the iterative solution of normal equations, two basic types of operations are performed at each iteration of the iterative solver. These are linear operations on dense vectors and the matrix-vector product of the form $y = (\mathbf{A}D^{-2}\mathbf{A}^T)x$ which is computed as a sequence of matrix-transpose-vector product $z = D^{-2}\mathbf{A}^T x$ and matrix-vector product $y = \mathbf{A}z$. The D^{-2} matrix changes at each iteration of the interior-point method and hence the $D^{-2}\mathbf{A}$ matrix is computed only once at the beginning of each solution of the normal equations by the iterative solver. Since D^{-2} is a diagonal matrix, $D^{-2}\mathbf{A}$ has the same sparsity pattern as $\mathbf{A}^T$. Thus, matrix D^{-2} is omitted while considering computational and communication requirements in the parallelization of the iterative solver.

Our goal is the parallelization of the computation in the conjugate gradient-like iterative solver through one dimensional rowwise or columnwise partitioning of the $M \times N$ matrix $\mathbf{A}$. In the rowwise partitioning of matrix $\mathbf{A}$, matrix $\mathbf{A}^T$ is effectively partitioned columnwise, i.e.,

$$\mathbf{A} = \begin{bmatrix} A_{1*} \\ \vdots \\ A_{k*} \\ \vdots \\ A_{K*} \end{bmatrix} \qquad \text{and} \qquad \mathbf{A}^T = [A_{1*}^T \cdots A_{k*}^T \cdots A_{K*}^T],$$

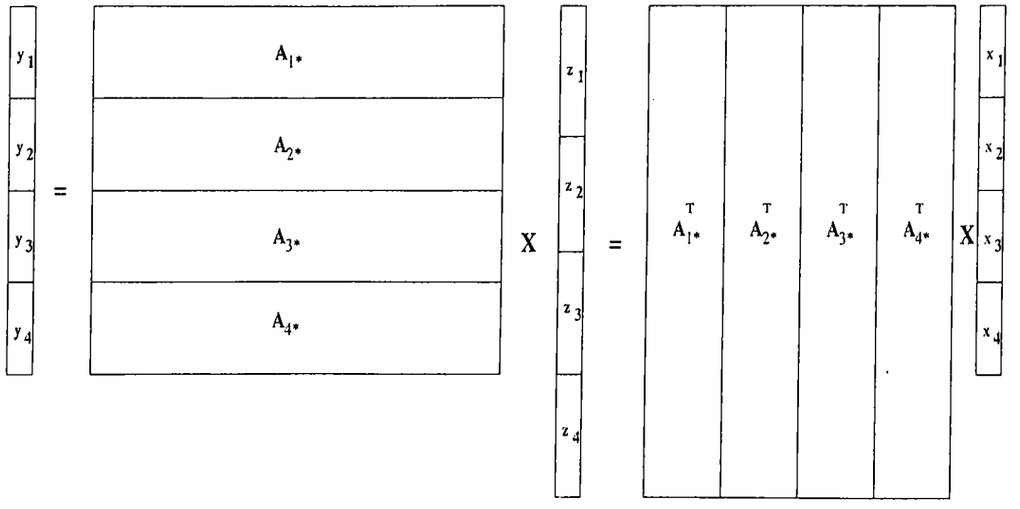

Figure 3.1: Overall partitioning of A for $K = 4$

where processor P_k holds the kth row stripe A_{k*} of $\mathbf{A}$ thus also holding the kth column stripe A_{k*}^T of $\mathbf{A}^T$. In order to avoid the communication of vector components during the linear vector operations, a symmetric partitioning scheme is adopted. That is, all vectors (each of size M) used in the solver are partitioned conformally with the row partitioning of $\mathbf{A}$ which is equivalent to the column partitioning of $\mathbf{A}^T$. In particular, the y and x vectors of size M are partitioned as $[y_1, \cdots, y_k, \cdots y_K]^T$ and $[x_1, \cdots, x_k, \cdots x_K]^T$, respectively. The z vector of size N is also K-way partitioned as $[z_1, \cdots, z_k, \cdots z_K]^T$ for the parallelization of matrix-transpose-vector product and matrix-vector product computations. Note that the z vector is an intermediate vector which is not involved in the linear vector operations and its size is in general different than the size of the vectors used by the iterative solver (e.g., y and x). So, the partitioning of the z vector is totally independent from the partitioning of the y and x vectors. The overall rowwise partitioning will be as shown in Fig. 3.1 for $K = 4$ processors. In this scheme, processor P_k is responsible for computing the local matrix-transpose-vector product $z^k = A_{k*}^T x_k$, where $z = \sum_{k=1}^{K} z^k$, and the local matrix-vector product $y_k = A_{k*} z$. Processor P_k is also responsible for computing the linear operations on the kth blocks of the vectors. With this partitioning scheme, the linear vector operations in the iterative solver can be easily and efficiently parallelized such that only the inner-product computations introduce global communication of whose volume does not scale up with increasing problem size. The matrix-transpose-vector product $z = \mathbf{A}^T x$

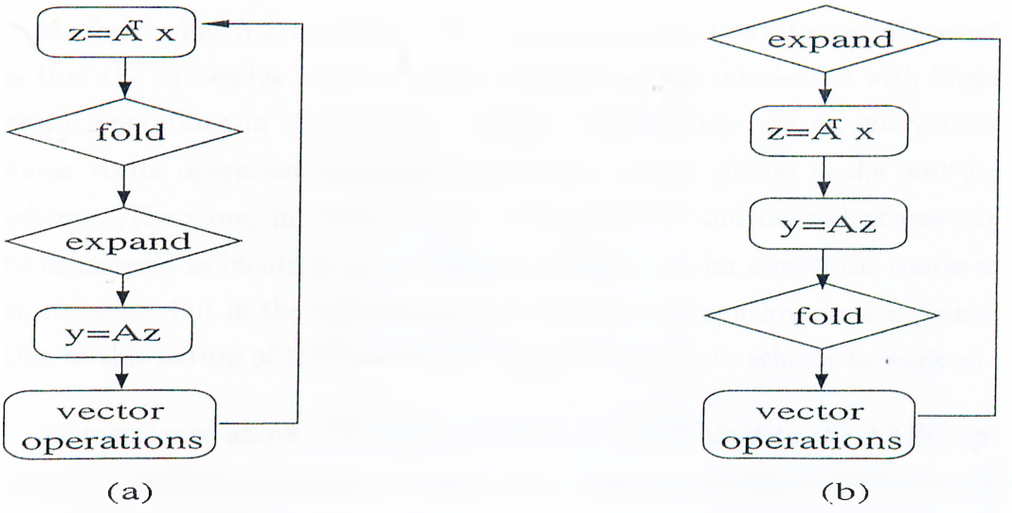

Figure 3.2: Two parallel computation schemes: (a) Post-pre scheme, (b) Pre-post scheme

requires communication just after the local matrix-transpose-vector product computations and the following matrix-vector product $y = \mathbf{A}z$ requires communication just before the local matrix-vector product computations. Thus, this parallelization scheme based on the rowwise partitioning of matrix $\mathbf{A}$ is also referred to here as *post-pre* communication scheme. The communication requirement in the post-pre scheme is as follows. Each processor P_k holds a local sparse vector z^k of size N and needs to know the resulting vector $z = \sum_{k=1}^{K} z^k$ of size N. This operation can be performed efficiently by a sequence of fold and expand communication steps; which effectively correspond to the post and pre-communication steps, respectively. In the fold communication operation, each processor P_k has a local vector z^k of size N and needs to gather the kth block z_k of resultant vector z. To accomplish the expand operation, an *all-to-all broadcast* (AABC) operation is needed on the local z_k vectors to provide each processor with a copy of the global z vector.

The parallelization scheme based on the columnwise partitioning of the $\mathbf{A}$ matrix is just the dual of that of the rowwise partitioning scheme. In this respect, this scheme is referred to here as the *pre-post* scheme, where the operations performed in the pre and post-communication steps are expand and fold operations, respectively. The flow diagrams of the post-pre and pre-post schemes are illustrated in Fig. 3.2.

As seen in Fig. 3.2, the major difference between the two schemes mentioned is that the successive fold and expand operations are interleaved with linear vector operations in the pre-post scheme, whereas there are no interleaving linear vector operations between the fold and expand phases in the post-pre scheme. Therefore, in the post-pre scheme the fold and expand phases can be considered as incurring only one synchronization point during the course of an iteration, but in the pre-post scheme there are two synchronization points. Due to this nature of the schemes we choose the post-pre scheme to work on.

As mentioned above, the expand operation is equivalent to the AABC operation. Since the communication pattern of the fold operation is the dual of the expand operation, the communication cost of the fold operation is same as that of the AABC operation. In the standard approach the fold and expand operations are performed on the global z vector of size N. Therefore, the total communication volume of the post-pre scheme is $2N(K-1)$ words, where half of the volume is due to the fold operation and the other half is due to the expand operation independent of the underlying interconnection network topology. The total message count varies with the interconnection topology. For example, the total message count is $2K(K-1)$, $4K(\sqrt{K}-1)$, and $2K\log K$ in ring, square mesh, and hypercube topologies, respectively. Hence, the concurrent communication cost of a global fold-expand operation becomes:

$$\begin{aligned} T_{comm} &= t_{su} \times 2(K-1) &+ t_w \times 2N(K-1)/K, \\ &= t_{su} \times 4(\sqrt{K}-1) &+ t_w \times 2N(K-1)/K, \\ &= t_{su} \times 2\log K &+ t_w \times 2N(K-1)/K \end{aligned}$$

in ring, square mesh, and hypercube topologies, respectively, where t_{su} represents the start-up cost and t_w represents the transmission-time per word.

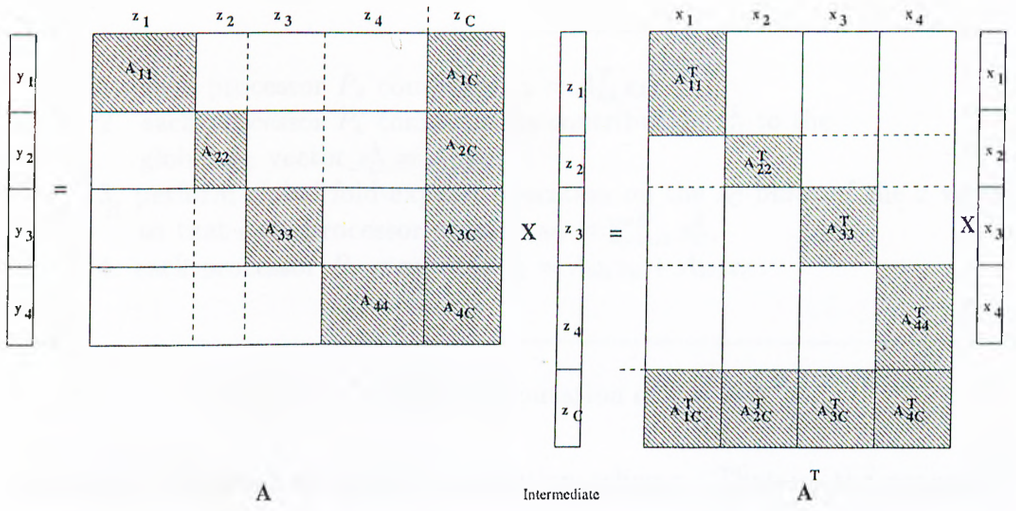

Figure 3.3: 4-way rowwise partitioning of **A** permuted in singly bordered form.

3.2 Minimizing Communication Cost in Global Fold-Expand Scheme

In this thesis, we show that minimizing the communication volume in the global fold-expand operation can be achieved by transforming matrix **A** into a singly bordered block-diagonal form (SB) through non-symmetric row/column permutation. Consider a 4-way SB form of the **A** and $\mathbf{A}^T$ matrices, where nonzero entries exist only in the shaded submatrices as shown in Fig. 3.3. The columns of the matrix aligned with z_C are coupling columns, i.e., the nonzeros in these columns span more than one row stripe. Each processor P_k holds the diagonal block A_{kk} of size $M_k \times N_k$ and border block A_{kC} of size $M_k \times N_C$ and kth slice of the x vector, i.e., x_k of size M_k, and hence the data blocks A_{kk}^T and A_{kC}^T with the responsibility to compute the kth slice of the y vector, i.e., y_k of size M_k. Note that $\sum_k M_k = M$ and $N_C + \sum_k N_k = N$. To accomplish its duty, each processor has to multiply its matrix blocks with the vector blocks that are aligned with these blocks. That is, the processor P_k has to know the x_k, z_k, and z_C, but not the whole z vector.

In the light of above discussion, the operations proceed as shown in Fig. 3.4. Note that in the proposed scheme the global fold-expand operation is restricted to the local z_C^k vectors of size N_C instead of the local z vectors of size N as

1. each processor P_k computes $z_k = A_{kk}^T x_k$,
2. each processor P_k computes its contribution z_C^k to the global z_C vector $z_C^k = A_{kC}^T x_k$,
3. perform global fold-expand operation on the z_C block of the z vector so that each processor obtains $z_C = \sum_{k=1}^{K} z_C^k$,
4. each processor P_k computes $y_k = A_{kk} z_k + A_{kC} z_C$.

Figure 3.4: Parallel computation of $y = \mathbf{A}\mathbf{A}^T x$

performed in the conventional computation scheme. That is, the proposed scheme reduces the concurrent communication volume from $(K-1) \times N/K$ words of the conventional scheme to $(K-1) \times N_C/K$ words. Thus, the problem of minimizing the communication cost in the global scheme can be defined as transforming the $\mathbf{A}$ matrix into an SB form such that the border size N_C is minimized while maintaining a balance on the nonzero counts per row stripes of the matrix $\mathbf{A}$. In recent works [7, 25, 26], the problem of permuting rectangular matrices into SB form is formulated as a hypergraph partitioning problem. In this formulation, the computational hypergraph model (column-net version) discussed in Section 2.3 is used. In this model, rows of matrix $\mathbf{A}$ (columns of $\mathbf{A}^T$) correspond to the nodes and the columns of matrix $\mathbf{A}$ (rows of $\mathbf{A}^T$) correspond to nets of the hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$. Each node of the hypergraph can be weighted by the nonzero entry count of the respective row of matrix $\mathbf{A}$. Consider a K-way partitioning $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of $\mathcal{H}$. Π can be decoded as a partial permutation on the rows and columns of matrix $\mathbf{A}$ to induce a permuted matrix $\mathbf{A}_{SB}$. In this permutation, the rows associated with the nodes in $\mathcal{V}_{k+1}$ are ordered after the rows associated with the nodes in $\mathcal{V}_k$, for $k = 1, 2, \ldots, K-1$. The columns associated with the nets internal to the nodes in $\mathcal{V}_{k+1}$ are ordered after the columns associated with the nets internal to the nodes in $\mathcal{V}_k$, for $k = 1, 2, \ldots, K-1$, where the columns associated with the external nets, $\mathcal{N}_\Pi^E$, are ordered last as the coupling columns. That is, a node $u_i \in \mathcal{V}_k$ corresponds to permuting row i of $\mathbf{A}$ to the kth slice of $\mathbf{A}_{SB}$. An internal net n_j of $\mathcal{V}_k$ corresponds to permuting column j of $\mathbf{A}$ to the kth column slice of $\mathbf{A}_{SB}$, and an external net n_j corresponds to permuting column j of $\mathbf{A}$ to the border of $\mathbf{A}_{SB}$. In the partitioning of the associated hypergraph,

minimizing the cutsize according to Eq. 2 corresponds to minimizing the border size N_C while maintaining a given balance according to Eq. 4 corresponds to maintaining the computational load balance during both matrix-vector product and matrix-transpose-vector product computations.

3.3 Minimizing Communication Cost in Personalized Communication Scheme

The mentioned global fold-expand scheme has two disadvantages. First, the startup cost due to the total message count is totally determined by the underlying interconnection topology. Second, the communication volume may be unnecessarily high due to the redundant communication.

Consider a close examination of the parallel algorithm shown in Fig 3.4. Only the nonzero segments of border matrix $\mathbf{A}_{kC}^T$ will incur nonzero entries in the local z_C^k vector computed by processor P_k at step 2. That is, the z_C^k vectors are likely to be sparse vectors, so that the $z_C = \sum_{k=1}^{K} z_C^k$ operation performed during the global fold operation at step 3 is effectively a global sparse vector addition operation. In other words, processor P_k does not have to participate in the fold operation on the zero-valued components of its z_C^k vector. In a dual manner, for the computation of the $A_{kC}z_C$ product each processor P_k needs to know the resultant z vector components that correspond to the nonzero column segments of $\mathbf{A}_{kC}$ which in turn corresponds to the nonzero row segments of $\mathbf{A}_{kC}^T$. For example in Fig. 3.5, only processors P_2 and P_3 should be involved in the computation of the cth component of the z vector during the fold operation. In a dual manner, only these processors need to know the accumulated result during the expand operation. These facts can be exploited to devise a *personalized communication scheme* to perform fold and expand operations as follows. For each entry of the z_C vector, a processor P_k is assigned the responsibility of gathering and adding all the partial results for that entry in the fold phase, where the processors that have partial results on this entry send their results to processor P_k. In the expand phase processor P_k also becomes responsible for sending the accumulated result back to the

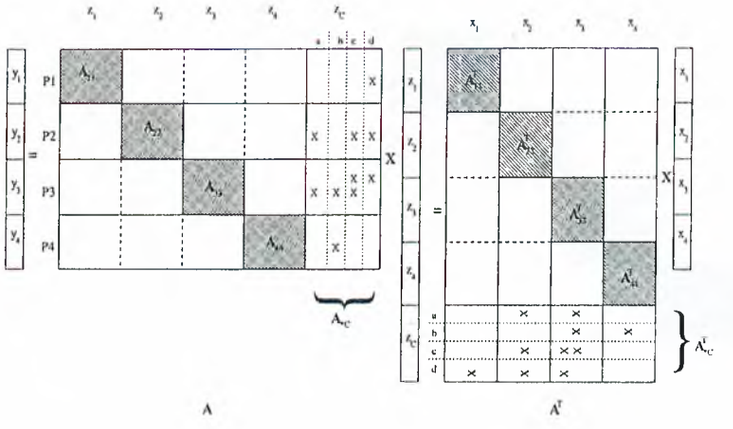

Figure 3.5: 4-way rowwise partitioning of **A**: detailed single bordered from

processors that need the final result —namely the processors that sent the partial results in the fold phase. For example, in Fig. 3.5, processor P_1, P_2, and P_3 should be involved in the fold-expand operation on dth entry of z_C. If processor P_1 is assigned the responsibility of folding the $d$the entry then it gathers and adds the results from processors P_2 and P_3 in the fold phase and sends the accumulated result back to processors P_2 and P_3 in the expand phase. Hence, the personalized fold-expand scheme has the flexibility of reducing the total message count while avoiding the redundancy in communication.

The problem of minimizing the communication volume in the personalized fold-expand scheme can also be defined as permuting matrix **A** into a K-way SB form as follows. As in the global fold-expand scheme, a given balance criterion should be maintained on the nonzero counts per row stripes of matrix **A** in order to maintain computational load balance during the local matrix-vector product and matrix-transpose-vector product computations. Different from the global scheme, the total number of nonzero column segments in the $\mathbf{A}_{kC}$ border submatrices should be considered for minimization. Consider the assumption that the fold-expand responsibility of each entry of the z_C vector is assigned to a processor that generates a partial result for that entry. Under this assignment constraint, each coupling column that spans λ row stripes of **A** contributes $\lambda - 1$ words to the total communication volume.

The problem of permuting matrix **A** into an SB form with the desired property for the personalized communication scheme can be defined as a hypergraph

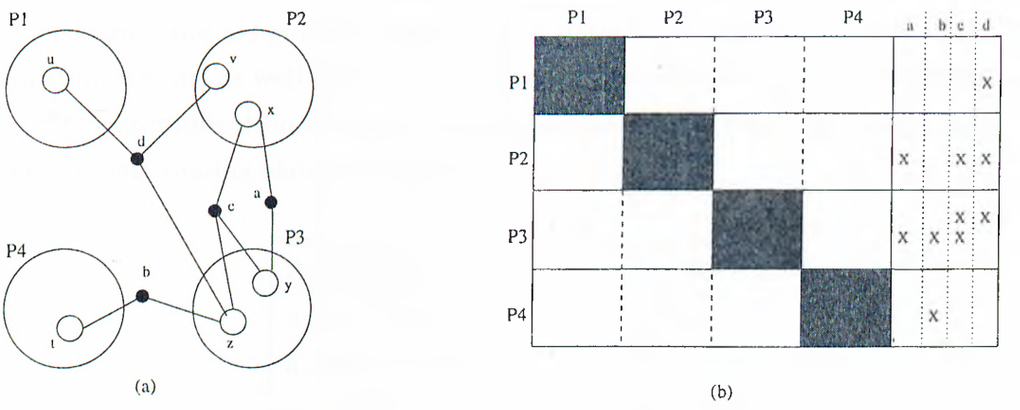

Figure 3.6: A 4-way partition of a hypergraph: (a) hypergraph and a partitioning (b) associated sparse matrix and 4-way decomposition

partitioning problem on the column-net model as described in Section 3.2 for the global communication scheme. However, in this case, the cutsize definition given in Eq. 3 should be considered for minimization instead of Eq. 2. Thus, minimizing the cutsize will correspond to minimizing the total communication volume under the above mentioned assignment restriction, and maintaining the given balance criterion will correspond to balancing the computational loads of the processors as in the case of global communication scheme..

Since the personalized communication scheme depends on the nonzero pattern of the partitioned and permuted matrix, to continue with the discussion we have to assume a partitioning on the matrix **A**. Due to its superiority in representing the actual communication requirements, the computational hypergraph model is assumed to be used in the partitioning of the input matrix.

A toy example of a computational hypergraph partitioning is seen in Fig. 3.6 along with its correspondence on the partitioned matrix. In this particular example $cost(\Pi) = \sum_{n_i \in \mathcal{N}_E^{\Pi}} (\lambda_i - 1) = 5$, which is equal to the total communication volume in both the fold and expand phases. Note that this equality is true under the assignment constraint mentioned above. For example, in Fig. 3.6(a) if net d is assigned to part P_3 then partial results generated by P_1 and P_2 must be sent to processor P_3, where 2-words of communication volume will incur in fold phase, and hence 2-words of communication volume will incur in expand phase due to messages from P_3 to P_1 and P_2.

Actually, the assignment of cut nets to parts determines the total communication volume as well as the total message count and and it is not considered in [6] for minimization of these quantities. Consider the following assignments for the partitioning shown in Fig. 3.6:

Assignment	V	M
$a, b \rightarrow P_1; d, c \rightarrow P_4$	9	6
$a, b, d, c \rightarrow P_3$	5	3
$a \rightarrow P_1; b \rightarrow P_4; d, c \rightarrow P_2$	6	5
$b \rightarrow P_4; d, a \rightarrow P_1; c \rightarrow P_2$	6	5

where V and M represent the total communication volume and the total message count both in the fold and expand phases, respectively. As seen from the above table, the total communication volume and the total message count can assume varying values under a given partitioning according to the assignment of cut nets to parts. Particularly, it increases if a net is assigned to a part that is not connected by that net.

Note the following facts concerning the personalized communication used in post-pre scheme.

FACT 1 *The total communication volume and the total message count sent in the fold phase is equal to the total communication volume and the total message count sent in the expand phase.*

Specifically, if a processor P_i sends a message to P_j in the fold phase containing its partial result on some element of z_C, then it will receive a message from P_j containing the folded result in the expand phase to compute $A_{kC} z_C$.

FACT 2 *Although, the computational load of the processor is the same in the matrix-vector product and matrix-transpose-vector product operations, the communication load of a processor is not the same in the fold and expand phases.*

In the case of partitioning square matrices a cut net n_j in the associated hypergraph is assigned to the part that holds v_j, —prerequisiting the existence

of nonzero diagonal entries. This is too restrictive in the applications that do not impose symmetric partitionings on the columns and rows of the input matrix. In those applications, we can exploit the same flexibility as in the rectangular matrices to minimize the communication cost. There is not any clue in the literature about the assignment of cut nets to parts in case of partitioning rectangular matrices. What should be done is to pay tribute to the effect of communication on efficiency and to consider it as a problem that must be attacked rigorously, which is the subject of the following sections.

In the following sections, we propose a two-stage approach for solution of communication cost minimization problem in personalized communication scheme where the total communication volume is minimized in the first stage and the total message count is minimized in the second stage. The objectives in the first stage is balancing the computational loads of the processors and setting a lower bound on the total communication volume. For this purpose, we use the computational hypergraph model (column-net version) of Çatalyürek and Aykanat and the hypergraph partitioning tool PaToH [6] to find a partition Π to induce an SB form on the associated sparse matrix. The objective in the second stage is to find a fold-expand responsibility for the nets that are cut by the partition Π found in the first stage to minimize the total message count while maintaining balance on communication loads of the processors. For this purpose, we propose the following communication hypergraph model and reduce the problem of the second stage to the hypergraph partitioning problem on the communication hypergraphs.

3.3.1 Communication Hypergraph Model

Given a partition $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ of a computational hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, the communication hypergraph is defined as follows. $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$ where $|\mathcal{V}_C| = |\mathcal{N}_E^{\Pi}|$ and $|\mathcal{N}_C| = K$. That is, for each $n_i \in \mathcal{N}_E^{\Pi}$ we have a distinct node in $\mathcal{H}_C$, and for each part $\mathcal{V}_j$ we have a distinct net in $\mathcal{H}_C$. The node $v_i \in \mathcal{V}_C$ corresponding to $n_i \in \mathcal{N}_E^{\Pi}$ resides in the pin list of $n_j \in \mathcal{N}_C$ if and only if $n_i \in \mathcal{N}_E^{\Pi}$ connects part $\mathcal{V}_j$ under the partitioning Π. That is $\sum_{v_i \in \mathcal{V}_C}(d_i) = \sum_{n_j \in \mathcal{N}_E^{\Pi}}(\lambda_j)$. Also, note that since the nodes in $\mathcal{V}_C$ correspond to cut nets in

the partitioned computational hypergraph, they have size of at least 2. The nodes in the communication hypergraph has weights associated with them, representing the communication volume due to this net in the computational hypergraph, which is equal to the connectivity of the corresponding cut net in the computational hypergraph. By this setting, the communication hypergraph $\mathcal{H}_C$ has the following relation to the input matrix $\mathbf{A}$. The rows of the matrix $\mathbf{A}^T$ that are in the border —a, b, c, d in Fig. 3.6(b) —necessitating folds and expands on the corresponding entries in z_C, comprises the nodes of $\mathcal{H}_C$. The row blocks in $\mathbf{A}$ are condensed in a single net, and the pin list of this net is set according to the nonzeros in the border blocks in the corresponding row block, meaning that some rows in this block require the entries in the x-vector that correspond to the column indices whose corresponding nets are cut by the partition.

Assume a partition $\Pi_C = \{\mathcal{V}_{C1}, \mathcal{V}_{C2}, \ldots, \mathcal{V}_{CK}\}$ where $n_i \in \mathcal{N}_C$ is the net that represents the processor that part $\mathcal{V}_i$ is associated in $\mathcal{H}_C$. Recall that s_i represents the sizo of the net n_i, $Pins(n_k)$ represents the nodes that are in the pin list of net n_k, and $Nets(\mathcal{V}_i)$ represents the nets that are connected to the nodes in the node set $\mathcal{V}_i$. Then this partitioning determines the distribution of communication as follows.

- the communication volume handled by processor $P_k = V_k^F + V_k^E$, where
  - V_k^F: is the communication volume sent by processor P_k during the fold phase. It is equal to $s_k - |Pins(n_k) \cap \mathcal{V}_{Ck}|$,
  - V_k^E: is the communication volume sent by processor P_k during the expand phase. It is equal to $\sum_{u \in \mathcal{V}_{Ck}}(d_u) - |Pins(n_k) \cap \mathcal{V}_{Ck}|$.
- the total communication volume during both the fold and expand phases is equal to $\sum_{k=1}^{k=K} s_k - |Pins(n_k) \cap \mathcal{V}_{Ck}|$,
- the message count handled by processor $P_k = M_k^F + M_k^E$, where
  - M_k^F: is the number of messages sent by processor P_k during the fold phase. It is equal to $\lambda_k - 1$,
  - M_k^E: is the number of messages sent by processor P_k during the expand phase. It is equal to $|Nets(\mathcal{V}_{Ck}) - n_k|$.

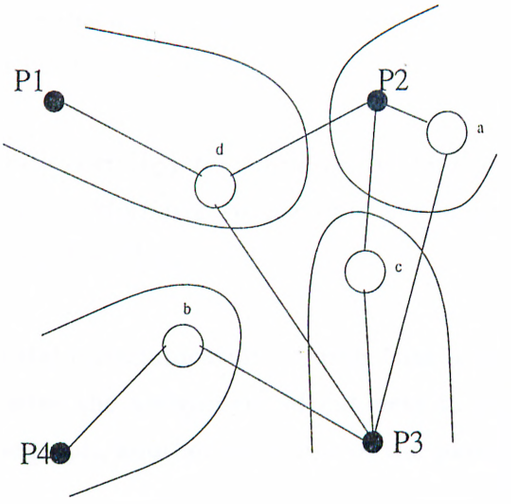

Figure 3.7: A partition in a communication hypergraph

- the total message count during both the fold and expand phases is equal to $\sum_{k=1}^{K} M_k^F$ ($\sum_{k=1}^{K} M_k^E$).

Note that the total message count in the fold phase is equal to $\sum_{n_i \in \mathcal{N}_C}(\lambda_i - 1)$ which is equal to $cost(\Pi_C)$. Due to Fact 1, it is also equal to the total message count in the expand phase.

The communication hypergraph of the toy example of Fig. 3.6 and a partitioning is seen in Fig. 3.7. In the figure P_1, P_2, P_3, and P_4 are set to be responsible for fold operations on d, a, c, and b entries of vector z_C, respectively. The communication load of the processors after this partitioning becomes:

part	fold		expand	
	V	M	V	M
p_1	-	-	2	2
p_2	2	2	1	1
p_3	3	3	1	1
p_4	-	-	1	1

where the total communication volume (V) and the total message count (M) in both the fold and expand phases are 5 words and 5, respectively.

In the light of above discussion the objectives of partitioning a communication hypergraph can be set as follows:

Objective 1 *Minimize cost*(Π_C) *to minimize the total message count in the fold and expand phases. Minimize connectivity set of nets,* λ_k, *in* $\mathcal{H}_C$, *and minimize nets of parts,* $Nets(\mathcal{V}_{Ck})$, *in* Π_C.

Observing that the total communication volume, that is minimized in the first stage, can increase after the assignment of cut nets to parts by partitioning communication hypergraph, another objective of the partitioning is set to be:

Objective 2 *For each part, minimize the number of pins running out of this part and running out of its associated net to other nets and parts, respectively.*

Objective 3 *Maintain balance on the communication volume handled by the processors.*

Objective 1 above stands for the minimization of the total message count in the system as well as message count handled by a single processor in the fold and expand phases. Objective 2 stands for minimization of total communication volume. The last objective is required to handle the concurrent communication volume. If the balance on the communication volume is not imposed, then trivial but deficient solutions such as assigning all the messages and hence all the communication volume to a subset of processors can be proposed. These solutions are deficient in the sense that the bottleneck processor's communication volume will be too high, and therefore the overall cost of the communication volume will not be reasonable.

3.4 Partitioning Computational Hypergraph

In this stage of the algorithm, we resort to the heuristic algorithms employed in PaToH in order to partition the input hypergraph. The objectives that are realized in this stage are balancing the computational loads of the processors

and setting a lower bound on the total communication volume. For this purpose, the column-net hypergraph that models the rectangular sparse matrix **A** is partitioned while the cost of the partitioning stated by Eq. 3 is minimized.

The data structure used to store a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ in PaToH [6] mainly consists of the following arrays. The $NETLST$ array stores the net lists of nodes. The $PINLST$ array stores the pin lists of nets. The sizes of both arrays are equal to the total number of pins in the hypergraph. Two auxiliary index arrays $VTXS$ and $NETS$ of sizes $|\mathcal{V}| + 1$ and $|\mathcal{N}| + 1$ hold the starting indices of the net lists and pin lists of the nodes and nets in the $NETLST$ and $PINLST$ arrays, respectively. The total storage requirement due to a hypergraph with M nodes,N nets, and Z pins —corresponding to column net model of an $M \times N$ sparse matrix with Z nonzero elements— is $2M + 2N + 2Z$ words, where M and N words are used to store the weights of nodes and nets respectively, another M and N words are used to store $VTXS$ and $NETS$, and $2Z$ words are used to store the arrays $NETLST$ and $PINLST$.

3.5 Partitioning Communication Hypergraph Using Existing Tools

Our first solution to the communication cost minimization problem in personalized communication scheme uses functions provided by the multilevel hypergraph partitioning tool PaToH's library. That is, we use PaToH as a black box. The skeleton of the overall algorithm can be seen in Fig. 3.8. After partitioning the associated computational hypergraph, the computational loads of the processors are determined while a lower bound on the total communication volume is established. The communication cost is minimized in two phases, i.e., partitioning and assignment, as seen in Fig. 3.8 at first and second steps, respectively. In the partitioning phase, the total message count is bounded from below. Then, in the assignment phase, exact communication loads of the processors are determined while trying to achieve the lower bound set on the total communication volume during partitioning of computational hypergraph.

and setting a lower bound on the total communication volume. For this purpose, the column-net hypergraph that models the rectangular sparse matrix **A** is partitioned while the cost of the partitioning stated by Eq. 3 is minimized.

The data structure used to store a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ in PaToH [6] mainly consists of the following arrays. The $NETLST$ array stores the net lists of nodes. The $PINLST$ array stores the pin lists of nets. The sizes of both arrays are equal to the total number of pins in the hypergraph. Two auxiliary index arrays $VTXS$ and $NETS$ of sizes $|\mathcal{V}| + 1$ and $|\mathcal{N}| + 1$ hold the starting indices of the net lists and pin lists of the nodes and nets in the $NETLST$ and $PINLST$ arrays, respectively. The total storage requirement due to a hypergraph with M nodes,N nets, and Z pins —corresponding to column net model of an $M \times N$ sparse matrix with Z nonzero elements— is $2M + 2N + 2Z$ words, where M and N words are used to store the weights of nodes and nets respectively, another M and N words are used to store $VTXS$ and $NETS$, and $2Z$ words are used to store the arrays $NETLST$ and $PINLST$.

3.5 Partitioning Communication Hypergraph Using Existing Tools

Our first solution to the communication cost minimization problem in personalized communication scheme uses functions provided by the multilevel hypergraph partitioning tool PaToH's library. That is, we use PaToH as a black box. The skeleton of the overall algorithm can be seen in Fig. 3.8. After partitioning the associated computational hypergraph, the computational loads of the processors are determined while a lower bound on the total communication volume is established. The communication cost is minimized in two phases, i.e., partitioning and assignment, as seen in Fig. 3.8 at first and second steps, respectively. In the partitioning phase, the total message count is bounded from below. Then, in the assignment phase, exact communication loads of the processors are determined while trying to achieve the lower bound set on the total communication volume during partitioning of computational hypergraph.

Input: $\mathcal{H}_C$: A communication hypergraph
Π: a partitioning on the associated computational hypergraph
K: number of partitions
Output: Π_C: Communication load decomposition
begin
$\Pi_C = PATOH\_RecursivePart(\mathcal{H}_C, K)$;
$DoAssignment(\Pi, \Pi_C, K)$;
return Π_C;
end

Figure 3.8: First partitioning scheme: Using existing tools

Phase 1: Partitioning Communication Hypergraph

In this phase, PaToH is fed with the communication hypergraph constructed from the computational hypergraph and its partitioning. The weights of the nodes in the communication hypergraph are set as described in Section 3.3.1. Note that this setting is true if a node resides in a part that it is not connected to after the partitioning. That is, the weighting scheme employed here is just an approximation. We resort to this approximation because before assigning the fold and expand responsibilities by partitioning the communication hypergraph, we do not know the exact communication volume requirements.

Theorem 3.1 *The lower bound for the total communication volume during both the fold and expand operation is equal to the cost of the partition,* Π*, of the computational hypergraph. The upper bound is equal to* $cost(\Pi) + |\mathcal{N}_E^{\Pi}|$.

Proof. By definition, $cost(\Pi)$ of the partition Π of the computational hypergraph gives the total communication volume accurately if and only if each cut net, n_j is assigned to a part that it is connected to, say P_k, necessitating the communication between the processor pairs $\langle P_k, P_i \rangle$ for $P_i \in \Lambda_j \wedge P_k \neq P_i$. That is we have $\lambda_j - 1$ volume of communication due to the net n_j. Assigning to any other part, $P_k \notin \Lambda_j$ will necessitate communication between the processor pairs $\langle P_k, P_i \rangle$ for $P_i \in \Lambda_j$. Thus, in this case we have λ_j volume of communication due to net n_j. Considering all of the cut nets $\mathcal{N}_E^{\Pi}$ we

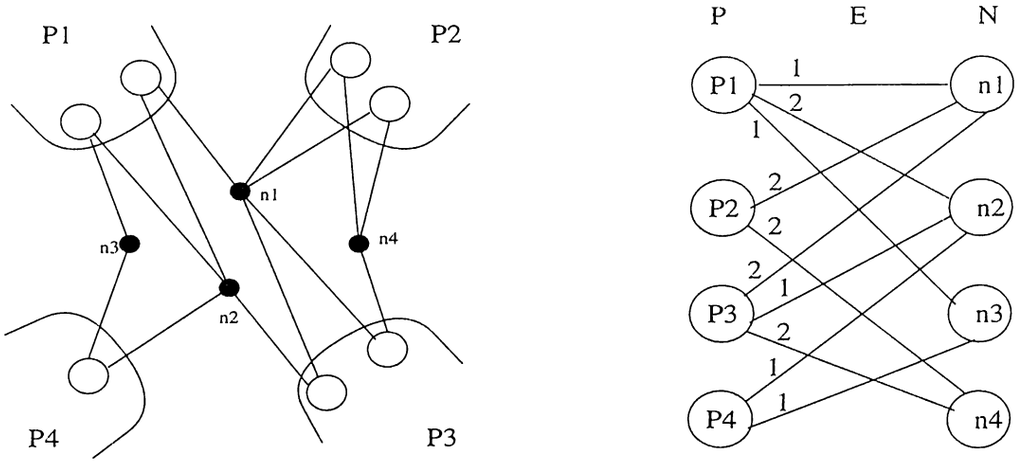

Figure 3.9: Phase 1 result: (a)Partition in $\mathcal{H}_{comm}$ (b) Corresponding weighted bipartite graph

have a communication volume of at least $\sum_{n_i \in \mathcal{N}_E^\Pi}(\lambda_i - 1) = cost(\Pi)$, and at most $\sum_{n_i \in \mathcal{N}_E^\Pi}(\lambda_i) = \sum_{n_i \in \mathcal{N}_E^\Pi}(\lambda_i - 1) + \sum_{n_i \in \mathcal{N}_E^\Pi}(1)$ which is in turn equal to $cost(\Pi) + |\mathcal{N}_E^\Pi|$.□

Phase 2: Assignment

Partitioning tool used in the previous phase does not solve the problem of assigning messages to the processors, because the part numbers generated in this phase are not related to the part numbers generated in the previous phase. So, we must somehow assign the parts generated in this phase to the processors. A particular instance of communication hypergraph partitioning can be seen in Fig. 3.9(a).

Obviously, the naive approach that matches part $\mathcal{V}_k$ of the first stage to part $\mathcal{V}_{Ck}$ of the second stage does not work. For the example in Fig. 3.9(a) it results in 9 messages and communication volume of 13 in both the fold and expand phases, whereas $cost(\Pi) = 7 = \sum_{v_i \in \mathcal{V}_{comm}}(d_i - 1)$, and $cost(\Pi_{comm}) = 6$. So, concerning this particular example, one can not say that Objectives 1 and 2 are achieved by this naive approach. What should be done is to match each part generated in the second stage to the nets in this stage, thus to the parts of the previous stage, with the above objectives in mind. The first can be

achieved by assigning each cut net in this stage to a part it is connected to. The second objective can be achieved by assigning each cut net to a part that it is maximally connected, while enabling the realization of first objective. If a net is assigned to a part that is not connected to, then additional messages will incur. To have all processors involved in communication in the fold and expand phases to impose balance on communication loads of the processors, we have to match each net to a unique part. This problem is the same as the maximum weighted perfect matching problem in weighted bipartite graph shown in Fig. 3.9(b). To clarify the problem and its solution, we need the following definitions.

Definition 3.1 *Two edges in a graph $G = (V, E)$, where V being the set of vertices, and E being the set of edges, are said to be* **independent** *if and only if they share no vertices.*

Definition 3.2 *A* **matching** *M in a graph is a set of independent edges. The edges in M are called* **matched edges**. *The vertices that are incident to the edges in M are said to be* **matched** *by M.*

Definition 3.3 *A matching is called* **perfect matching** *if it matches all the vertices in the graph.*

Definition 3.4 *A perfect matching M in a weighted graph is called* **maximum weighted perfect matching** *if the sum of the weights of edges in M is maximum among all perfect matchings.*

Definition 3.5 *A bipartite graph $G = (U \cup V, E)$ is called a* **complete bipartite graph** *if and only if there exist an edge between all $(u \in U, v \in V)$ pairs.*

The maximum weighted perfect matching problem in bipartite graphs can be solved by the *Kuhn-Munkres* algorithm [8], having a run time complexity of $O(V^4)$ on a bipartite graph with $2V$ vertices. The algorithm requires

an input of complete bipartite graph with positive weights on edges, and returns a maximum weighted perfect matching in the input bipartite graph. The input complete bipartite graph $G = (V \cup U, E)$ to the algorithm is constructed from the communication hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$ and partitioning $\Pi_C = \{\mathcal{V}_{C1}, \mathcal{V}_{C2}, \ldots, \mathcal{V}_{CK}\}$ as follows:

1. $V = v_1, v_2, \ldots, v_K$ where v_k representing $\mathcal{V}_{Ck} \in \Pi_C$,
2. $U = u_1, u_2, \ldots, u_k$ where u_i representing $n_i \in \mathcal{N}_C$,
3. the weight of the edge $\langle v_i, u_j \rangle$, $w(u_i, v_j)$, is set to $|Pins(n_j) \cap \mathcal{V}_{Ci}|$.

Note that the vertices in set V correspond to the parts in Π_C, i.e., $|V|$ is equal to K, and the vertices in set U correspond to the nets of $\mathcal{H}_C$ and hence to the parts in Π, i.e., its size is also K.

By the above setting, the weight of the existing edges are set to their corresponding values in $\mathcal{H}_C$, and the missing ones are set to 0, which should be positive. In order to tune the algorithm to penalize the matchings that will cause additional messages, we adjust weights of the edges assigned in Item 3 as follows;

4. $w(u_i, v_j) = \begin{cases} w(u_i, v_j) + t_\alpha & \text{if } w(u_i, v_j) > 0 \\ 1 & \text{if } w(u_i, v_j) = 0 \end{cases}$

where t_α is the ratio of the time elapsed to send a unit length message to the transfer time of sending a unit length message. By the help of this adjustment, we can say that the algorithm is biased towards matching the nets to the parts that they are connected under the partition Π_C.

The matching M generated by the Kuhn-Munkres algorithm is interpreted by considering each edge $\langle v_i, u_j \rangle \in M$ as an assignment of the fold-expand responsibilities of the nodes in $\mathcal{V}_{Ci}$ to the processor P_j. Thus, the total communication volume that will incur during the fold and expand phases will be equal to $\sum_{e \in E} w(e) - \sum_{e_m \in M} w(e_m)$ and the total message count will be equal to $cost(\Pi_{comm}) + |\{e : e \in M \wedge w(e) = 1\}|$. If M contains only the edges that

are incremented by t_α, then the cost of the partition Π_C becomes equal to the total message count.

Theorem 3.2 *The total message count during the fold and expand operations after this assignment is bounded below by the cost of the partition* Π_C *and bounded above by* $\Pi_C + K$

Proof. By construction, each net, n_i, in $\mathcal{H}_C$ represents the entity that needs the data depending on the nodes in $Pins(n_i)$. If this net is matched with any one of them by the Kuhn-Munkres algorithm then the processor, associated with this net, will communicate with $|Pins(n_i)| - 1$ processors. If, it is matched with any other part, then the communication will be among $|Pins(n_i)| + 1$, resulting in $|Pins(n_i)|$ messages in fold phase. Summing this lower and upper bounds for all nets, we get the mentioned bounds on total message count.□

Analysis

Partitioning communication hypergraphs with PaToH according to the cut-size metric stated by Eq. 3 successfully minimizes the total message count in the personalized fold and expand scheme. Because, the cost function minimized here exactly corresponds to the total message count. Unfortunately, the balance criterion maintained by the algorithm is not proper. The functions provided in PaToH's library maintain balance on the sum of the weights of nodes in each part. Recall that the weights associated with the nodes of the communication hypergraph denote the communication volume due to respective node if the fold-expand responsibility of the associated entry in z_C is assigned to a part it is connected to. The partitioning algorithm can assign a node to a part that it is not connected to in order to reduce the cost of the partition. In fact, since the relation of nets to parts is not exploited, the nodes that are assigned to part $\mathcal{V}_{Ck}$ can be connected by some nets other than the one representing $\mathcal{V}_k$ of the partitioning found on the computational hypergraph. Hence, the overall algorithm discussed here, tries to achieve balance on something different than the required one.

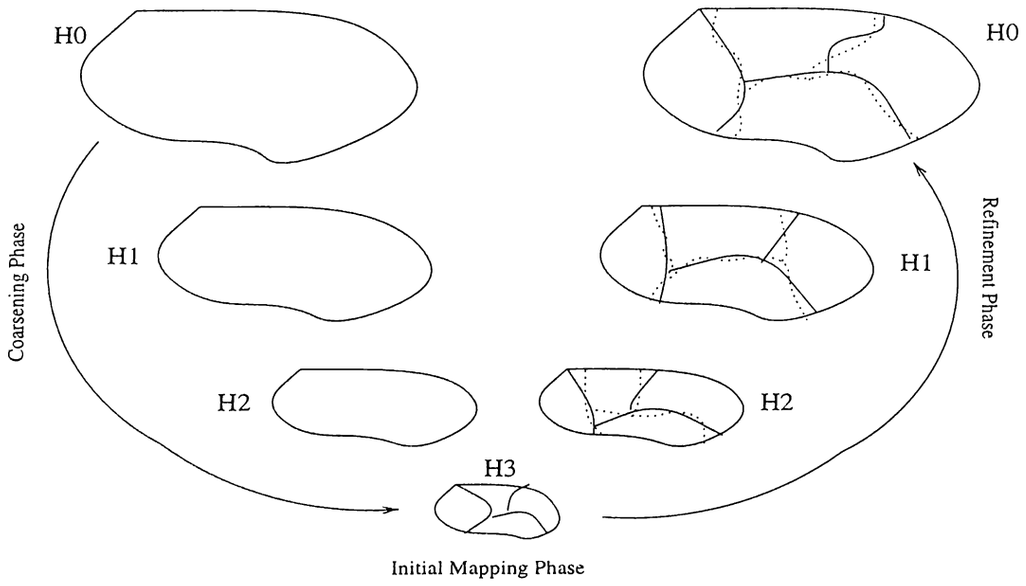

Figure 3.10: Three phases of multilevel K-way partitioning algorithm. During the coarsening phase size of the input hypergraph is successively reduced; an initial mapping is then found for K-parts during initial mapping phase; the partitioning found is then refined while it is projected towards the initial hypergraph. Solid lines representing the partitions, dashed lines representing the refinement took place.

3.6 New Partitioning Tool for Communication Hypergraphs

Our second scheme is a novel approach to hypergraph partitioning problem, in which the standard K-way partitioning algorithms are tuned to work on communication hypergraphs. That is, the algorithm recognizes the relationship of nets to parts that are obtained after partitioning the computational hypergraph. Our algorithm is a multilevel approach. Like other multilevel approaches we have coarsening, initial mapping, and refinement phases. Here, the input communication hypergraph is coalesced using highly tuned coarsening algorithms with Objectives 1, 2, 3 stated above in mind. The coarsest communication hypergraph is partitioned into K parts in initial mapping phase. Then, this initial mapping is projected to finer hypergraphs and refined with tuned K-way refining algorithms. The algorithm is sketched in Fig. 3.10.

The data structure used to store a communication hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$

is composed of those mentioned in Section 3.4 and two arrays parallel to $NETLST$ and $PINLST$ arrays where the weights of the pins are stored. Hence, the total memory requirement used by a communication hypergraph becomes $2|\mathcal{V}_C| + 2|\mathcal{N}_C| + 4Z$ words, where Z is the size of the communication hypergraph.

Coarsening Phase

While coarsening the input communication hypergraph we pay special attention to avoid the nets become internal to a multinode in the coarser hypergraphs. The rationale behind this restriction is that, if a net becomes internal to a multinode then it will tend to become internal to a part at the end of partition in which case the processor that corresponds to this net will be idle during the fold phase. Another modification to the standard coarsening phase is to maintain the weights of the pins in coarser hypergraphs. Like the standard one, the weight of a multinode is set to the sum of the weights of its constituent nodes and multiple pins to this multinode are contracted to a single pin. Unlike the standard one, the weights of the pins in the coarser hypergraph is set to the sum of the weights of the contracted pins. This serves to realize Objective 2.

In order to realize the Objective 1 we would like to minimize the number of pins running out a node, as well as a net in the coarser hypergraph. We try to achieve this goal by employing the following metrics during the selection of nodes to be coalesced.

Heavy Connectivity Metric (HCM). While considering the node u to be coalesced, choose the node v giving *heavy connectivity* $N_{u,v} = |Nets(u) \cap Nets(v)|$. The rationale behind this metric is to match two nodes into a multinode where $N_{u,v}$ nets are common to both of its constituent nodes. The hierarchical clustering method using HCM works as follows. Nodes of the hypergraph $\mathcal{H}_i$ are visited in a random order. If a node $u \in \mathcal{V}$ has not been matched yet, one of its unmatched *adjacent* nodes that will not create an internal net is selected with the highest $N_{u,v}$ value. If there is no such unmatched adjacent node, then node u remains unmatched. The agglomerative clustering that employs $N_{u,v}$

metric is also viable.

Scaled Heavy Connectivity Metric (SHCM). This metric is a modification to the heavy connectivity metric. For two nodes u and v it is given by $\frac{N_{u,v}}{d_u+d_v-N_{u,v}}$. The rationale behind this metric is to coalesce two nodes into a multinode where most of pins running out of the multinode is common to both of its constituent nodes. The hierarchical clustering method using the SHCM works like the one using HCM. Slight modifications like selecting the lightest node v among the set $\{v : Nets(u) \subseteq Nets(v)\}$ while considering the node u and allowing the cluster weights grow up till a predetermined threshold value is added to tune the clustering algorithm for communication hypergraphs. Again, agglomerative version of this clustering scheme is viable.

Initial Mapping Phase

In this phase of the algorithm, we find a K-way partitioning of the coarsest hypergraph such that the communication volume handled by a single processor is balanced, and the total communication volume is minimized by trying to achieve lower bound set while partitioning the computational hypergraph. This is possible because of the proper maintenance of node weights as well as pin weights.

There are many alternative ways to find an initial partitioning. One of them is to continue the coarsening operation till the coarsest hypergraph has K nodes, in which case one-to-one mapping of these nodes to parts is established. This scheme is said to be problematic [17] because of two reasons; first one is the degeneration of the run time of the coarsening algorithm due to smaller reduction in the size of the hypergraphs in each coarsening level, second one is the unbalanced initial mapping due to inflexibility when there is only one node per part. We find the initial partitioning with two schemes. In the first one, we use PaToH's recursive bisection algorithm. In this case, we again encounter the problem of assigning nets to parts in the resultant partitioning. At this point, instead of solving the assignment problem optimally with Kuhn-Munkres algorithm discussed above, we resort to a greedy algorithm which maps each part generated by the recursive bisection algorithm to a distinct

net with highest connectivity. The reasons are the following. First, Kuhn-Munkres algorithm is not reasonable in terms of run time efficiency. Second, in case of multilevel algorithms optimum initial partitionings do not necessarily lead to optimum solutions [13]. In our second scheme, we use a linear time greedy algorithm which assigns parts to nets. In this scheme we generate part numbers directly from the net indices, avoiding the necessity of assignment phase.

Refinement Phase

During the refinement phase, a partitioning of the coarsest hypergraph is successively projected back to the next finer hypergraph, and this projected partitioning is refined by considering the communication hypergraph partitioning objectives set in Section 3.3.1.

We have designed two schemes to refine a communication hypergraph partitioning. The first one is an extension to Sanchis' approach [29], where the second one follows novel approaches to refine a communication hypergraph and its partitioning. The basic steps of both schemes are seen in Fig. 3.11. We have utilized the same gain concept that is used by Sanchis [29] in both of the schemes. Therefore, gain-initialization is the same in both of the approaches which can be seen in Fig. 3.12. Note that the algorithm we have developed is different than the one used by Sanchis. We were able to contribute on the algorithm's running time complexity. Original scheme runs in $O(pK)$-time, whereas this is a loose upper bound on the complexity of our algorithm. The two schemes differ in the steps where a node is selected to move and the gain update is accomplished. The first one uses priority queues to do the selection, hence update operations are performed on priority queues. However, the second one does not utilize that priority queues. The gain-update operation becomes easier with this second approach. Unfortunately, selection becomes more complicated. Gain-update operations in both approaches require the same steps seen in Fig. 3.13. The difference is that in the first one we adjust the priority queues, but in the second one since we are not using priority queues this operation is limited to updating the values of gains, their locations

```
begin
    while (more gain possible)
        InitializeGains();
        Unlock all nodes;
        while (no more unlocked nodes)
            u ← SelectNode(to);
            from ← part[u];
            UpdateGains(u, from, to);
            lockNode(u);
end
```

Figure 3.11: K-way refinement algorithm

are untouched.

For a given node u, $part[u]$ represents the partition that the node is currently assigned to, $weight[u]$ refers to the weight of the node. $G[u \rightsquigarrow t]$ represents the improvement in the cost when u moves to the part t. This gain corresponds to the decrease in the total message count after the move according to the cost metric given in Eq. 3. For a given net n, $pins[n,t]$ refers to the number of nodes belonging to n that are assigned to part t. Positive $pins[n,t]$ value represents a message from processor n to processor t during the fold phase, because of the duality in the fold and expand phases, it also fixes a message from t to n in the expand phase. $MessagesFold[n]$ represents the number of messages sent by processor n during the fold phase. It is equal to the number of nonzero entries in the n^{th} row of the *pins* data structure, which is equal to λ_n. $MessagesExpand[n]$ refers to the number of messages sent by processor n during the expand phase. It is equal to the number of nonzero entries in n^{th} column of *pins* data structure. $Volume[t]$ denotes the total communication volume that will be handled by processor t. This information is set prior to the K-way refinement algorithm and is kept as invariant throughout the refinement by necessary adjustments in the $UpdateGains()$ function sketched in Fig. 3.13.

```
begin
    for each node u ∈ V do
        for t = 1 to K do
            G[u ⇝ t] ← −d_u;
    for each node u ∈ V do
        g[u] ← 0;
        s ← part[u];
        for each net n ∈ nets[u] do
            if pins[n, s] = 1 then
                g[u] ← g[u] + 1;
        for each part t ∈ λ(n) do
                G[u ⇝ t] ← G[u ⇝ t] + 1;
        for t = 1 to K do
            G[u ⇝ t] ← G[u ⇝ t] + g[u];
end
```

Figure 3.12: K-way gain-initialization algorithm

Extension to Sanchis' Approach

Sanchis' approach requires $K(K-1)$ priority queues, $K-1$ for each of the parts, because a node in a part can move to any other part during the refinement. In the original algorithm proposed in [28] each of the $K(K-1)$ priority queues are searched to find the move with maximum gain value that is feasible under the balance condition. After the move, each of the $K(K-1)$ queues are updated. These operations on all of the priority queues make the algorithm impractical. What we do instead of considering all of the priority queues is selecting a part to reduce its communication cost, both in terms of the communication volume and message count. After figuring out the most heavily loaded processor, we consider moving nodes either from this processor —thus enabling the reduction in the communication volume as well as reduction in the message count sent by this processor during the expand phase—, or moving nodes that reduce the connectivity of the corresponding net to decrease the number of messages sent by this processor during fold phase. Among, the alternatives we choose the one with highest gain. In case of ties we choose the node which reduces the communication volume handled by this processor above all. In this framework

```
UpdateGains(node u, partition from, partition to)
begin
    pin_weight_to ← 0;
    pin_weight_from ← 0;
    pin_weight ← 0;
    for each net n ∈ nets[u] do
        pin_weight ← pin_weight + weightOf(v, n);
        if n = from then
            pin_weight_from ← weightOf(v, n);
        else if n = to
            pin_weight_to ← weightOf(v, n);
        for each node v ∈ pins[n] ∧ v ≠ u do
            if pins[n, from] = 1 then
                G[v ⇝ from] ← G[v ⇝ from] − 1;
            if pins[n, from] = 2 ∧ part[v] = from then
                for each part t ≠ from do
                    G[v ⇝ t] ← G[v ⇝ t] + 1;
        pins[n, from] ← pins[n, from] − 1;
        pins[n, to] ← pins[n, to] + 1;
        if pins[n, from] = 0 then
            MessagesFold[from] ← MessagesFold[from] − 1;
            MessagesExpand[n] ← MessagesExpand[n] − 1;
        if pins[n, to] = 1 then
            MessagesFold[n] ← MessagesFold[n] + 1;
            MessagesExpand[to] ← MessagesExpand[to] + 1;
        for each node v ∈ pins[n] ∧ v ≠ u do
            if pins[n, to] = 1 then
                G[v ⇝ to] ← G[v ⇝ to] + 1;
            if pins[n, to] = 2 ∧ part[v] = to then
                for each part t ≠ to do
                    G[v ⇝ t] ← G[v ⇝ t] + 1;
    Volume[from] ← Volume[from] + pin_weight_from;
    Volume[to] ← Volume[to] − pin_weight_to + pin_weight;
end
```

Figure 3.13: K-way gain-update algorithm

we consider at most $2K$ priority queues, and then update only $K-1$ priority queues.

The operations on priority queues are limited to decrease-key, increase-key, delete-key operations. Decrease-key and increase-key operations are necessary during the gain updates. Delete-key operation is used to delete the node moved. We do not have an insert-key operation due to the fact that, a node can change its part during a pass of the refinement algorithm only once. That is, we do not insert the moved node to the priority queues of its new partition. This is somehow an implementation of locking mechanism. At the beginning of each pass, the priority queues are constructed where each node is residing in $K-1$ priority queues of its current part, meaning that the nodes are unlocked. After moving a node we delete it from all $K-1$ priority queues. Via this setting —since we consider priority queues during selection of nodes to move— we provide locking mechanism implicitly.

Implementation without Priority Queues

In this approach, we do not utilize priority queues to select the nodes to be moved. What we do instead is to select the part with maximal message count to be sent in fold and expand phases. We then determine the bottleneck phase according to this part. If the bottleneck one becomes to be the fold phase, then we try to reduce the connectivity of the net associated with this part by moving the unique node of this net in any of the other parts to any of the connected parts, if any. The one with the maximum gain is selected to be moved to the target part where the associated gain is realized. Ties are broken in favor of target parts with minimum message count. To reduce the number of messages sent during the expand phase, we have to move single pins of other nets in the maximally loaded part to any of the connected parts. Note that if we were not able to reduce the number of messages during the fold phase, we would also try to reduce the number of messages in expand phase. This operation is performed on the basis of following criterion:

- Choose the node with maximum gain among the unmoved nodes in the maximally loaded part,

- in case of ties choose
  - the node that is not connected to the net of the maximally loaded part,
  - the node moving to the part with least message count.

Note that the node that is selected to move does not have to have positive gain. Unless, the algorithm will stuck into a local optima, where it can not get out. We try to provide our algorithm escape from the local optima by first considering the pins of nets that have only one pin in the bottleneck part. If no node is found, then we look both of the pins of nets that have two pins, and so on. This is somehow an approximate implementation of employing higher level gains. Updating the gains is same as the one using priority queues, except the priority queue operations. In this approach, we again maintain the locking mechanism by disregarding nodes that are removed from the lists.

Operations on Finest Communication Hypergraph

In the last step of the refinement phase after refining the initial partitioning for the original communication hypergraph we further refine the partitioning by considering all moves with nonnegative gains subject to the constraint that the moved node should not be connected to the net associated with its current partition. Our aim is to reduce the total communication volume further. A node residing in a part that it is not connected to means an increase in the total communication volume. So, on behalf of the parts that hold such nodes, the communication volume handled by these processors is minimized.

Analysis

The algorithm discussed here starts deviating from the standard hypergraph partitioning in the initial mapping phase, where the parts are numbered according to the nets. The deviation then continues in the refinement phase where the relation of nets to parts is used to determine the communication loads of the processors by the formulation given in Section 3.3.1 instead of the

sum of weights of the nodes in the associated part. Note that the communication cost given by the formulation is decomposed as the message count and the communication volume, because of missing machine dependent variables start-up cost and transmission time per word. Hence, in the algorithm the total message count is considered to be the entity that is minimized. The minimization of the total communication volume is tried to be accomplished by assigning node u_i to a part whose associated net, say n_k, is in $Nets(u_i)$, which enables deviating from the lower bound set on the total communication volume in Theorem 3.1, as small as possible. Since the minimization of the total communication volume is considered as secondary objective the algorithm imposes a weak balance criterion on the communication volumes of the processors to enable further reductions in the partitioning cost —namely, the total message count.

The refinement scheme that uses priority queues tries to achieve balance on communication volumes while minimizing the total message count. The second refinement scheme tries to achieve balance on message counts while minimizing the total message count. These are possible because of the tie breaking strategies employed in the refinement algorithms. The first one breaks the ties by considering the communication volumes of the parts, whereas the second one breaks the ties by considering the message counts of parts.

Chapter 4

Experiments and Results

In this chapter, results of various experiments that have been conducted are presented in order to validate the usage of the communication hypergraph model and evaluate the quality of the proposed solution methods to the communication problem. The first section describes the data set used in experimental verification. Then, the implementation details of the algorithms are presented. The next section compares the solution quality of our methods against a naive solution. Then, observations related to our solutions will be presented.

4.1 Data Set

The proposed model and the associated solution methods are tested on various Linear Programming problems obtained from various sources such as MatrixMarket[1], University of Florida Sparse Matrix Collection[2]. Table 4.1 represents the characteristics of the matrices used in the experiments.

Note that columns and rows of the matrices correspond to the nets and nodes of the associated computational hypergraph in column-net model. Hence, the table also characterizes the properties of the hypergraphs we have used.

[1] `http://math.nist.gov/MatrixMarket/`
[2] `http://www.cise.ufl.edu/davis/sparse/`

Matrix	Number of		
	Rows	Columns	Nonzeros
CO9	10789	14851	101578
CQ9	9278	13778	88897
fome12	24284	48920	142528
fxm4_6	22400	30732	248989
GE	10099	11098	39554
kent	31300	16620	184710
mod2	34774	31728	165129
NL	7039	9718	41428
pltexpA4_6	26894	70364	143059
world	34506	32734	164470

Table 4.1: Properties of test matrices

The test matrices are partitioned for 16, 32, and 64 ways using the column net model and communication hypergraphs are generated—see Appendix for computational load imbalance values. Table 4.2 characterizes the average-case properties of communication hypergraphs that are generated and subsequently used to verify the proposed communication hypergraph model and associated methods.

The computational hypergraphs are partitioned 5 times using different random seeds in order to evaluate the average case quality of the proposed methods. Hence, Table 4.2 represents the average characteristics of the communication hypergraphs for different number of partitions. Also note that the number of pins is equal to the minimized cost metric during the partitioning of computational hypergraph reduced by the number of nets in the communication hypergraph, e.g. for CO9 we can state that average case partitioning of the matrix results in a total volume of communication of 11984.6, 17755, and 25463.8 words for 16, 32, and 64-way partitionings, respectively.

4.2 Implementation of Algorithms

All algorithms described were implemented in *C* programming language, compiled and run on a workstation equipped with a 133MHz PowerPC processor with 512-Kbyte external cache and 64-Mbytes of memory.

$\mathcal{H}_C$	K	Number of			Node degree		Net size	
		Nodes	Nets	Pins	avg	max	avg	max
	16	5582.4	16	12002.6	2.15	4.2	750.16	1085.6
CO9	32	6944.6	32	17787.0	2.56	5.6	555.84	846.2
	64	7947.6	64	25527.8	3.21	8.4	398.87	622.0
	16	4858.4	16	10699.2	2.20	4.2	668.70	1054.4
CQ9	32	6069.0	32	16177.4	2.67	5.4	505.54	779.6
	64	7500.8	64	24126.4	3.22	7.8	376.98	641.2
	16	11756.6	16	23909.2	2.03	4.0	1494.33	1901.6
fome12	32	17035.4	32	35561.8	2.09	4.8	1111.31	1406.4
	64	20702.6	64	44642.4	2.16	6.0	697.54	919.6
	16	1559.6	16	3195.0	2.05	5.0	199.69	393.2
fxm4	32	2584.8	32	5321.0	2.06	5.8	166.28	448.0
	64	3693.0	64	7697.0	2.08	6.2	120.27	431.8
	16	1409.0	16	2865.2	2.03	3.8	179.08	302.8
GE	32	1805.4	32	3721.8	2.06	4.2	116.31	238.8
	64	2511.8	64	5224.0	2.08	4.4	81.63	155.8
	16	4386.8	16	9007.4	2.05	3.4	562.96	1162.8
kent	32	9272.4	32	20150.0	2.17	4.2	629.69	1279.6
	64	12533.0	64	32522.0	2.59	5.6	508.16	1022.6
	16	7012.4	16	14630.8	2.09	4.0	914.43	1935.0
mod2	32	9313.8	32	20004.2	2.15	4.4	625.13	1424.4
	64	11842.0	64	26035.8	2.20	4.8	406.81	883.8
	16	2904.8	16	6461.2	2.22	5.0	403.83	627.8
NL	32	3797.8	32	9313.2	2.45	5.6	291.04	438.8
	64	4478.0	64	12143.2	2.71	7.2	189.74	312.4
	16	1821.2	16	3911.4	2.15	3.8	244.46	461.8
pltexpA4_6	32	2130.6	32	4953.2	2.33	4.6	154.79	336.6
	64	2639.4	64	6831.6	2.59	4.8	106.74	325.2
	16	7698.6	16	16011.6	2.08	4.0	1000.73	2022.0
world	32	10372.2	32	22408.8	2.16	4.2	700.28	1596.6
	64	13011.2	64	28760.8	2.21	4.8	449.39	946.2

Table 4.2: Average case properties of communication hypergraphs.

Our objective was to minimize the total message count and total communication volume during the parallelization of algorithms including repeated sparse matrix-vector and matrix-transpose-vector product through the usage of personalized communication scheme described in Section 3.3. As stated previously, the mapping of cut nets to parts is not touched, i.e., there is not any algorithm known that adopts the personalized communication scheme. That is, we have not got any known algorithm to evaluate the relative performance of our algorithms. To validate the usage of the proposed model and algorithms, the naive algorithm, sketched in Fig. 4.1 was developed to map cut nets under partition of a computational hypergraph to parts. Note that for a given cut net the algorithm selects a part holding minimum number of nets among the parts that are connected by this net. Due to the fact that the nets are mapped to the connected parts, this naive algorithm guarantees achieving the lower bound on the communication volume. This algorithm will be referred to as the *Naive* assignment algorithm in the following discussion.

Two algorithms have been designed to solve the communication cost minimization problem. Both algorithms exploit the communication hypergraph model described in Section 3.3.1. The first algorithm described in Section 3.5, using the multilevel hypergraph partitioning tool PaToH [6] as a black box, is named $2Phase$. Recall that the algorithm $2Phase$ solves the communication cost minimization problem in two phases, where the lower bound on the total communication volume is set in the first phase, and total message count is minimized along with the determination of communication loads of the processors in the second phase. The second algorithm that partitions communication hypergraphs to reduce communication cost as described in Section 3.6 assigns messages to the processors during partitioning, hence it solves the communication minimization problem without any auxiliary effort. The first version of this algorithm using priority queues as Sanchis's method will be referred to as $M1$, and the one that does not use priority queues will be referred to as $M2$.

```
Input:  H = (V, N), a computational hypergraph;
        Π, a partitioning on H;
Output: map, a mapping for each n ∈ N_E^Π;
begin
        for all n ∈ N_E^Π do
            p ← min_{p∈Λ_n}{weights[p]};
            weights[p] ← weights[p] + 1;
            map[n] ← p;
        return map;
end
```

Figure 4.1: Naive mapping heuristic

4.3 Experiments

A large number of experiments were conducted with algorithms $Naive$, $2Phase$, $M1$, and $M2$ regarding 16, 32, and 64-way partitionings. Note that the algorithms that are proposed and used are multilevel algorithms —including a random coarsening phase. Experiments are conducted with the algorithms using HCM, SHCM, SHCC metrics. The algorithm that uses HCM during coarsening generated worse results than the other two metrics. There was no clear winner between SHCM and SHCC. For convenience, the results of the algorithms using the SHCM metric during coarsening are presented. This section gives an overview of the experimental studies on those algorithms. Exhaustive presentation of the presented experiments will be presented in Appendix.

In this section, comparative listing of the total message count and the total communication volume are presented (see Table 4.3) for the algorithms listed above. Furthermore, the communication requirements of the global communication scheme will be presented. Throughout the discussion, the label *Global* refers to this scheme. For each partitioning way, two results will be shown. The first one characterizes the total number of messages by giving average message count and maximum message count handled by a processor normalized by the number of processors. The second one characterizes the total communication

volume by giving average communication volume and maximum communication volume handled by a single processor normalized by the number of columns in the associated matrix.

We have conducted experiments to validate performing fold and expand operations by following the personalized communication scheme instead of performing them by following the global scheme. For this purpose, PaToH was run on the computational hypergraphs associated with the matrices listed in Table 4.1 to minimize the number of nets cut by the partition, i.e., with the cost function given in Eq. 2, for communication cost minimization in global communication scheme. Then, the total message count and the total communication volume in the parallelization is computed by assuming an underlying architecture of mesh of processors. For systems with $16, 32$, and 64 processors the meshes are assumed to be $4 \times 4, 4 \times 8$, and 8×8, respectively. Under these settings, the costs are computed as shown in Table 4.3 in the rows *Global*. Note that maximum number of messages is equal to average number of messages because of the global all-to-all broadcast operation. A similar discussion also holds for the communication volume results. In a mesh of processors with r rows and q columns, we have $2((r-1)+(q-1)) \times (r \times q)$ messages to perform the global fold and expand. Furthermore, the total message volume during the fold and expand of *m-words* data segment is $2m(r \times q - 1)$ words. In all of the experiments the global scheme resulted in the maximum communication volume. The total message count due to global communication scheme is smaller than that of the proposed algorithms in 3 instances for 16-way partitioning, and smaller than $2Phase$'s result in only one instance for 32-way instance among 30 experiments. All of the results of the scheme *Global* regarding the total message count for 64-way partitioning are worse than that of the proposed algorithms.

In all of the communication hypergraphs, the algorithm *Naive* approach results in the minimum total communication volume, but the total message count is the poorest one among the algorithms mentioned. The algorithms using the proposed communication hypergraph model increases the total communication volume by an amount up to 53% of the naive algorithm's results ($9\% - 35\%$ on the overall averages). Fortunately, the algorithms reduce the total message

Matrix	Method	16-way				32-way				64-way			
		# of Mssg. per proc.		Comm. Vol. per proc.		# of Mssg. per proc.		Comm. Vol. per proc.		# of Messg. per proc.		Comm. Vol. per proc.	
		avg	max	avg	max	avg	max	avg	max	avg	max	avg	max
CO9	Global	12.0	12.0	0.720	0.720	20.0	20.0	0.897	0.897	28.0	28.0	0.963	0.963
	Naive	28.1	30.0	0.053	0.075	40.5	56.6	0.046	0.070	49.0	90.6	0.037	0.059
	2Phase	13.9	23.0	0.080	0.101	19.9	31.0	0.063	0.084	24.8	53.0	0.049	0.071
	M1	12.2	14.2	0.062	0.081	16.8	20.7	0.052	0.069	20.2	30.5	0.041	0.067
	M2	12.3	13.6	0.068	0.120	17.5	19.2	0.055	0.090	22.0	25.4	0.042	0.074
CQ9	Global	12.0	12.0	0.735	0.735	20.0	20.0	0.854	0.854	28.0	28.0	0.980	0.980
	Naive	27.8	30.0	0.052	0.076	44.5	58.0	0.046	0.065	50.0	91.2	0.039	0.062
	2Phase	13.4	19.0	0.079	0.095	21.3	33.0	0.062	0.088	23.3	48.0	0.045	0.060
	M1	11.4	13.8	0.062	0.081	17.2	21.8	0.051	0.066	21.2	29.5	0.041	0.057
	M2	12.1	13.6	0.065	0.101	17.9	19.8	0.054	0.084	21.8	25.0	0.042	0.080
fome12	Global	12.0	12.0	0.486	0.486	20.0	20.0	0.654	0.654	28.0	28.0	0.793	0.793
	Naive	6.0	6.0	0.031	0.040	14.0	14.0	0.024	0.030	28.9	30.0	0.015	0.019
	2Phase	4.9	8.0	0.037	0.047	8.8	13.0	0.033	0.041	17.0	24.0	0.024	0.029
	M1	4.4	5.0	0.032	0.041	8.4	9.5	0.027	0.033	15.1	16.8	0.019	0.026
	M2	3.5	4.0	0.037	0.051	7.6	8.4	0.030	0.061	15.3	16.4	0.020	0.038
fxm4	Global	12.0	12.0	0.046	0.046	20.0	20.0	0.068	0.068	28.0	28.0	0.194	0.194
	Naive	9.2	19.0	0.006	0.014	10.0	27.4	0.005	0.014	9.3	36.0	0.004	0.014
	2Phase	5.3	11.0	0.004	0.007	6.3	13.0	0.005	0.008	5.9	14.0	0.004	0.008
	M1	5.5	9.6	0.006	0.013	6.4	11.2	0.006	0.016	6.9	13.6	0.004	0.011
	M2	5.4	8.2	0.006	0.013	6.5	10.2	0.006	0.016	7.1	11.4	0.004	0.011
GE	Global	12.0	12.0	0.272	0.272	20.0	20.0	0.330	0.330	28.0	28.0	0.425	0.425
	Naive	16.1	24.4	0.016	0.029	21.5	37.0	0.011	0.019	21.4	42.4	0.008	0.015
	2Phase	10.6	16.0	0.026	0.043	12.8	25.0	0.017	0.029	13.2	27.0	0.011	0.019
	M1	9.4	11.6	0.020	0.028	12.0	17.0	0.013	0.021	13.1	21.2	0.009	0.014
	M2	9.6	10.6	0.021	0.029	12.8	13.8	0.014	0.022	13.7	16.0	0.009	0.015
kent	Global	12.0	12.0	0.550	0.550	20.0	20.0	1.011	1.011	28.0	28.0	1.440	1.440
	Naive	5.2	10.6	0.035	0.063	7.4	13.6	0.039	0.077	12.0	24.2	0.039	0.091
	2Phase	3.9	6.0	0.038	0.061	5.3	10.0	0.046	0.082	7.8	13.0	0.043	0.087
	M1	3.8	5.8	0.038	0.074	5.5	8.3	0.041	0.075	8.8	12.6	0.039	0.084
	M2	4.0	6.0	0.039	0.074	5.7	7.2	0.041	0.072	8.4	10.4	0.040	0.082
mod2	Global	12.0	12.0	0.346	0.346	20.0	20.0	0.453	0.453	28.0	28.0	0.597	0.597
	Naive	13.9	25.6	0.029	0.073	18.6	45.6	0.022	0.047	23.0	72.4	0.014	0.028
	2Phase	9.3	20.0	0.038	0.086	13.6	39.0	0.025	0.055	15.0	35.0	0.018	0.034
	M1	10.0	15.6	0.035	0.070	12.6	22.2	0.025	0.045	14.0	33.0	0.016	0.034
	M2	10.4	13.3	0.036	0.069	13.3	17.2	0.027	0.047	15.4	25.0	0.017	0.033
NL	Global	12.0	12.0	0.513	0.513	20.0	20.0	0.651	0.651	28.0	28.0	0.791	0.791
	Naive	27.7	30.0	0.046	0.067	41.4	53.2	0.035	0.058	49.1	70.2	0.025	0.044
	2Phase	14.0	20.0	0.054	0.076	22.3	32.0	0.047	0.059	27.3	52.0	0.033	0.044
	M1	12.5	13.8	0.054	0.072	19.0	22.0	0.040	0.055	22.0	27.5	0.027	0.044
	M2	13.2	14.6	0.055	0.105	19.9	21.2	0.041	0.071	24.9	27.8	0.028	0.046
pltexpA4	Global	12.0	12.0	0.030	0.030	20.0	20.0	0.027	0.027	28.0	28.0	0.074	0.074
	Naive	7.5	18.2	0.003	0.006	11.1	29.6	0.002	0.006	13.6	48.6	0.002	0.005
	2Phase	4.9	7.0	0.003	0.005	8.7	17.0	0.003	0.005	8.4	21.0	0.002	0.005
	M1	5.0	8.4	0.003	0.006	5.9	11.0	0.002	0.005	8.8	19.0	0.002	0.004
	M2	5.3	7.8	0.003	0.006	6.3	9.8	0.002	0.005	9.1	14.8	0.002	0.004
world	Global	12.0	12.0	0.368	0.368	20.0	20.0	0.489	0.489	28.0	28.0	0.622	0.622
	Naive	17.4	27.6	0.032	0.064	22.9	49.0	0.022	0.049	27.3	81.0	0.015	0.034
	2Phase	11.8	17.0	0.043	0.059	15.1	30.0	0.029	0.049	16.1	46.0	0.019	0.039
	M1	11.2	14.8	0.040	0.065	13.9	26.4	0.025	0.050	15.9	33.5	0.017	0.031
	M2	11.0	12.4	0.043	0.067	14.7	18.2	0.027	0.048	17.3	24.4	0.017	0.030

Table 4.3: Communication requirements during fold and expand

Algorithm	16				
	C	I	R	K-M	Total
PaToH	5.4	0.7	2.0	-	8.9
$M1$	3.7	0.0	0.6	-	4.3
$M2$	3.5	0.0	0.3	-	3.8
$2Phase$	4.6	0.03	0.13	0.00	4.8
	32				
PaToH	6.1	0.9	2.6	-	10.4
$M1$	4.1	0.0	1.4	-	5.5
$M2$	4.2	0.0	0.8	-	5.0
$2Phase$	7.8	0.07	0.23	0.01	8.2
	64				
PaToH	6.6	1.2	3.2	-	12.0
$M1$	3.3	0.0	3.9	-	7.2
$M2$	4.1	0.0	2.4	-	6.5
$2Phase$	12.7	0.12	0.38	0.04	13.5

Table 4.4: Total running time of the algorithms in seconds; C,I,and R stand for time elapsed during coarsening, initial partitioning(mapping), and refinement phases; K-M stands for the time elapsed during the Kuhn-Munkres based assignment algorithm.

count by an amount up to 61% of the naive algorithm's results ($37\% - 44\%$ on the overall averages).

Algorithms $M1$ and $M2$ have approximately the same total message count on the average and they beat algorithm $2Phase$ on this metric by an amount of 10% of the result of $2Phase$ on most of the experiments. Algorithm $M1$ produces total communication volume results less than $M2$ by an amount of 6% of the naive algorithm's volume on the average. Algorithm $M2$ results in total communication volume less than $2Phase$'s results by 15% of the naive algorithm's results. The algorithm $M2$ produces more balanced distribution of message counts than the other algorithms. This can be seen from Table 4.3 by comparing the columns *max* and *avg* under the heading *Number*. The smaller the difference between these two data items, the more balanced the distribution of message count. The same argument applies to the columns under *Volumes*. Unfortunately, the performance of the algorithms on the balance of communication volume is not as satisfying as the balance of message count.

Timing results for the proposed algorithms are also obtained (Table 4.4).

Note that after observing the ratio between the partitioning times of computational hypergraphs and communication hypergraphs, the proposed algorithms were run 5 times and experimental results stated in the previous section were selected from these runs. The coarsening phase is the bottleneck phase. With SHCM metric a coarsening iteration runs in time proportional to the sum of the squares of the sizes of the nets. Therefore, it is not surprising that the time requirement for coarsening communication hypergraph is sometimes more than the time required for coarsening computational hypergraph. As expected, the refinement phase in $M2$ takes less time than the refinement in $M1$, due to the implementation without priority queues. The timing results for the proposed algorithms do not include the time required to partition the computational hypergraph. The total time of partitioning sparse rectangular matrices can be obtained by adding the rows that lists the time requirement for PaToH. This scheme is proper because one can use another partitioning tool in the first stage of the overall algorithm to minimize the total message volume while balancing the computational load. Then, applying our algorithms $2Phase$, $M1$, and $M2$ will incur the times listed in Table 4.4.

4.4 Results

In the light of the results presented in the previous section, the global scheme is proven not to be viable in the parallelization. The message count in the global scheme was smaller than that of the algorithms considering personalized communication in only one 16-way partitioning instance. We further claim that, if the number of processors gets bigger the global scheme's performance will become worse (the results for 32 and 64-way partitioning confirm this claim). The global scheme might be used in systems with small number of processors when the number of nets cut by the hypergraph partitioner is reasonable. Therefore, the discussion will proceed with the proposed algorithms for communication cost minimization in personalized communication scheme.

The algorithm *Naive* seemed to beat the rest of the algorithms on total message volume metric, but it resulted in the largest total message count. That is, there is not a clear winner yet. In this section, the actual communication costs

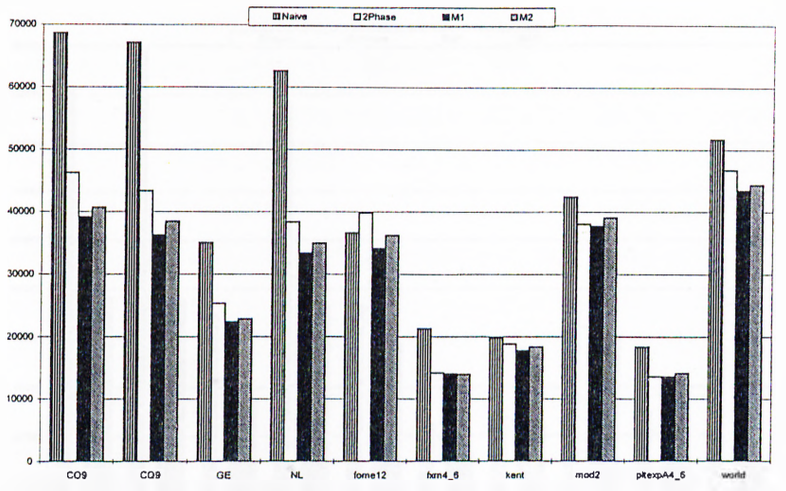

Figure 4.2: 16-way normalized partitioning results of communication hypergraphs

generated by the algorithms will be presented in order to verify the necessity of minimizing total message count as well as minimizing total message volume.

In order to evaluate the performance and quality of the proposed algorithms, the total message counts and the total communication volumes are combined to determine the communication cost that will be realized. Total cost of the communication after partitioning communication hypergraphs is set as follows:

$$cost_{comm} = t_\alpha * M + V \tag{1}$$

where t_α is set to 500/4 as stated by Dongarra and Dunigan [10], 500 is the ratio of latency over bandwidth in a multiprocessor system with *Ethernet* connection which is divided into 4 to convert the unit to a *word*; M is the total message count and V is the total communication volume. Figures Fig. 4.2,Fig. 4.3, and Fig. 4.4 display these normalized results for 16, 32, and 64-way partitionings, respectively for algorithms using personalized communication scheme. The performance of the global communication scheme is not shown, because total communication volume results are not adequate to display.

In figures Fig. 4.2, 4.3, and 4.4, we see that the naive approach does not work. $2Phase$, $M1$, and $M2$ generate promising results. They are better

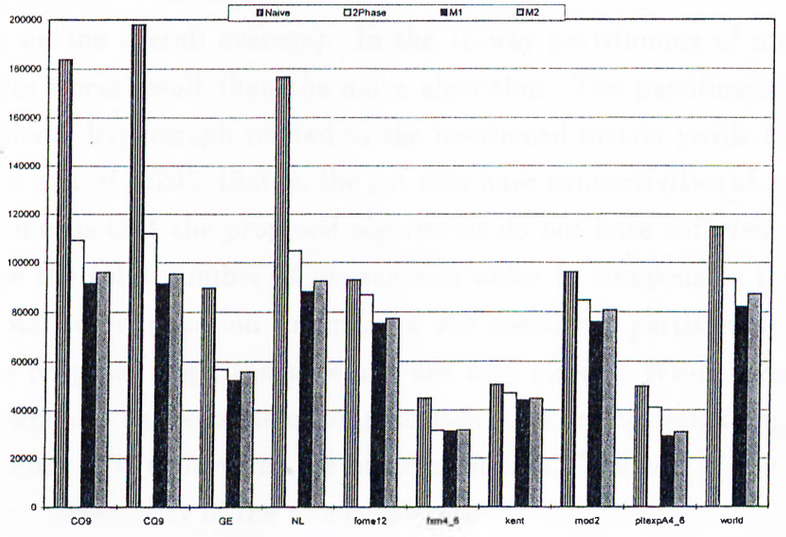

Figure 4.3: 32-way normalized partitioning results of communication hypergraphs

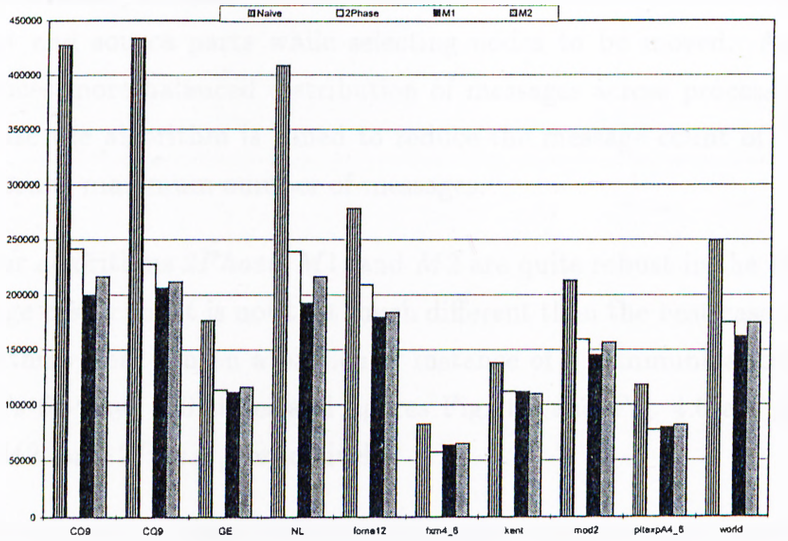

Figure 4.4: 64-way normalized partitioning results of communication hypergraphs

than algorithm *Naive* on the overall cost of the communication. They generate results up to 46% of the naive algorithm's result (55% – 72% on the overall averages). $2Phase$ produces results up to 61% of the naive one's results (81% on the overall average). In the 16-way partitioning of matrix *fome12*, it gives worse result than the naive algorithm. The partitioning of the computational hypergraph related to the mentioned matrix yields 11835 cut nets with a cost of 12295, that is, the cut nets have connectivities of approximately 2. It means that the proposed algorithms do not have sufficient flexibility to reduce the total number of messages in order to compensate the increase in the total communication volume. For the mentioned partitioning instance, all of the proposed algorithms' results are very close to *Naive*'s result. Hence, the difference between the communication costs are not satisfying, and in case of $2Phase$ it is in favor of the naive algorithm. The rest of the experiments confirm the validity of the proposed model and algorithms.

$M1$ and $M2$ are superior to $2Phase$. In addition, $M1$ gives better results than $M2$ on the total message count, second one gives better results on the message imbalance among the processors. This is due to the natures of the algorithms. The first one determines the best move (the move with maximum gain) among the possible ones that are reducing the number of messages handled by a determined processor, whereas second one is biased towards determining the target and source parts while selecting nodes to be moved. Algorithm $M2$ produces more balanced distribution of messages across processors then $M1$, because the algorithm is tuned to reduce the message count of the processor that sends maximum number of messages.

Our algorithms $2Phase$, $M1$, and $M2$ are quite robust in the sense that the average case's result is not that much different than the best case's result. The algorithms were run on a particular instance of a communication hypergraph (CQ9's 64-way) 100 times and figures Fig. 4.5 and Fig. 4.6 are generated for $M1$, $M2$, and $2Phase$, respectively.

In Fig. 4.5, $M1$'s and $M2$'s best and worst results deviate from the average result by 3% of the average total message count. $2Phase$'s best and worst results deviate from the average result by 7% and 25% of the average total message count, respectively.

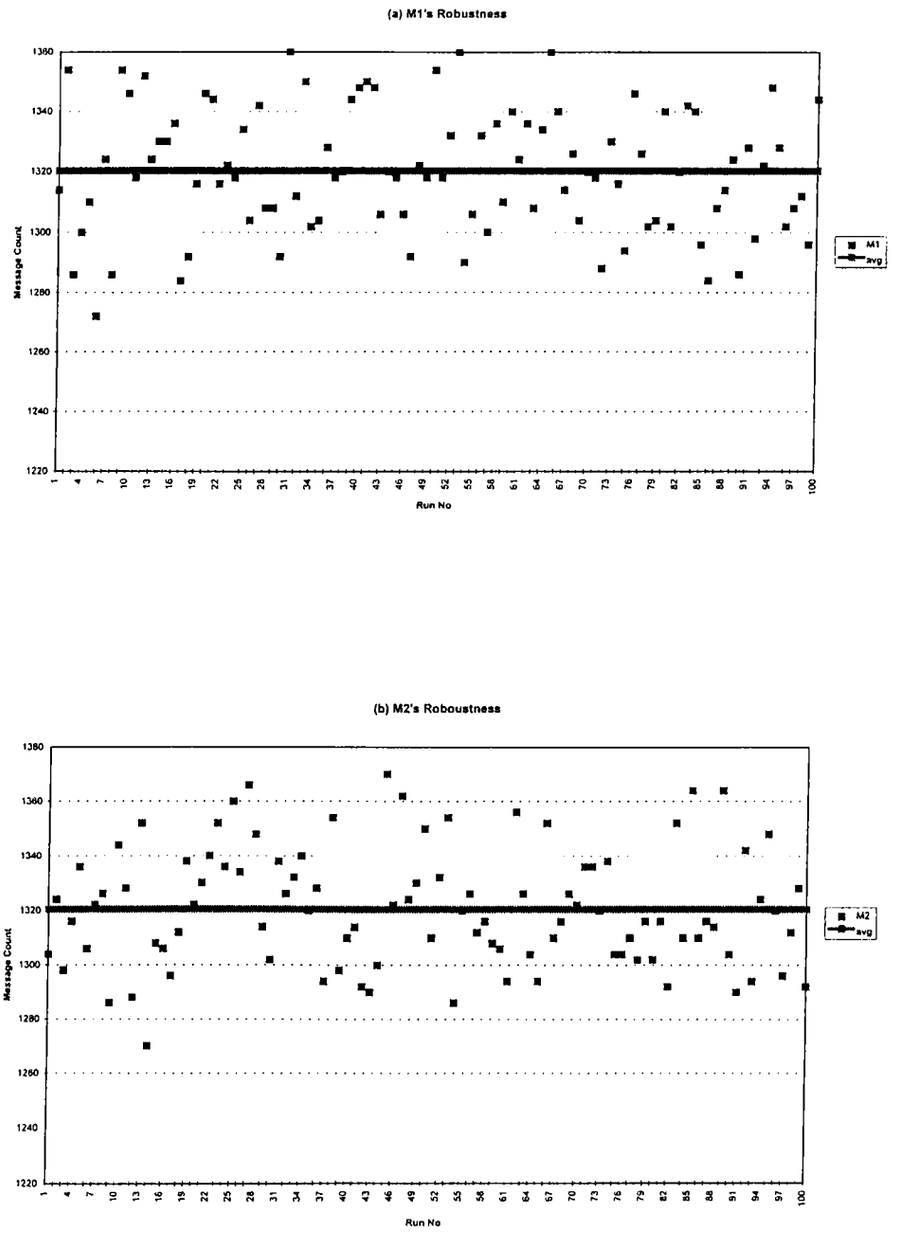

Figure 4.5: 64-way partitioning of CQ9 (a) message counts for $M1$; (b) message counts for $M2$

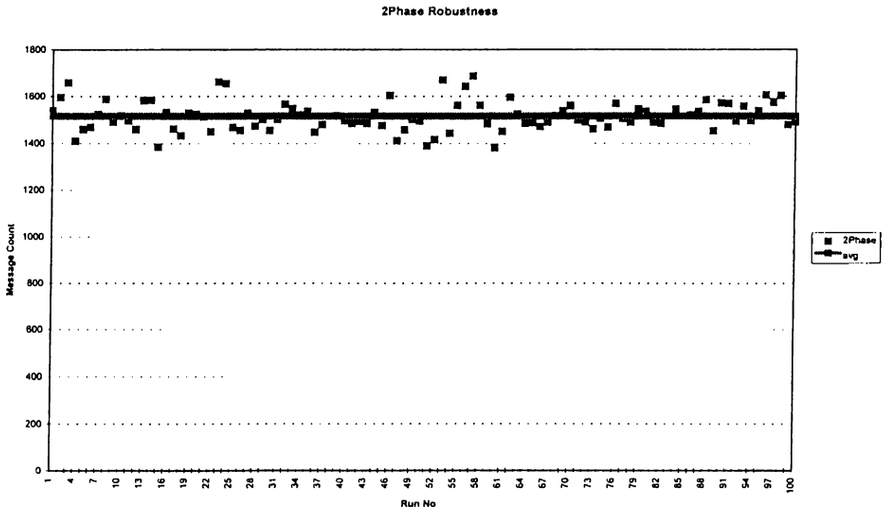

Figure 4.6: 64-way partitioning of CQ9 using $2Phase$: message counts

The coarsening scheme takes an important part on the performance of the proposed algorithms. First of all, note that the net sizes in the communication hypergraph are not low. This implies that the FM-like algorithms will have troubles during the refinement of finer hypergraphs because of high net sizes (see Section 2.4.1). That is, the partitioning realized near the finest hypergraph will not be sufficiently refined. Hence, the constituent nodes in a super-node will be in the same part with high probability. As stated previously, one would like to minimize number of pins running out of a part and running out of its associated net to other nets and parts, respectively (Objective 2 in Section 3.3.1) during the partitioning of communication hypergraph. Anticipating the deficiency of FM on finer hypergraphs, one would like to coalesce two nodes that have almost the same net lists, in order to realize the first goal in the mentioned objective. To realize the second goal, it is preferable to coalesce two nodes that are connected by a common net. The coarsening metrics HCM, SHCM, and SHCC satisfy these requirements. The latter two metrics are biased towards creating nets with smaller sizes than the first one. Therefore, it is not surprising that the latter two generated better communication cost results than the first one.

Another observation related to $2Phase$ is the following. The *connectivity*

metric that is minimized corresponds to minimizing the number of pins in the communication hypergraph. This minimization directly affects the running time of partitioning communication hypergraph. It also helps the minimization of total message volume. As stated previously, the number of nets that are cut by the computational hypergraph partition comprises the nodes in the communication hypergraph. We have conducted experiments where total number of nodes in the communication hypergraphs is minimized using the *cutnet* metric of PaToH on computational hypergraphs. In doing so, the hope was to get an improvement on the communication costs. Unfortunately, this approach resulted in worse total message count in all of the proposed algorithms. The reason is having cut nets with large connectivities after partitioning the computational hypergraph, i.e., the parts in the communication hypergraph have large number of pins running out of the part.

Chapter 5

Conclusion

As an aid to the reader, this final chapter of the thesis restates the research problem and reviews the solutions. Also, directions for future work will be presented.

5.1 Conclusions

In the parallelization of many scientific applications involving the repeated sparse matrix-vector and matrix-transpose-vector product operations we require the distribution of the nonzero elements of the matrix in such a way that the computational loads of the processors are nearly equal and communication cost is low. To achieve these goals in the parallelization of various applications, two models are often used. One of them is the graph model and other is the recently proposed hypergraph model. The problem is then stated in terms of graph partitioning and hypergraph partitioning for the two models, respectively. The algorithms using the above models, accurately balances the computational loads of the processors by partitioning the underlying model's instance. The communication cost that is minimized in these algorithms is the total communication volume, where the algorithms using the first model minimizes just an approximate cost function. The algorithms using the latter model provide efficient methods to minimize the total communication volume

accurately. Unfortunately, minimizing the total communication volume is not sufficient enough to minimize the overall communication cost. The total message count is another component in the communication cost which has a great impact on the performance of parallel applications.

In this work, the problem of minimizing total communication volume and total message count in interior point methods is attacked. While trying to achieve minimum total message count and minimum total communication volume, balance on the communication loads of the processors is also tried to be achieved. A communication hypergraph model was proposed to model the problem and two associated solution methods were also proposed. The validity of the model and solution methods are established by various experiments. The experiments proved that the proposed solutions are necessary by displaying the quality of the solutions obtained by the proposed methods being better than those of the methods like global communication scheme and naive approaches.

Although the proposed algorithms are applied to the interior point methods the application area is not limited by this problem domain. The applications that can use the results and solutions of this thesis is those that have repeated sparse matrix-vector and matrix-transpose vector products. The proposed solutions can be applied to the subclass of applications that follows post-pre communication scheme.

5.2 Future Work

For some reason someone may want to follow the pre-post scheme in his/her application. Therefore, algorithms that minimizes the total message volume and total message count for the pre-post scheme are needed.

Another work to be done is to reduce the total message count in applications involving either matrix-vector product or matrix-transpose-vector products, but not both.

Yet another work can be carried out to minimize the total communication cost while balancing the total communication loads of the processors in the

presence of machine dependent variables start-up cost and per-byte transmission cost.

Bibliography

[1] G. Karypis V. Kumar R. Aggarwal and S. Shekhar. hmetis a hypergraph partitioning package version 1.0.1. Technical report, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

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

[3] C. J. Alpert and A. B. Kahng. Spectral partitioning: the more eigenvectors, the better. In *Proc. ACM/IEEE Design Automation Conference*, 1995.

[4] T. Bultan and C. Aykanat. Circuit partitioning using parallel mean field annealing algorithms. In *Proceedings of Third IEEE Symposium on Parallel and Distributed Processing*, pages 534–541, December 1991.

[5] T. Bultan and C. Aykanat. A new mapping heuristic based on mean field annealing. *Journal of Parallel and Distributed Computing*, 16:292–305, 1992.

[6] Ü. V. Çatalyürek and C. Aykanat. Hypergraph-partitioning based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Distrib. Sys.*, (to appear).

[7] A. Pınar Ü. V. Çatalyürek C. Aykanat and M. Pınar. Decomposing linear programs for parallel solution. In *Lecture Notes in Computer Science 1041*, pages 473–482. Springer-Verlag, 1996.

[8] G. Chartrand and O. R. Oellermann. *Applied and algorithmic graph theory.* International series in pure and applied mathematics. McGraw-Hill, 1993.

[9] J. Cong and M'L. Smith. A parallel bottom-up clustering algorithm with applications to circuit partitioning in vlsi design. In *Proc. ACM/IEEE Design Automation Conference*, 1993.

[10] J. J. Dongarra and T. H. Dunigan. Message-passing performance of various computers. *Concurrency-Practice and Experience*, 9(10):915–926, 1997.

[11] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proceedings of 19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W.H. Freeman and Co., New York, 1979.

[13] S. Hauck and G. Boriello. An evaluation of bipartitioning techniques. In *Proc. Chapel Hill Conf. on Adv. Research in VLSI*, 1995.

[14] B. Hendrickson. Graph partitioning and parallel solvers: has the emperor no clothes? In *Lecture Notes in Computer Science, 1457*, pages 218–225. Springer-Verlag, 1998.

[15] B. Hendrickson and T. G. Kolda. Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel processing. *SIAM J. Sci. Comput.*, (to appear).

[16] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. Technical report, Sandia National Laboratories, 1993.

[17] G. Karypis and B. Kumar. Multilevel k-way hypergraph partitioning. Technical report, University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[18] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, to appear.

[19] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.*, 49(2):291–307, 1970.

[20] B. Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Transactions on Computers*, 33(5):436–446, May 1984.

[21] J. H. Huang L. Hagen and A. B. Kahng. On implementation choices for iterative improvement partitioning algorithms. *IEEE Transactions on CAD*, 16(10):1199–1205, 1997.

[22] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Wiley-Teubner, 1990.

[23] C. W. Qu M. Kondurra and S. Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Parallel and Distributed Technology*, pages 63–69, 1995.

[24] O. C. Martin and S. W. Otto. Partitioning of unstructured meshes for load balancing. *Concurrency:Practice and Experience*, 7(4):303–314, 1995.

[25] A. Pınar and C. Aykanat. An effective model to decompose linear programs for parallel solution. In *Lecture Notes in Computer Science 1184*, pages 592–601. Springer-Verlag, 1997.

[26] C. Aykanat A. Pınar and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into singly bordered block-diagonal form for parallel solution of lp problems. *IEEE. Trans. Parallel and Distributed Systems*, Submitted.

[27] C. W. Qu and S. Ranka. Parallel incremental graph partitioning. *IEEE Transactions on Parallel and Distributed Systems*, 8(8):884–896, 1997.

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

[29] L. A. Sanchis. Multiple-way network partitioning with different cost functions. *IEEE Transactions on Computers*, 42(12):1500–1504, December 1993.

[30] D. G. Schweikert and B. W. Kernighan. A proper model for the partitioning of electrical circuits. In *Proceedings of 9th ACM/IEEE Design Automation Conference*, pages 57–62, 1972.

[31] A. Gupta V. Kumar, A. Grama and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.

[32] L. N. Bhuyan V. Lakamsani and D. S. Linthicum. Mapping molecular dynamics computations on to hypercubes. *Parallel Computing*, 21:993–1013, 1995.

[33] B. Hendrickson W. Camp, S. J. Plimpton and R. W. Leland. Massively parallel methods for engineering and science problems. *Communication of ACM*, 37:31–41, April 1994.

[34] W. Wang and D. P. O'Leary. Adaptive use of iterative methods in interior point methods for linear programming. Technical report, Dept. Comp. Sci. Univ. Maryland, College Park, 1995.

Appendix A

Experimental Results in Detail

Note that in any row of Table A.1 total volume of messages (column TV) is equal to two times cost of partition (C), because of the equlity of total volume of messages in fold and expand phases.

Matrix	P	TM	TV	VI	MM	EMM	EMV	FMM	FMV	C	CN	MV	MI
	16	448.8	12564.4	42.22	30	15	498.6	15	692.6	6282.2	5437.6	1118	7.41
CO9	32	1295.6	21919.2	51.77	56.6	28	414.4	30.8	658.6	10959.6	6877.8	1041	40.02
	64	3136.4	35547.2	57.54	90.6	45.2	432	51.4	548.4	17773.6	8081	873.6	84.63
	16	444.8	11520.4	45.78	30	15	452	15	676	5760.2	4832.2	1050.8	7.94
CQ9	32	1422.8	20182.4	42.07	58	29	448.2	31	524.6	10091.2	6126.8	898	30.48
	64	3199.6	33972.8	59.92	91.2	47.6	416.8	51.8	493.2	16986.4	7599.6	849	82.75
	16	96	24590.4	27.36	6	3	824	3	1154.4	12295.2	11835	1958	0.00
fome12	32	448	36886	28.94	14	7	626.8	7	891.4	18443	16913.4	1486.4	0.00
	64	1848.4	47768.8	27.56	30	15	429.2	15	575	23884.4	20787.8	952.2	3.88
	16	147.2	2879.2	132.14	19	8.6	151.4	13.2	274	1439.6	1369.6	416.6	106.56
fxm4_6	32	318.8	5088	167.16	27.4	10.2	141.6	20.6	298	2544	2377.6	423.6	175.22
	64	596	8019.6	235.04	36	12.2	101	26.8	326.4	4009.8	3696	416.8	284.50
	16	257.2	2900	78.43	24.4	12.2	111	13.6	224.4	1450	1410	323	52.42
GE	32	686.4	3800.4	74.28	37	17.4	79.8	21.8	141.4	1900.2	1794.2	206.8	72.52
	64	1372.8	5384.8	99.26	42.4	19.4	55.8	27	126.2	2692.4	2500	167.6	97.28
	16	83.6	9373.2	79.22	10.6	5	412.6	6	683.4	4686.6	4455.2	1050.2	102.98
kent	32	237.6	20788.8	97.05	13.6	7.2	537	8.8	832.6	10394.4	9266.2	1279	81.69
	64	771.2	41208.4	134.70	24.2	12.8	609.4	14	1001.4	20604.2	12639.8	1515.8	100.92
	16	222.8	14576.4	153.68	25.6	12.4	598.4	15	1803	7288.2	6802.2	2323.2	86.00
mod2	32	594.4	21886.4	116.58	45.6	18.8	495	29.4	1094.6	10943.2	9613.8	1479	147.33
	64	1472	29065.6	99.23	72.4	30	338.2	45	703.6	14532.8	12002	903.2	215.18
	16	443.6	7164	45.71	30	15	336.2	15	431.2	3582	2935.8	653.4	8.23
NL	32	1326	10886.4	65.40	53.2	28.8	281.8	29.2	333.2	5443.2	3815	562.6	28.44
	64	3145.6	15454	75.17	70.2	43	223.6	43.4	228	7727	4426.2	423	42.84
	16	120	3356	101.60	18.2	8.2	150.2	10.2	305.2	1678	1485.2	420.4	143.27
pltexpA4_6	32	355.6	5367.6	134.65	29.6	12.2	131	17.8	283.2	2683.8	2068.6	393	165.07
	64	873.2	8572.4	144.49	48.6	18.2	109.8	31.4	250.2	4286.2	2824.8	327.6	254.74
	16	278.8	16855.6	100.20	27.6	13.2	685.6	15	1497.4	8427.8	7778.8	2105	60.46
world	32	732.4	23338.8	120.70	49	23	521.8	30	1224.6	11669.4	10082.8	1599.2	117.94
	64	1745.6	31731.2	124.27	81	34	350.6	51.6	872.4	15865.6	13267.6	1111.2	198.96

Table A.1: Communication results of naive algorithm on the computational hypergraphs using SHCM metric. P: number of parts; TM: total number of messages; TV: total volume of messages; VI: volume imbalance among parts; MM: maximum number of messages handled by a processor; EMM: MM during expand phase; EMV: maximum volume handled by a processor during expand phase; FMM: MM during fold phase; FMV: maximum volume during fold phase; C: cost of partition; CN: number of cut nets; MV: maximum cumulative volume handled by a processor; MI: number of messages imbalance among processors.

Matrix	P	TM	TV	VI	MM	EMM	EMV	FMM	FMV	C	CN	MV	MI
	16	241.6	18424.8	31.8	21.6	13.2	657.8	11.4	915.4	6282.2	5437.6	1514.8	43.2
CO9	32	784.8	29642.8	35.0	41.4	23.2	539.4	23.4	787.4	10959.6	6877.8	1250.8	68.8
	64	1768.4	43487.2	38.8	55.6	33	397.8	28.4	610.4	17773.6	8081	942.2	101.3
	16	254.8	16548	32.8	21.8	14.8	603	12.6	846.2	5760.2	4832.2	1374.6	37.0
CQ9	32	770.4	26768.4	27.5	40	23.4	488	21.8	637	10091.2	6126.8	1067.6	66.3
	64	1801.2	41230.8	32.9	54.4	30.2	398.6	29.8	539.8	16986.4	7599.6	857.2	93.4
	16	83.2	30015.6	30.4	7.6	4.2	1100.4	4.6	1368.2	12295.2	11835	2445.2	46.1
fome12	32	296.4	51577.6	28.7	13	7.6	1011.8	6.8	1179	18443	16913.4	2074.2	40.3
	64	1156.8	73186.8	22.7	25.2	15	648.8	13.4	795.4	23884.4	20787.8	1402.8	39.5
	16	93.6	3563.2	87.5	9.8	7	167	7.8	327	1439.6	1369.6	415.8	68.8
fxm4_6	32	208	6403.2	120.4	15	9	155	11.2	355.2	2544	2377.6	442.6	129.4
	64	420.4	9986.8	179.0	20.2	9.4	126.8	17	362	4009.8	3696	432.2	205.4
	16	166	4091.6	58.2	15.6	10.4	157.6	9.4	269	1450	1410	403.6	50.7
GE	32	459.6	5654.4	52.1	23.6	13.6	109.8	15.2	177.8	1900.2	1794.2	268.4	64.0
	64	922	7802	64.3	25.8	13.8	77.6	16.8	142.4	2692.4	2500	200.4	79.0
	16	66.8	11017.6	59.1	6.2	3.8	541.2	4.4	768.6	4686.6	4455.2	1097.2	48.2
kent	32	186.4	25726.8	66.9	10	6.2	616.6	6.2	972.2	10394.4	9266.2	1341.2	72.4
	64	546.4	49288.8	89.7	15.4	8.8	542.4	9.8	1079.4	20604.2	12639.8	1464.2	80.5
	16	160	19449.6	122.9	20.2	8.8	771.6	14.2	2060.8	7288.2	6802.2	2697.4	106.3
mod2	32	459.6	30302.4	93.8	32.4	15.4	613.2	23.2	1311.6	10943.2	9613.8	1830.8	128.4
	64	1196	41330.4	73.9	46.2	21.2	410.2	31	775.8	14532.8	12002	1122.2	148.4
	16	259.2	10328.8	30.1	23	14.4	379	12.2	517.6	3582	2935.8	840.6	42.2
NL	32	828.8	15645.2	33.5	37.6	21	271.8	19.4	411	5443.2	3815	652.4	45.2
	64	2080.4	20812	35.0	51.2	30.6	188.2	28	278	7727	4426.2	439	57.7
	16	81.2	3774	91.5	10	5.6	170.6	6.8	326.8	1678	1485.2	450	97.3
pltexpA4_6	32	225.2	6242	114.4	15	9	139.2	10	318.8	2683.8	2068.6	417.6	112.1
	64	583.2	10029.6	135.7	25	11.2	113.2	17.6	282.4	4286.2	2824.8	371	170.6
	16	184.8	23152	79.3	21.6	10.6	936.4	13.4	1807.2	8427.8	7778.8	2590	88.0
world	32	545.2	33020.4	90.9	36	18.2	640.8	25.8	1413.2	11669.4	10082.8	1954.4	115.1
	64	1432	46514.8	97.4	56.6	25.2	455.8	42.6	1017.8	15865.6	13267.6	1432.6	155.5

Table A.2: Communication results of *2Phase* on the communication hypergraphs. P: number of parts; TM: total number of messages; TV: total volume of messages; VI: volume imbalance among parts; MM: maximum number of messages handled by a processor; EMM: MM during expand phase; EMV: maximum volume handled by a processor during expand phase; FMM: MM during fold phase; FMV: maximum volume during fold phase; C: cost of partition; CN: number of cut nets; MV: maximum cumulative volume handled by a processor; MI: number of messages imbalance among processors.

Matrix	P	TM	TV	VI	MM	MI	EMM	EMV	FMM	FMV	C	MV
	16	195.2	14706	30.46	14.2	16.49	10.4	858.8	10	893.2	97.6	1199.4
CO9	32	536.00	24497.30	33.74	20.67	23.60	15.00	607.67	14.00	661.33	268.00	1023.67
	64	1290.00	38493.00	65.40	30.50	51.34	21.50	630.00	24.00	750.00	645.00	995.00
	16	181.60	13588.40	31.38	13.80	21.52	10.60	816.40	10.00	881.00	90.80	1117.60
CQ9	32	551.00	22515.00	28.91	21.75	26.42	14.75	520.25	14.50	603.25	275.50	907.25
	64	1359.00	36280.00	38.88	29.50	38.90	18.50	427.75	20.25	510.50	679.50	787.25
	16	150.40	3570.00	38.58	11.60	23.44	8.60	206.60	7.80	244.60	75.20	308.80
GE	32	382.80	4518.80	65.62	17.00	41.93	11.20	111.80	13.40	199.80	191.40	233.20
	64	840.40	6265.60	62.25	21.20	61.98	12.80	79.00	16.80	133.80	420.20	158.80
	16	200.00	8347.60	33.49	13.80	10.40	12.00	567.80	11.00	524.80	100.00	697.40
NL	32	607.20	12303.20	37.88	22.00	15.93	16.20	332.20	16.00	381.40	303.60	530.40
	64	1405.00	16900.00	60.58	27.50	25.37	22.50	257.50	19.50	236.50	702.50	424.00
	16	70.80	25283.60	25.90	5.00	13.02	3.00	1274.60	3.00	1437.80	35.40	1989.80
fome12	32	267.50	42000.50	23.43	9.50	13.63	7.00	1164.00	6.75	1164.75	133.75	1620.50
	64	966.40	58966.40	37.37	16.80	11.23	13.80	839.80	12.20	745.20	483.20	1266.20
	16	87.60	3054.40	107.68	9.60	75.51	7.00	149.20	7.20	320.80	43.80	393.00
fxm4	32	204.40	5709.20	177.34	11.20	74.97	9.60	179.40	9.60	443.80	102.20	494.20
	64	444.80	8148.40	178.15	13.60	95.57	8.40	124.40	10.80	287.80	222.40	352.00
	16	61.20	10080.00	95.89	5.80	51.21	3.60	515.80	4.20	979.20	30.60	1231.00
kent	32	177.33	21952.70	81.36	8.33	50.52	6.33	589.33	6.00	889.67	88.67	1240.33
	64	561.20	41252.80	115.96	12.60	43.62	10.40	608.00	9.00	993.80	280.60	1391.00
	16	160.00	17735.20	100.44	15.60	57.44	8.60	892.20	12.80	1928.40	80.00	2211.80
mod2	32	403.20	25540.80	78.75	22.20	76.62	14.00	715.00	18.60	1245.00	201.60	1423.40
	64	897.20	32719.20	111.82	33.00	135.03	16.20	409.40	28.20	950.40	448.60	1084.60
	16	80.40	3560.40	79.68	8.40	67.33	5.00	184.40	6.60	350.60	40.20	399.60
pltexpA4	32	189.60	5498.40	101.01	11.00	86.88	6.20	125.80	9.00	295.80	94.80	342.00
	64	565.20	8949.20	114.78	19.00	114.46	10.60	105.60	14.20	247.40	282.60	299.80
	16	178.80	21100.80	62.20	14.80	32.20	10.60	1090.60	11.60	1755.80	89.40	2129.40
world	32	444.40	26609.60	97.55	26.40	90.77	14.60	641.60	21.80	1399.00	222.20	1644.20
	64	1020.50	34862.50	84.41	33.50	109.87	17.25	506.75	27.75	845.00	510.25	1005.25

Table A.3: Communication results of $M1$ algorithm on the communication hypergraphs using SHCM metric. P: number of parts; TM: total number of messages; TV: total volume of messages; VI: volume imbalance among parts; MM: maximum number of messages handled by a processor; EMM: MM during expand phase; EMV: maximum volume handled by a processor during expand phase; FMM: MM during fold phase; FMV: maximum volume during fold phase; C: cost of partition; MV: maximum cumulative volume handled by a processor; MI: number of messages imbalance among processors.

Matrix	P	TM	TV	VI	MM	MI	EMM	EMV	FMM	FMV	C	MV
	16	195.6	16344.4	72.55	13.2	8.01	10.8	1548.6	8.8	1026	97.8	1769.4
CQ9	32	542	26575.2	67.51	18.4	8.70	15.2	1303.8	15.4	782.4	271	1391.8
	64	1386.4	41191.2	90.13	24.2	11.75	20.8	1058.4	19.8	662	693.2	1224.6
	16	186.8	14510.4	64.69	12.8	9.68	9.2	1275.8	9	928	93.4	1496.4
CQ9	32	566.4	24484	74.04	19	7.30	15.8	1154.4	14.4	699.2	283.2	1332.4
	64	1350.4	38126.8	91.79	24	13.70	20.2	989.2	18.4	554.4	675.2	1142.8
	16	153.6	3860.4	48.71	10.6	10.52	9	318.8	8.4	265.6	76.8	357.6
GE	32	400	4896.4	53.39	13.8	10.45	12.6	191.6	12	213.8	200	234.2
	64	870	6848	65.33	15.8	16.26	14.4	165.2	14	147.4	435	177
	16	203.6	8911.2	69.14	14.4	13.14	11.2	838.6	10.2	540.2	101.8	948.2
NL	32	616.4	13432.8	58.14	20.4	5.93	16.6	593	16	425.8	308.2	663.6
	64	1547.2	17984	59.65	27	11.70	21.6	350.2	18.6	251.6	773.6	449
	16	54.8	29418	43.80	4	16.83	3	2575.6	3	1617.4	27.4	2646
fome12	32	236	47462	107.54	8	8.51	7	2934	6.4	1271.2	118	3080.2
	64	971.2	63331.2	123.24	16	5.45	13.4	1991.8	12.2	780.8	485.6	2209.4
	16	87.6	3132.8	102.89	8	47.30	6.8	173.4	6.6	327.4	43.8	393
fxm4	32	209.2	5639.6	181.45	10.6	62.84	7.8	167.4	9.6	442.8	104.6	494
	64	450.8	8284	169.55	11.8	67.22	8.4	136.8	10.6	311.4	225.4	345.4
	16	64.4	10551.2	86.47	5.6	39.33	4.4	744.4	4.6	988.2	32.2	1231
kent	32	177.6	22240.8	72.21	7.6	36.98	7	848.6	6	930.8	88.8	1195.4
	64	541.6	42361.2	96.58	10.4	22.79	9.2	922	9	1089	270.8	1299.8
	16	163.2	18359.2	101.67	13	27.97	9.4	1446.4	11.6	2147.4	81.6	2300.6
mod2	32	419.2	27171.2	74.92	17	30.44	14.4	1171.2	16	1354.6	209.6	1485
	64	982.8	33769.6	96.22	25.6	66.41	18.4	716.4	24.6	1008.4	491.4	1035.6
	16	87.2	3588.4	80.44	8	46.72	5.6	197.6	6.4	355.6	43.6	405.4
pltexpA4	32	202	5533.2	96.53	9.8	56.14	6.4	184.4	8.8	315.6	101	336
	64	570.8	9155.6	98.66	13.8	55.13	12	188.6	12.2	268.2	285.4	283.4
	16	174.4	23033.6	63.49	12.8	17.62	11.2	2219.6	10.4	1909.6	87.2	2361.4
world	32	482.8	28612.8	82.25	19.2	27.07	15	1171.8	18	1512.6	241.4	1620
	64	1104.8	36391.2	76.83	25	45.03	21.6	841.4	23.8	949.6	552.4	1004.6

Table A.4: Communication results of $M2$ on the communication hypergraphs using SHCM metric. P: number of parts; TM: total number of messages; TV: total volume of messages; VI: volume imbalance among parts; MM: maximum number of messages handled by a processor; EMM: MM during expand phase; EMV: maximum volume handled by a processor during expand phase; FMM: MM during fold phase; FMV: maximum volume during fold phase; C: cost of partition; MV: maximum cumulative volume handled by a processor; MI: number of messages imbalance among processors.

Matrix	K	Part Weights		
		max	avg	imb
CO9	16	6654	5674.312500	0.067443
	32	3563	2837.156200	0.024949
	64	1759	1418.578125	0.005859
CQ9	16	5771	4976.187500	0.062392
	32	3134	2488.093750	0.025351
	64	1566	1244.046875	0.006318
GE	16	2062	1840.937500	0.046907
	32	1026	920.468750	0.011196
	64	551	460.234375	0.004815
NL	16	2373	2149.312500	0.040654
	32	1226	1074.656250	0.013753
	64	737	537.328125	0.009072
fome12	16	8140	7390.250000	0.039629
	32	4223	3695.125000	0.013951
	64	2225	1847.562500	0.004988
fxm4	16	15715	14161.812500	0.042842
	32	8440	7080.906250	0.018744
	64	4252	3540.453125	0.004907
kent	16	11193	9588.125000	0.065383
	32	5509	4794.062500	0.014563
	64	3004	2397.031250	0.006182
mod2	16	9573	8147.187500	0.068362
	32	4992	4073.593750	0.022017
	64	2535	2036.796875	0.005972
pltexpA4	16	8075	7260.312500	0.043832
	32	4286	3630.156250	0.017643
	64	2333	1815.078125	0.006966
world	16	9740	8122.750000	0.077774
	32	5049	4061.375000	0.023748
	64	2517	2030.687500	0.005847

Table A.5: Computational load distribution