MINIMIZING COMMUNICATION LATENCIES IN CONJUGATE GRADIENT TYPE PARALLEL ITERATIVE SOLVERS

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BİLKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

M. Mustafa Özdal

July, 2001

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. Cevdet Aykanat(Advisor)

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

Assist. Prof. 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.

Assist. Prof. Uğur Güdükbay

Approved for the Institute of Engineering and Science:

Prof. Mehmet Baray
Director of the Institute

ABSTRACT

MINIMIZING COMMUNICATION LATENCIES IN CONJUGATE GRADIENT TYPE PARALLEL ITERATIVE SOLVERS

M. Mustafa Özdal
MS in Computer Engineering
Supervisor: Prof. Cevdet Aykanat
July, 2001

Conjugate Gradient (CG) type iterative solvers are widely used for the solution of large, sparse, linear system of equations on multicomputers. Typically, the basic operations performed at each iteration are sparse matrix vector multiplications (SpMxV), and inner product computations. In the parallel CG algorithm, SpMxV operations require point-to-point type communications, whereas inner product computations require all-to-all broadcast (AABC) type communications. In this thesis, we propose a novel communication scheme in which the point-to-point communications are embedded into the AABC operations. The purpose here is to minimize the number messages sent by each processor, so that the communication latencies of a parallel CG program are minimized. However, such a scheme has the disadvantage that the communication volume requirements might increase. For this reason, a cost model and a methodology to minimize the overhead in communication volume is given. Some experiments have been performed to test the practical validity of the proposed scheme. It is observed that the execution times for communication operations decrease in this scheme, especially for the configurations in which the message start-up costs dominate the total communication times.

Keywords: sparse matrices, parallel iterative solvers, parallel matrix-vector multiplication, communication minimization, hypergraph partitioning.

ÖZET

EŞLENİK GRADYAN TİPİ PARALEL ÖZYİNELİ ÇÖZÜCÜLERDE İLETİŞİM GECİKME SÜRELERİNİN EN AZA İNDİRGENMESİ

M. Mustafa Özdal
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Cevdet Aykanat
Temmuz, 2001

Eşlenik Gradyan (EG) tipi özyineli çözücüler büyük, seyrek, doğrusal denklem sistemlerinin çozümlerinde sıkça kullanılmaktadırlar. Genelde, her özyinelemede uygulanan temel işlemler, seyrek matris vektör çarpımları (SyMxV) ve vektör iç çarpımlarıdır. Paralel EG algoritmalarında, SyMxV işlemleri *noktadan noktaya* tipinde iletişim gerektirirken, iç çarpım işlemleri *herkesten herkese yayım* (HHY) tipinde iletişim gerektirmektedirler. Bu tezde, noktadan noktaya iletişim işlemlerinin HHY işlemlerinin içine gömüldüğü yeni bir iletişim metodu önerilmektedir. Buradaki amaç, her bir işlemci tarafından gönderilen mesaj sayısının en aza indirgenmesi ve böylece paralel EG algoritmasındaki iletişim gecikme sürelerinin azaltılmasıdır. Diğer yandan, böyle bir metodun iletişim hacmini arttırma gibi bir dezavantajı vardır. Bu yüzden, iletişim hacmindeki ek yükleri en aza indirgemek için bir yöntem maliyet modeli ile birlikte sunulmaktadir. Önerilen yöntemlerin pratikteki geçerliliklerini gözlemlemek için deneyler yapılmıştır. Özellikle mesaj başlama maliyetlerinin toplam iletişim zamanlarinda baskın olduğu durumlarda, işlemci zamanlarında bir azalma gözlemlenmiştir.

Anahtar sözcükler: seyrek matrisler, paralel özyineli çözücüler, paralel matris-vektör çarpımı, haberleşme en aza indirme, hiperçizge bölümleme.

Acknowledgement

I would like to express my gratitude to my supervisor Cevdet Aykanat for his guidance, useful suggestions, and invaluable encouragement throughout the development of this thesis. It was really motivating and enjoyable to work with him.

I would like to thank Assist. Prof. Attila Gürsoy and Assist. Prof. Uğur Güdükbay for reading and commenting on this thesis.

I would also like to thank B. Barla Cambazoğlu and Eray Özkural for maintaining the parallel computer system on which the experiments in this thesis have been performed. I am also grateful to Bora Uçar for providing some of the preprocessing software and test matrices.

Finally, I am grateful to my family and all my friends for their infinite moral support and help throughout this thesis.

To my family and friends

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Conjugate Gradient (CG) type iterative solvers are widely used for the solution of large, sparse, linear system of equations on multicomputers. In most cases, the basic operations performed at each iteration are a sparse matrix vector multiplication (SpMxV) of the form $y = Ax$, followed by linear vector operations on dense vectors y and x. In the parallel versions of these solvers, sparse matrix A is partitioned among different processors, and each processor becomes responsible for the computation of the data assigned to it. Here, the processors might need data from other processors to perform their local computations, hence communication between different processors is required. Specifically, each SpMxV operation requires point to point communication of some vector elements between specific processors; whereas some of the linear vector operations (e.g. inner product) require the communication of a single word among all the processors. Note that for the latter type of operations, a common term *reduction*, will be used throughout this thesis.

In SpMxV computations, each nonzero element in a row/column incurs a multiply-add operation. So, to obtain a balanced computational load, it is necessary to distribute the input matrix among processors such that each processor has balanced number of nonzeros. Furthermore the amount of data needed to be communicated between processors depends on the way the matrix is partitioned. Based on these considerations, there exist different models and algorithms for

partitioning the input matrix among processors such that the amount of communication is minimized, and the computational load of each processor is balanced. However, as also noted in [17], the existing partitioning schemes try to minimize the total volume of communication, but not the total number of messages. Such approaches have the disadvantage of ignoring the message start-up costs, which can be an important factor in the performance of parallel programs.

The message passing time between two processors is determined by three principal parameters: *start-up time* (t_s), *per-hop time* (t_h), and *per-word transfer time* (t_w) [28]. The start-up time is defined to be the time required to send a message of zero length (or of a very small length). It is in fact the time required to perform initial operations such as preparing the message, executing the routing algorithm, establishing an interface between the local processor and the router, etc. The per-hop time is defined as the time taken by the header of a message to travel between two directly connected processors. However, the per-hop delay is negligible on most of the current message passing multiprocessor systems due to wormhole routing techniques and the small diameter of the network [13]. The per-word transfer time is the time for a word to traverse the link from the source processor to the destination processor, and it is inversely proportional to the channel bandwidth. If the per-hop time is to be neglected, the communication time for a message of m words can be expressed as: $t_{comm} = t_s + mt_w$. In the work of Dongarra and Dunigan [11], the latency (t_s) and bandwidth (B) values for various architectures are measured. Assuming that a floating point word is 4 bytes, we calculate t_w values as $t_w = 4/B$, and give this data in Table 1.1. Observe in the table that the ratio t_s/t_w can be in the order of a thousand, depending on the architecture. Hence it is appropriate to state that start-up costs can be significant in the total communication time if the number of words to be communicated is not large enough.

In this thesis, a novel scheme for CG type solvers is proposed to avoid the communication start-up costs occuring in SpMxV computations. The approach will be to embed the messages needed to be communicated in the SpMxV operations into the following reduction operations in the CG algorithm. A reduction is in fact an AABC (all-to-all broadcast) type operation, in which each processor

Table 1.1: t_s and t_w values for various parallel architectures

Machine	OS	t_s (μsec)	t_w (μsec)
Convex SPP1000(PVM)	SPP-UX 3.0.4.1	76	0.364
Convex SPP1000(sm 1-n)	SPP-UX 3.0.4.1	2.5	0.049
Convex SPP1000 (sm m-n)	SPP-UX 3.0.4.1	12	0.068
Convex SPP1200 (PVM)	SPP-UX 3.0.4.1	63	0.267
Convex SPP1200 (sm 1-n)	SPP-UX 3.0.4.1	2.2	0.043
Convex SPP1200 (sm m-n)	SPP-UX 3.0.4.1	11	0.056
Cray T3D (sm)	MAX 1.2.0.2	3	0.0313
Cray T3D (PVM)	MAX 1.2.0.2	21	0.148
Intel Paragon	OSF 1.0.4	29	0.026
Intel Paragon	SUNMOS 1.6.2	25	0.023
Intel Delta	NX 3.3.10	77	0.500
Intel iPSC/860	NX 3.3.2	65	1.333
Intel iPSC/2	NX 3.3.2	370	1.429
IBM SP-1	MPL	270	0.571
IBM SP-2	MPI	35	0.114
KSR-1	OSF R1.2.2	73	0.500
Meiko CS2 (sm)	Solaris 2.3	11	0.100
Meiko CS2	Solaris 2.3	83	0.093
nCUBE 2	Vertex 2.0	154	2.353
nCUBE 1	Vertex 2.3	384	10
NEC Cenju-3	Env. Rel 1.5d	40	0.308
NEC Cenju-3 (sm)	Env. Rel 1.5d	34	0.16
SGI	IRIX 6.1	10	0.063
TMC CM-5	CMMD 2.0	95	0.444
Ethernet	TCP/IP	500	4.444
FDDI	TCP/IP	900	0.412
ATM-100	TCP/IP	900	1.143

has its own message, and it is required that each processor receives the messages of all the other processors. Assume that in an SpMxV operation, processor p_k is required to send a vector element to processor p_ℓ. Since it is definite that a message from p_k will reach p_ℓ in the following reduction operation, it is possible for p_k to embed the vector element it needs to send p_ℓ into this message. Such an approach completely avoids the start-up costs of the messages communicated in SpMxV computations. However, it requires some redundant linear vector operations as will be explained in Chapter 3, and it increases the total volume of communication, compared to the original scheme. The reason for the increase in the communication volume is that, in the reduction operations, a message from p_k does not necessarily reach p_ℓ in a single step. Hence, the embedded vector elements needed to be communicated from p_k to p_ℓ might also require multiple steps to reach their destinations. However, these vector elements are communicated in a single step in the original scheme.

In the single port communication model, an efficient AABC algorithm for a *ring* network of n processors can be summarized as follows [28]. In the beginning, each processor sends its own message to one of its neighbors. Then in the subsequent $n-2$ steps, it forwards the data received from one of its neighbors to its other neighbor. Based on this, the AABC for 2D mesh is given with a two-phase approach. In the first phase, each row of the processor mesh performs AABC using the algorithm for the ring network. Then again the AABC for ring algorithm is performed in the second phase, but this time on each column of the mesh. Observe that in these algorithms, multiple steps are required for a message from a processor to reach another processor. It is possible to define a *distance* concept between two processors based on the steps required for the communication of a message between them. In Chapter 3, a hypergraph theoretical cost function to model the new version of the communication volume is proposed using this distance concept.

For the volume of communication incurred in this approach be small, it is also required that the processors communicating during the SpMxV computations be *close* to each other. As will be described in detail in Chapter 4, this is realized by trying to minimize the cost function mentioned above. For this, a two-phase

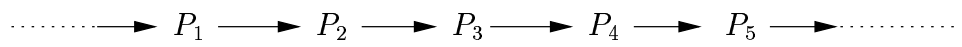

Figure 1.1: An example ring network of processors

approach is used. In the first phase, column-net hypergraph model [8] is used to partition the input matrix such that the total number of words that are required to be communicated is minimized regardless of the distance . For this, the multilevel hypergraph partitioning tool PaToH [8] is used. In the second phase, based on the distances between processors, each part is mapped to a processor such that the incurred volume of communication is minimized. Here, the iterative improvement heuristics given by Kernighan and Lin [27] is used to minimize the cost function defined.

It is expected that, a successful partitioning and mapping of the matrix provides that the processors that have a large number of words to be communicated between them are close to each other, and the ones with small number of words are far away. In accordance with this, consider the example in Figure 1.1, where the AABC operation is defined to be from left to right on the ring of processors. Assume that P_1 is required to send n words to P_2, which has a *distance* of 1 to P_1, and P_2 is to send m words to P_5, which has a distance of 3 to P_2, and assume further that $n \geq 3m$. The straightforward approach for communicating this data is the following: In the first AABC step, P_1 sends n words to P_2 and P_2 sends m words to P_3. In the second step, P_3 forwards these m words to P_4. Finally in the third step P_4 forwards them to P_5. It is important here to remember that regardless of the extra data communicated (i.e the packets with n and m words), each processor sends a message to its right neighbor in each step to perform the AABC operation. The extra communicated data are simply embedded into these messages without incurring any additional start-up costs. Based on this consideration, the extra communication cost becomes $(n + 2m)t_w$, since the communication of m and n words are performed concurrently in the first step. Now assume that the communication pattern for the packet from P_2 to P_5 is unchanged, but the packet from P_1 to P_2 is sent in three steps as subpackets of size $n/3$. In this case the extra communication cost reduces to nt_w due to the concurrent communication of $n/3$ and m words in each of the three steps. This

example shows that it is also important how to embed the data to be communicated into the AABC operation. In Chapter 4, a simple scheduling algorithm is proposed for this purpose.

In this thesis, the models and algorithms are given for rowwise decomposition of a sparse symmetric matrix. The computation model of the sparse matrix vector multiplication for rowwise decomposition on 2D meshes is outlined in Chapter 2 together with the existing partitioning schemes. In Chapter 3, the new communication model for CG type solvers is explained. Chapter 4 gives the details of the algorithms required for such a model. The effect of the new communication model is observed by the help of experiments in Chapter 5. Finally, the concepts given for 2D meshes are generalized for multidimensional meshes in Chapter 6.

Chapter 2

Rowwise Decomposition Models for SpMxV

Matrix vector multiplication that is performed repeatedly on the same sparse square matrix is common among CG type iterative solvers. Furthermore rowwise decomposition scheme is assumed for the matrix, and symmetric partitioning for the input vectors. Specifically, for the sparse matrix vector multiplication (SpMxV) of the form $y = Ax$, the decomposition for K processors will be as follows:

$$A = \begin{bmatrix} A_1 \\ \vdots \\ A_k \\ \vdots \\ A_K \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ \vdots \\ x_K \end{bmatrix}, \qquad \textit{and} \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_k \\ \vdots \\ y_K \end{bmatrix}$$

Here, processor P_k owns row stripe A_k and vector parts x_k and y_k. It will be responsible for computing $y_k = A_k x$ during the SpMxV computations, and the linear vector operations on x_k and y_k. Observe that for the local SpMxV computations, processor P_k requires the vector entries $x[i]$ corresponding to all nonzero columns i of A_k. Since P_k computes only the x_k portion of the x vector, it has to receive all $x[i]$ values for which $x[i] \notin x_k$, from the corresponding processors

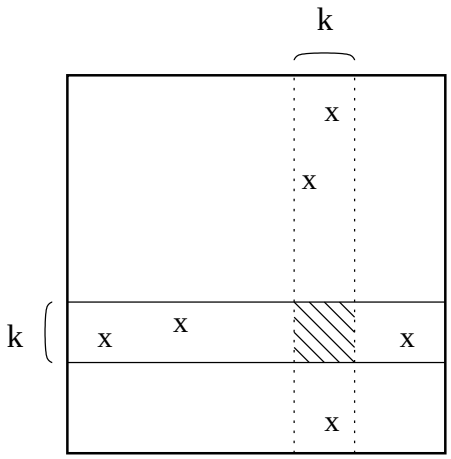

Figure 2.1: Communication requirements for rowwise decomposition

before SpMxV computations. Thus rowwise decomposition can be considered as a *pre* communication scheme.

The communication requirements for rowwise decomposition can be seen more clearly by considering Figure 2.1. Here it is again assumed that the row stripe A_k is assigned to processor P_k, together with the corresponding vector parts x_k and y_k. In the figure, each symbol 'x' marks a nonzero element in the row stripe and column stripe k of the matrix. If a processor P_ℓ has an 'x' marked on its row stripe A_ℓ, it means that P_ℓ needs a vector element that has been computed by P_k. Similarly, if A_k has an 'x' marked on the column stripe ℓ, it means that processor P_k needs a vector element that has been computed by P_ℓ. Hence, the nonzeros in column stripe k and row stripe k indicate *send* and *receive* operations to be performed by P_k respectively. Observe that the nonzero elements in the shaded block of the figure incur no communication, and such blocks are called *diagonal blocks*. The number of words P_k is required to receive is given as the number of nonzero columns in the off-diagonal blocks of A_k (3 in the figure). Hence it is possible to make a generalization that the total communication volume for a rowwise decomposition of a matrix is equal to the sum of the nonzero columns in each off-diagonal block [8, 18].

Note that during the decomposition of the matrix, each row can be assigned to each part, that is there is no requirement for contingency of the rows. However,

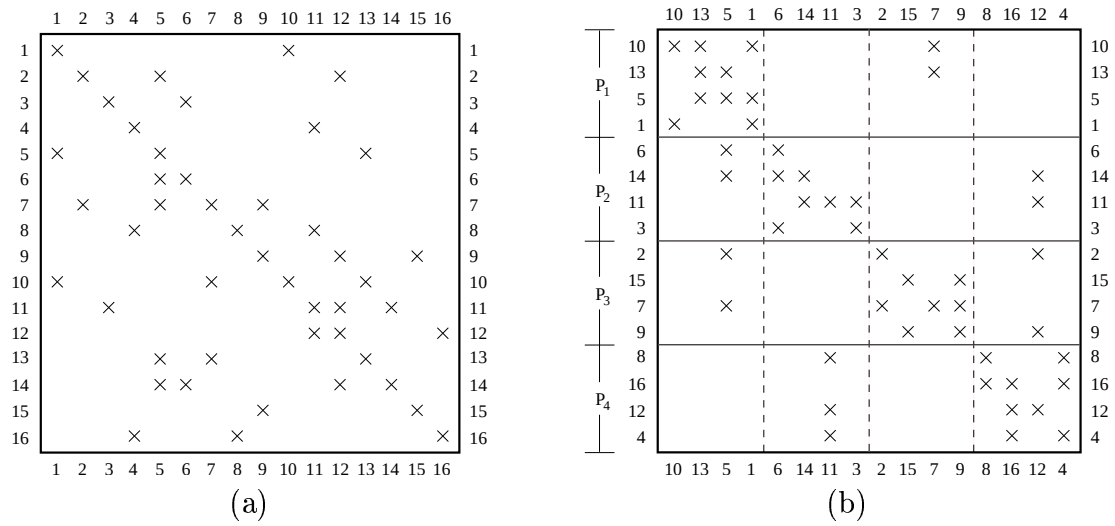

Figure 2.2: Nonzero pattern of a 16×16 matrix (a) before partitioning, and (b) after partitioning

for the above discussions to hold, the rows and columns of the input matrix are repermuted after the matrix is partitioned. Figure 2.2 gives such an example. Observe that after partitioning and repermutation of the matrix, the number of nonzeros in the off-diagonal blocks is small. In fact this is one of the objective metrics of the existing partitioning models, i.e. keeping the number of off-diagonal nonzeros small so that the volume of communication for the parallel SpMxV is small. In the subsequent sections, such existing partitioning schemes based on graph and hypergraph models are summarized.

2.1 Graph Model Based Decomposition

An undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is defined as a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Every edge $e_{ij} \in \mathcal{E}$ connects a pair of distinct vertices v_i and v_j. The degree d_i of a vertex v_i is equal to the number of edges incident to v_i. It is possible to assign a weight w_i for each vertex $v_i \in \mathcal{V}$ and a cost c_{ij} for each edge $e_{ij} \in \mathcal{E}$.

The computational graph model is widely used in the representations of computational structures of various scientific applications, including repeated SpMxV computations, to decompose the computational domains for parallelization

[4, 5, 22, 26, 31, 34]. The problem of 1D sparse matrix decomposition for minimizing the communication volume while maintaining the load balance is formulated in this model as the K-way *graph partitioning by edge separator* (GPES) problem.

2.1.1 Graph Partitioning by Edge Separator (GPES)

An edge subset $\mathcal{E}_S \subseteq \mathcal{E}$ is a K-way *edge separator* if its removal disconnects the graph into at least K connected components. $\Pi_{GPES} = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ is a K-way *partition* of $\mathcal{G}$ by edge separator $\mathcal{E}_S$ if the following conditions hold:

- each part $\mathcal{V}_k$ is a nonempty subset of $\mathcal{V}$, i.e., $\mathcal{V}_k \subseteq \mathcal{V}$ and $\mathcal{V}_k \neq \emptyset$ for $1 \leq k \leq K$,
- parts are pairwise disjoint, i.e., $\mathcal{V}_k \cap \mathcal{V}_l = \emptyset$ for all $1 \leq k < l \leq K$,
- union of K parts is equal to $\mathcal{V}$, i.e., $\bigcup_{k=1}^{K} \mathcal{V}_k = \mathcal{V}$.

A K-way partition is also called a *multiway* partition if $K > 2$ and a *bipartition* if $K = 2$. Here, all edges between the vertices of different parts belong to $\mathcal{E}_S$. Edges in $\mathcal{E}_S$ are called *cut* or *external* edges, and all other edges are called *uncut* or *internal* edges. In a partition Π_{GPES} of $\mathcal{G}$, a vertex is said to be a *boundary* vertex if it is incident to a cut edge.

The weight W_k of a part $\mathcal{V}_k$ is defined as the sum of the weights of the vertices in that part, that is $W_k = \sum_{v_i \in \mathcal{V}_k} w_i$. A partition is said to be balanced if each part $\mathcal{V}_k$ satisfies the balance criterion: $W_k \leq W_{avg}(1+\varepsilon)$. Here, $W_{avg} = (\sum_{v_i \in \mathcal{V}} w_i)/K$ denotes the weight of each part under the perfect load balance condition, and ε represents the predetermined maximum imbalance ratio allowed.

The *cutsize* definition for representing the cost $\chi(\Pi_{GPES})$ of a partition Π_{GPES} is defined as follows:

$$\chi(\Pi_{GPES}) = \sum_{e_{ij} \in \mathcal{E}_S} c_{ij} \tag{2.1}$$

Here, observe that each cut edge e_{ij} contributes its cost c_{ij} to the cutsize. Hence, the K-way GPES problem can be defined as the task of dividing a graph into K parts such that the cutsize is minimized, and each part satisfies the balance criterion. Note that GPES problem is known to be an NP-hard problem [14]. However, multilevel graph partitioning heuristics [3, 19, 26] have been proposed leading to fast graph partitioning tools such as Chaco [20], MeTiS [25], and WGPP [15].

2.1.2 Standard Graph Model for Structurally Symmetric Matrices

A structurally symmetric sparse matrix A can be represented as an undirected graph $\mathcal{G}_A = (\mathcal{V}, \mathcal{E})$, where the sparsity pattern of A corresponds to the adjacency matrix representation of graph $\mathcal{G}_A$. In the rowwise decomposition of the matrix, each vertex $v_i \in \mathcal{V}$ corresponds to atomic task i of computing the inner product of row i with column vector x. Since each nonzero entry in a row of A incurs a multiply-and-add operation during the local SpMxV computations, the computational load for the atomic task i can be defined to be the number of nonzero entries in row i. Assume that w_i and d_i are the weight and degree of vertex v_i respectively. Then, $w_i = d_i$ if the diagonal entries of A are zero, and $w_i = d_i + 1$ otherwise.

Observe that this model displays a bidirectional computational interdependency view for SpMxV. Since matrix A is assumed to be structurally symmetric, each edge $e_{ij} \in \mathcal{E}$ can be considered as incurring the computations $y_i = y_i + a_{ij} \times x_j$ and $y_j = y_j + a_{ji} \times x_i$. If rows i and j are assigned to the same processor, then edge e_{ij} does not incur any communication, because the vector elements x_i and x_j are also assigned to this processor. However, if rows i and j are assigned to different processors, then cut edge e_{ij} necessitates the communication of two floating point words. This is because, atomic task i needs the vector element x_j, and atomic task j needs the vector element x_i for the SpMxV computations. Depending on these considerations, it is possible to reduce the rowwise decomposition of matrix

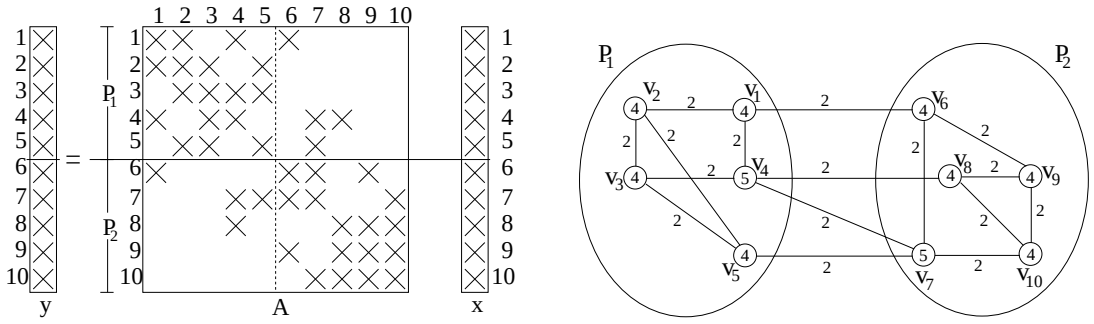

Figure 2.3: Two-way rowwise decomposition of a matrix A, and the corresponding bipartitioning of its associated graph $\mathcal{G}_A$

A to the K-way partitioning of its associated graph $\mathcal{G}_A$. For this purpose, the cost c_{ij} for each edge $e_{ij} \in \mathcal{E}$ is set to 2, and the cutsize metric described in Section 2.1.1 is used together with the balance condition. Note that minimizing the cutsize of the graph is an effort towards minimizing the total volume of interprocessor communication, and maintaining the balance condition corresponds to maintaining the computational load balance during local SpMxV computations.

An example 10×10 symmetric sparse matrix A and its associated graph $\mathcal{G}_A$ are illustrated in Figure 2.3. Here, a two-way rowwise decomposition is given for a two-processor system. The numbers inside the circles indicate the computational weights of the respective vertices of $\mathcal{G}_A$. Here, a perfect load balance is achieved by assigning 21 nonzero entries to each row stripe. Observe also that the cutsize of the given graph bipartitioning is 8, which is also equal to the total number of nonzero entries in the off-diagonal blocks of A.

2.1.3 Alternative Models and Metrics for Graph Partitioning

The standard graph model is not suitable for the partitioning of nonsymmetric matrices. Catalyurek [7] generalizes the standard graph model to nonsymmetric square matrices by assigning an appropriate cost value for the edges. Hendrickson and Kolda proposes a *bipartite graph model* [18, 16] that enables the partitioning of rectangular matrices. It is sometimes the case that balancing a single quantity

such as the number of nonzeros is not enough. Karypis and Kumar proposes the *multi-constraint* model [23], where each vertex is assigned a vector of k weights, and the objective of the graph partitioning is to minimize the communication costs while keeping each of k weights balanced. Another approach is the *skewed partitioning* proposed by Pellegrini [33] and Hendrickson [21]. Here, each vertex has a set of k preference values for each part k, and these preference values are considered together with other partitioning objectives. For further details on these subjects, the reader may refer to the survey paper by Hendrickson and Kolda [17].

2.1.4 Flaws of the Graph Model

Consider the example given in Figure 2.3, and assume that parts $\mathcal{P}_1$ and $\mathcal{P}_2$ are mapped to processors P_1 and P_2 respectively. Here, the cut size for this partitioning is equal to 8, thus the communication volume is estimated to be 8 words. However, observe that the off-block-diagonal nonzeros $a_{4,7}$ and $a_{5,7}$ in processor P_1 incur the need for the same nonlocal x vector component x_7. So, P_2 will send x_7 only once to P_1, instead of twice as depicted in the graph model. Similarly, P_1 will send x_4 only once to P_2 because of the nonzeros $a_{7,4}$ and $a_{8,4}$. Hence, the actual communication volume is 6, not 8 as estimated by the graph model. In fact, the edge cut metric of the graph model can not recognize that two or more edges may represent the same information flow, so it overcounts the true volume of communication [17]. In matrix theoretical view, the nonzero entries in the same column of an off-diagonal block incur the communication of a single x value. However, the graph models try to minimize the total number of off-block-diagonal nonzeros without considering their relative spatial locations [8].

There are also some other shortcomings of the graph model [17]. First of all, it does not consider the number of messages to be sent, hence it ignores the communication start-up costs. However, depending on the machine architecture and the problem size, number of messages can be as important as the message volume, as also mentioned in Chapter 1. Another shortcoming of the graph model

is that it does not try to balance the volume of communications performed by each processor. It is possible that a processor is assigned most of the communication load, although the total volume of communication is minimized. In that case, the parallel performance shall degrade, because the processor with the most communication load will be the bottleneck.

The hypergraph model proposed by Catalyurek and Aykanat [8] addresses the principle problem of the edge cut metric of the graph model. In this work, they model the actual total communication volume exactly using hypergraph partitioning models. The next section outlines these concepts.

2.2 Hypergraph Model Based Decomposition

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$ is defined as a set of vertices $\mathcal{V}$ and a set of nets (hyperedges) $\mathcal{N}$ among those vertices [2]. Every net $n_j \in \mathcal{N}$ is a subset of vertices. The vertices in a net n_j are called its pins and denoted as $pins[n_j]$. The size s_j of a net n_j is equal to the number of its pins, i.e., $s_j = |pins[n_j]|$. The set of nets connected to a vertex v_i is denoted as $nets[v_i]$. The degree of a vertex is equal to the number of nets it is connected to, i.e., $d_i = |nets[v_i]|$. Graph is a special instance of hypergraph such that each net has exactly two pins. Similar to graphs, w_i and c_j denote the weight of vertex $v_i \in \mathcal{V}$ and the cost of net $n_j \in \mathcal{N}$, respectively.

2.2.1 Hypergraph Partitioning

Definition of K-way partitioning of hypergraphs is identical to that of GPES. In a partition Π of $\mathcal{H}$, a net that has at least one pin in a part is said to *connect* that part. *Connectivity* set Λ_j of a net n_j is defined as the set of parts connected by n_j. *Connectivity* $\lambda_j = |\Lambda_j|$ of a net n_j denotes the number of parts connected by n_j. A net n_j is said to be *cut* if it connects more than one part (i.e. $\lambda_j > 1$), and *uncut* otherwise (i.e. $\lambda_j = 1$). The cut and uncut nets are also referred to here as

external and *internal* nets, respectively. The set of external nets of a partition Π is denoted as $\mathcal{N}_E$. There are various cutsize definitions [10, 35] for representing the cost $\chi(\Pi)$ of a partition Π. Two relevant definitions are:

$$\text{(a)} \quad \chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} c_j \qquad \text{and} \qquad \text{(b)} \quad \chi(\Pi) = \sum_{n_j \in \mathcal{N}_E} c_j(\lambda_j - 1) \tag{2.2}$$

In the first definition, cutsize is equal to the sum of costs of the cut nets, whereas in the second one, the cost of each net n_j is multiplied by $(\lambda_j - 1)$. Similar to the graph model, the weight of a part is equal to the sum of vertex weights in that part. The same balance condition given in Section 2.1.1 also applies here. Based on these, the definition of hypergraph partitioning problem can be defined as dividing a hypergraph into two or more parts such that the cutsize is minimized and a given balance criterion among part weights is maintained. Note that the hypergraph partitioning problem is known to be NP-hard [29]. However based on some algorithmic heuristics, fast hypergraph partitioning tools exist such as PaToH [8, 7] and hMeTiS [24].

2.2.2 Column Net Model for Rowwise Decomposition

Catalyurek and Aykanat [8] propose the column net model for rowwise decomposition of sparse matrices. Here, the input matrix A is represented as a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{N})$, where each row i of A corresponds to $v_i \in \mathcal{V}$ and each column j corresponds to $n_j \in \mathcal{N}$. Net n_j contains the vertices corresponding to the rows which have a nonzero entry in column j (i.e. $v_i \in n_j$ if and only if $a_{ij} \neq 0$). Each vertex $v_i \in \mathcal{V}$ corresponds to atomic task i of computing the inner product of row i with column vector x. Thus, the computational weight w_i of a vertex $v_i \in \mathcal{V}$ is equal to the number of nonzeros in row i. Each net n_j can be considered as incurring the computations $y_i = y_i + a_{ij} \times x_j$ for all i values such that $v_i \in n_j$. So, net n_j denotes the set of atomic tasks that need x_j. Assume that this hypergraph is partitioned and each part is assigned to a processor. If net n_j is an internal net, then no interprocessor communication is required for this

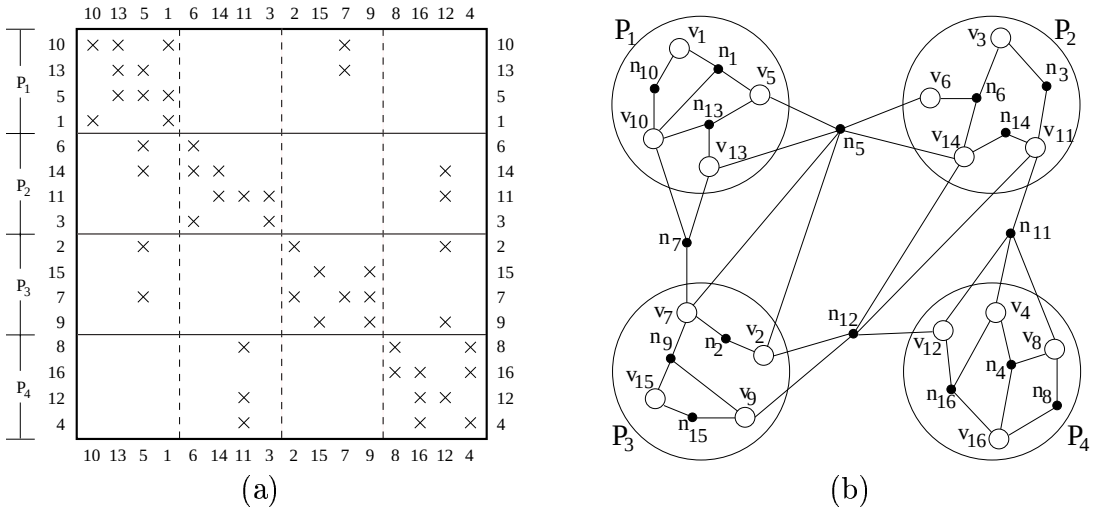

Figure 2.4: (a)Nonzero pattern of a 16×16 matrix after partitioning, (b) The corresponding column-net model

net. However, if it is a cut net, the processor that owns x_j is required to send one word (i.e. x_j) to each processor that owns at least one $v_i \in n_j$. Thus, the communication volume incurred by n_j is equal to $\lambda_{n_j} - 1$, where λ_{n_j} denotes the connectivity of the net n_j. So, by assigning unit costs to each net, minimization of communication volume for the column net model reduces to the problem of K-way hypergraph partitioning problem according to the cutsize definition given in 2.2. As an example, the 4-way partitioned matrix given in Figure 2.2(b) is redrawn in Figure 2.4(a). The corresponding hypergraph is illustrated in Figure 2.4(b). Observe that the communication volume depicted by this hypergraph is equal to $(\lambda_{n_5} - 1) + (\lambda_{n_7} - 1) + (\lambda_{n_{11}} - 1) + (\lambda_{n_{12}} - 1) = 2 + 1 + 1 + 2 = 6$. This is exactly equal to the number of nonzero columns in the off-diagonal blocks of matrix A.

Chapter 3

Communication Scheme for Parallel CG

The Conjugate Gradient method is an optimization technique that is used for the solution of linear equations of the form $\mathbf{Ax} = \mathbf{b}$. Here, the space of $\mathbf{x}$ vectors is searched in such a way that the objective function $f(x) = 1/2\langle \mathbf{x}, \mathbf{Ax} \rangle - \langle \mathbf{b}, \mathbf{x} \rangle$ is minimized. If matrix $\mathbf{A}$ is a *symmetric, positive definite* matrix, then this objective function has a global minimum where its gradient vector vanishes [30], i.e., $\nabla f(x) = \mathbf{Ax} - \mathbf{b} = 0$. Figure 3.1 displays a coarse-grain CG algorithm presented by Aykanat et.al. [1]. The rationale behind this coarse-grain formulation was to rearrange the computational steps of the original CG algorithm so that two inner products can be communicated within a single AABC operation at step 2. Note that the models we will propose in this chapter are also valid for the standard CG algorithm, which requires two separate AABC operations. However, for the ease of presentation we will stick to this coarse grain formulation.

Choose initial $\mathbf{x}$, let $\mathbf{r} = \mathbf{p} - \mathbf{A}\mathbf{x}$, and compute $rr = \langle \mathbf{r}, \mathbf{r} \rangle$.

while not converged
1. calculate $\mathbf{q} = \mathbf{A}\mathbf{p}$
2. form $\langle \mathbf{p}, \mathbf{q} \rangle$ and $\langle \mathbf{q}, \mathbf{q} \rangle$
3. compute α, β and rr as:
   - 3.1. $\alpha = \frac{rr}{\langle \mathbf{p}, \mathbf{q} \rangle}$
   - 3.2. $\beta = \alpha \frac{\langle \mathbf{q}, \mathbf{q} \rangle}{\langle \mathbf{p}, \mathbf{q} \rangle} - 1$
   - 3.3. $rr = \beta \cdot rr$
4. $\mathbf{x} = \mathbf{x} + \alpha \mathbf{p}$
5. $\mathbf{r} = \mathbf{r} - \alpha \mathbf{q}$
6. $\mathbf{p} = \mathbf{r} + \beta \mathbf{p}$

Figure 3.1: Coarse Grain CG algorithm

3.1 A New Formulation for Parallel CG Algorithm

For the following discussions, we assume that the input matrix $\mathbf{A}$ is rowwise partitioned among K processors based on the discussions in the previous chapter. Remember that, after the sparse matrix $\mathbf{A}$ is partitioned, the rows and columns of $\mathbf{A}$ are repermuted such that most of the nonzeros of a part $\mathbf{A_k}$ are on diagonal blocks (as in Figure 2.2), and they require no communication at all. Figure 3.2(a) illustrates such a row stripe $\mathbf{A_k}$ of an $N \times N$ matrix $\mathbf{A}$. Here, $\mathbf{A_k}$ has r_k rows and N columns, its diagonal block is shown with the shaded area, and its off-diagonal nonzero entries are marked with $\times$ symbols. Assume that this part is to be assigned to processor P_k. Then, for efficient storage and computations of the local matrix, it is possible to eliminate the columns in which $\mathbf{A_k}$ has no nonzero entry, and rename the remaining columns such that the format in Figure 3.2(b) is obtained. Observe that the number of columns in $\mathbf{A_k}$ is reduced to c_k, which is equal to number of nonzero columns. The vector $\mathbf{q_k}$ in Figure 3.2(c) and the vector $\mathbf{p_k}$ in Figure 3.2(d) correspond to the first r_k and c_k columns of the reordered $\mathbf{A_k}$, respectively. The shaded areas in these vectors indicate the elements that are computed locally by processor $\mathbf{P_k}$, and the unshaded area in $\mathbf{p_k}$ indicate the elements that P_k has to receive for the SpMxV operations.

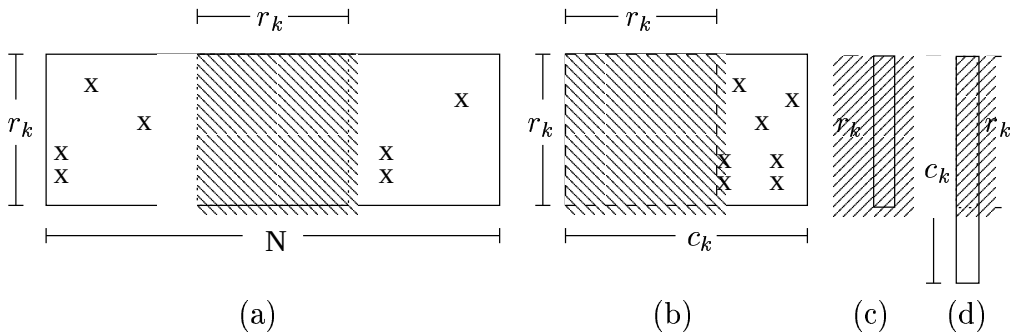

Figure 3.2: (a) Row stripe $\mathbf{A_k}$ of the global matrix, (b) reordered local matrix $\mathbf{A_k}$, (c) vector $\mathbf{q_k}$, (d) vector $\mathbf{p_k}$.

In the first step of the CG algorithm given in Figure 3.1, processor P_k computes the inner product of each row in $\mathbf{A_k}$ with column vector $\mathbf{p_k}$, and obtains the vector $\mathbf{q_k}$. Then in the second step, two global reduction operations are required for the inner product computations $\langle \mathbf{p}, \mathbf{q} \rangle$ and $\langle \mathbf{q}, \mathbf{q} \rangle$. In these operations, each processor P_k first computes the inner products of its local vectors, and then a global AABC operation is performed for two words. In the local computations, processor P_k is responsible for the inner product of first r_k elements of its vectors $\mathbf{p_k}$ and $\mathbf{q_k}$, i.e. the shaded areas in Figures 3.2(c) and 3.2(d). The third step of the algorithm requires computations on local scalar values to find α, β, and rr values. Using these values, linear vector operations are performed on local vectors in steps 4, 5, and 6. Here, $\mathbf{x}$ is the solution vector computed until this iteration, and $\mathbf{r}$ is the residual associated with $\mathbf{x}$ vector, i.e. $\mathbf{r} = \mathbf{b} - \mathbf{Ax}$. Note that the vector operations are performed for the r_k elements of the vectors also in these steps, After the sixth step, each processor P_k has the first r_k elements of its $\mathbf{p_k}$ vector, but it needs to receive the remaining $c_k - r_k$ elements from the respective processors, before the next iteration begins. So, point-to-point communications are performed between processors before the SpMxV of the next iteration begins.

In Figure 3.3, we propose a new formulation for the parallel CG algorithm. Observe that point-to-point communications are performed in step two, together with the inner product computations, so that the communication model we will propose in the subsequent sections is applicable for this application. The main difference in this algorithm is that the point-to-point communications are performed for the elements of vector $\mathbf{q}$, not vector $\mathbf{p}$. So, a processor P_k does not

while not converged

1. calculate $\mathbf{q_k} = \mathbf{A_k p_k}$
2. form $\langle \mathbf{p}, \mathbf{q} \rangle$ and $\langle \mathbf{q}, \mathbf{q} \rangle$ and communicate $\mathbf{q_k}$ values
3. compute α, β and rr as:
   - 3.1. $\alpha = \frac{rr}{\langle \mathbf{p}, \mathbf{q} \rangle}$
   - 3.2. $\beta = \alpha \frac{\langle \mathbf{q}, \mathbf{q} \rangle}{\langle \mathbf{p}, \mathbf{q} \rangle} - 1$
   - 3.3. $rr = \beta \cdot rr$
4. $\mathbf{x_k} = \mathbf{x_k} + \alpha \mathbf{p_k}$ /* *for a number of* r_k *elements* */
5. $\mathbf{r_k} = \mathbf{r_k} - \alpha \mathbf{q_k}$ /* *for a number of* c_k *elements* */
6. $\mathbf{p_k} = \mathbf{r_k} + \beta \mathbf{p_k}$ /* *for a number of* c_k *elements* */

Figure 3.3: Reformulated Parallel Coarse Grain CG algorithm

directly receive the $c_k - r_k$ elements of vector $\mathbf{p_k}$ (the unshaded area in Figure 3.2(d)) from the respective processors, but it computes these elements using the corresponding $\mathbf{q_k}$ elements it receives (see steps 5 and 6 in Figure 3.2). Note that for these computations, each of the local vectors $\mathbf{q_k}$ and $\mathbf{r_k}$ needs an extra space of size $c_k - r_k$.

Such a reformulation of the parallel CG algorithm requires redundant computations for $c_k - r_k$ elements in steps 5 and 6. However in the current technology, these computational costs are negligible compared to the costs due to communication of these elements. So the cost models we will propose in the following sections will be based only on the communication costs.

3.2 Communication Model Based on AABC Operations

The all-to-all broadcast operation (AABC) for a ring of n processors is given as follows [28]. In the beginning, each processor sends its own message to one of its neighbors. Then in the subsequent $n-2$ steps, it forwards the data received from one of its neighbors to its other neighbor. It is possible to extend this method to processor networks of 2-D meshes. For this, the given ring algorithm is used to perform AABC on each row of the mesh. Then, each processor consolidates the

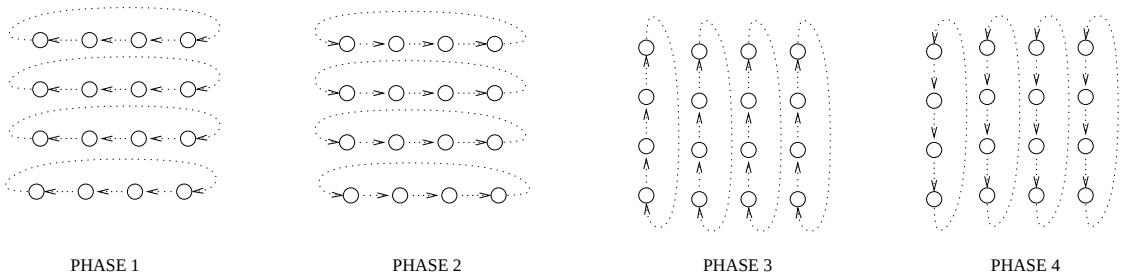

Figure 3.4: Four phases in AABC operation for a 4×4 processor mesh. Each phase corresponds to one direction.

messages it received into a single message. After that, the AABC algorithm is performed for each column of the mesh to broadcast these consolidated messages.

For the communication model we propose, we assume that the AABC operation for a ring is performed in two phases. For a ring of n processors, the first $\lfloor (n-1)/2 \rfloor$ steps will be performed in one direction in the first phase, and the remaining $\lceil (n-1)/2 \rceil$ steps will be performed in the opposite direction in the second phase. If we extend this method for 2D meshes, there will be four phases in the AABC algorithm, one for each direction: *left, right, up, down.* Figure 3.4 illustrates these phases for an example 4×4 mesh. Since $n = 4$ both for the horizontal and the vertical rings, there exist one step in phase 1, two steps in phase 2, one step in phase 3, and two steps in phase 4.

3.2.1 Embedding Point-to-Point Communications into AABC Operations

Assume that processors P_k and P_ℓ have the coordinates (x_k, y_k) and (x_ℓ, y_ℓ) respectively, for a mesh of $n \times m$ size. If these processors are not on the same row of the mesh (i.e. $y_k \neq y_\ell$), then the message from P_ℓ to P_k will first reach a processor P_m with coordinates (x_m, y_m), such that $y_m = y_\ell$ and $x_m = x_k$. This will occur in the phase corresponding to the *left* direction if $(x_\ell - x_m)\ mod\ n \leq \lfloor (n-1)/2 \rfloor$, and to the *right* direction otherwise. After that, P_m will forward this message towards P_k in the vertical direction. This direction will be *up* if $(y_m - y_k)\ mod\ m \leq \lfloor (m-1)/2 \rfloor$, and *down* otherwise. Based on these

considerations, the *direction* of P_k with respect to P_ℓ and the *distance* between these processors are defined as follows.

Definition 3.1 For an $n \times m$ mesh network, P_k is defined to be to the *left* of P_ℓ if it is the case that $(x_\ell - x_k)$ *mod* $n \leq \lfloor (n-1)/2 \rfloor$, and to the *right* otherwise. Similarly, P_k is defined to be to the *up* of P_ℓ if $(y_\ell - y_k)$ *mod* $m \leq \lfloor (m-1)/2 \rfloor$, and to the *down* otherwise.

Definition 3.2 The *distance* from P_ℓ to P_k, $d(P_k, P_\ell)$, is defined to be the number of steps required for a message originating from P_ℓ to reach P_k in the AABC operation. It is possible to define distance in the x-coordinate, $d_x(P_k, P_\ell)$, and distance in the y-coordinate, $d_y(P_k, P_\ell)$, such that $d(P_k, P_\ell) = d_x(P_k, P_\ell) + d_y(P_k, P_\ell)$. For a mesh of $n \times m$ processors, d_x and d_y between processors P_k and P_ℓ are defined as:

$$d_x(P_k, P_\ell) = \begin{cases} (x_\ell - x_k) \bmod n, & \text{if } P_k \text{ is to the } \textit{left} \text{ of } P_\ell \\ (x_k - x_\ell) \bmod n, & \text{otherwise} \end{cases} \tag{3.1}$$

$$d_y(P_k, P_\ell) = \begin{cases} (y_\ell - y_k) \bmod m, & \text{if } P_k \text{ is to the } \textit{up} \text{ of } P_\ell \\ (y_k - y_\ell) \bmod m, & \text{otherwise} \end{cases} \tag{3.2}$$

It is obvious that in an AABC operation, there is a regular communication pattern among the processors of the mesh. If processor P_ℓ is required to send a vector element to processor P_k, it can simply embed this data into this pattern, instead of sending it in a point-to-point fashion. For this purpose, the data is forwarded from P_ℓ to P_k by the intermediate processors in the appropriate phases of the AABC operation (i.e. in accordance with the direction of P_k with respect to P_ℓ).

Figure 3.5 illustrates such an example for a 6×4 mesh. According to the given definitions, P_k is to the *left* and to the *down* of P_ℓ. Furthermore, $d_x(P_k, P_\ell) = 2$ and $d_y(P_k, P_\ell) = 1$, hence $d(P_k, P_\ell) = 3$. The arrows in the figure indicate the route for the data to be communicated from P_ℓ to P_k. Note that in the AABC operations, the direction of communication will be *left* for 2 steps, *right* for 3 steps, *up* for 1 step, and *down* for 2 steps. The indicated route from P_ℓ to P_k

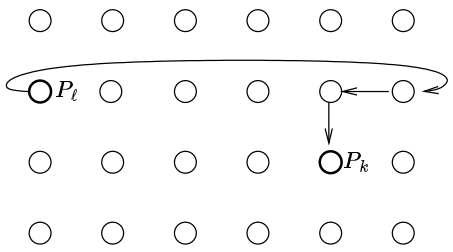

Figure 3.5: Communication of data from P_ℓ to P_k for a 6×4 mesh

can be realized *somehow* in the phases corresponding to these directions (e.g. the arrow entering P_k can be realized in one of the 2 steps in the phase corresponding to *down* direction).

Such a communication scheme has the advantage of avoiding start-up costs of the point-to-point communications, because no extra communication is initiated here. However, it has the disadvantage that a data item occurs more than once in the total communication volume. Namely, a single word to be communicated from P_ℓ to P_k incurs a cost of $d(P_k, P_\ell)$ instead of 1, for the total communication volume. So, it is possible to argue intuitively that the processors that are communicating vector elements should be close to each other as much as possible to minimize the overhead in terms of communication volume. The following section gives a cost model to formalize this concept.

3.3 Cost Model for Communication Volume Minimization

Assume that the input matrix $\mathbf{A}$ has been modeled as a hypergraph $\mathcal{H}_A$ based on the column net model, as explained in Chapter 2. Assume further that, a K-way partitioning Π has been given for $\mathcal{H}_A$, and each part has been mapped to a processor on the mesh. In this section, we give a cost function to model the communication volume for such a configuration, if the communication scheme proposed in the previous sections is to be used.

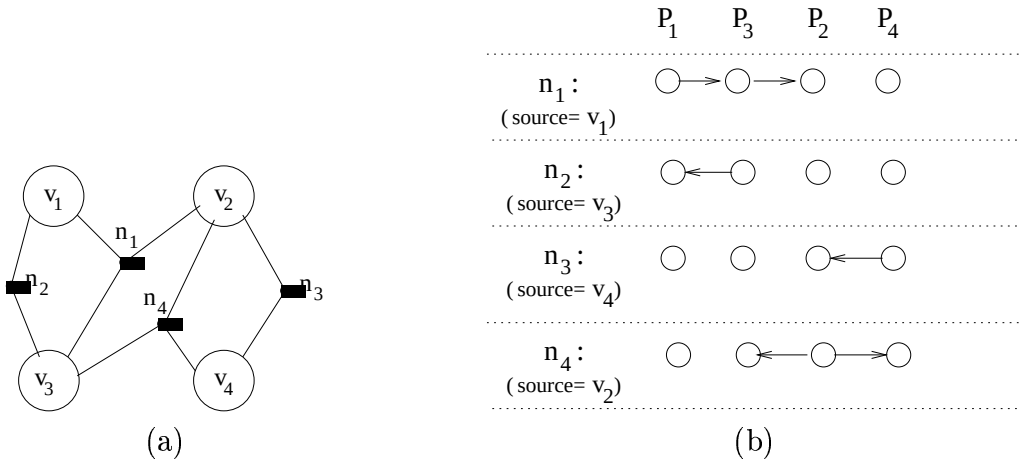

Figure 3.6: (a)Compressed hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$, (b) Communication requirements for each net in $\mathcal{N}_C$ on a 4-processor ring

For this purpose, we first define a *compressed* hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$ using the original hypergraph $\mathcal{H}_A = (\mathcal{V}_A, \mathcal{N}_A)$ and the partitioning information Π. For each external net $n_a \in \mathcal{N}_A$, a net n_c is created in $\mathcal{H}_C$ such that $pins[n_c]$ contains the corresponding parts of the vertices in $pins[n_a]$, and *size* of n_c is equal to *connectivity* of n_a. So, the vertex set $\mathcal{V}_C$ has K vertices in it, each one corresponding to a part in Π.

Observe that $\mathcal{H}_C$ models the communication requirements among the given parts in a more concise way compared to $\mathcal{H}_A$. Here, each net $n_c \in \mathcal{N}_C$ corresponds to a single vector element to be communicated among the parts in $pins[n_c]$. However for the cost model we are to propose, we need to know the *source* part in $pins[n_c]$, i.e. the processor that will send the vector element to the remaining parts in $pins[n_c]$. For this reason, the source part associated with each net is also indicated in our hypergraph model.

Figure 3.6(a) gives the compressed hypergraph $\mathcal{H}_C$ corresponding to the hypergraph $\mathcal{H}_A$ given in Figure 2.4. Note that v_1-v_4 in $\mathcal{H}_C$ correspond to the parts P_1-P_4 in $\mathcal{H}_A$. Also, for the external nets n_5, n_7, n_{11}, and n_{12} in $\mathcal{H}_A$, the nets n_1, n_2, n_3, and n_4 are created respectively in $\mathcal{H}_C$. Assume that these parts are mapped to a ring of processors in the order: P_1-P_3-P_2-P_4. Assume further that the source vertices of n_1-n_4 are v_1, v_3, v_4, and v_2 respectively. Figure 3.6(b) illustrates the communications incurred by each net in such a configuration. For

instance, the vector element corresponding to n_1 is required to be sent twice in the right direction. Note that P_3 is not only a destination for this net, but also a midway in the route from P_1 to P_2. So, when P_3 receives the data, it should both copy it into its local vector, and forward it to the right. In this scheme, the communication volume incurred by n_1 is equal to 2 words. Although n_4 seems to incur a different case (i.e. data is sent once to the left, once to the right), the incurred communication volume is again 2 words. The following lemma gives a formal way to express the cost of a net for a ring of processors.

Lemma 3.1 For a given compressed hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$, assume that the parts corresponding to the vertices in $\mathcal{V}_C$ are mapped to a rowwise (columnwise) ring of processors. Then the cost of a net $n \in \mathcal{N}_C$ in terms of communication volume is equal to the *distance* between the *leftmost* (*upmost*) and the *rightmost* (*downmost*) parts that are in $pins[n]$. Note that here, the directions are with respect to the *source* part.

Observe that, the position of the source vertex is not important for cost calculations. The reason is that, the distance between the leftmost (upmost) and the rightmost (downmost) parts is equal to the number of leftwise (upwise) steps plus the number of rightwise (downwise) steps. Another point to note about is that the intermediate (i.e. neither the leftmost (upmost) nor the rightmost(downmost)) parts do not have an effect on the cost. This is because, they simply keep a copy of the vector element while they are forwarding them to their neighbors.

We can extend this cost definition for 2D meshes as follows.

Lemma 3.2 For a given compressed hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$, assume that the parts corresponding to the vertices in $\mathcal{V}_C$ are mapped to a 2D mesh such that each v_k has the coordinates (x_k, y_k). Assume further that the coordinates for the source vertex of net n is given as (x_s, y_s). Then it is possible to find the cost of n in terms of communication volume as follows:

1. Ignore the y coordinates (i.e. treat all parts as if on the same row), and find the cost for the rowwise ring.

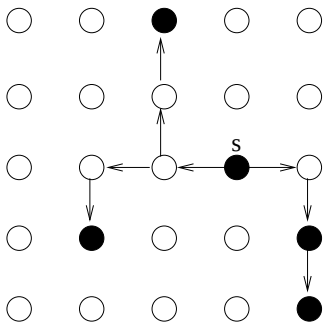

Figure 3.7: Cost calculation for a net on a 5×5 mesh. The dark circles indicate *pins* of n, and the source is the one with an 's' mark in it.

2. For each column j in which there exists at least one vertex $v_k \in pins[n]$ (i.e. $x_k = j$), assume that there is a source at (j, y_s), and calculate the cost for the columnwise ring $x = j$.

3. Add all the costs in steps 1 and 2, and find the total cost for net n.

Intuitively, this method is based on the AABC operation for 2D meshes. Remember that in the beginning, AABC algorithm is performed for each row of the mesh. Then, each processor consolidates the messages it received into a single message and broadcasts it to the processors in the same column. Similarly, the vector element corresponding to net n is first distributed in the row $y = y_s$. Then each vertex $v_k \in pins[n]$ receives the vector element in the columnwise direction from the processor that has the coordinates (x_k, y_s). Figure 3.7 shows an example of cost calculation for a net n on a 5×5 processor mesh. Here assume that *pins* of n are indicated with dark circles, and the source part is marked with 's'. The cost for n can be found with the method given in Lemma 3.2 as: $3+1+2+2 = 8$.

Chapter 4

Minimization of Communication Volume

In the previous chapter, we have given a novel cost model for the communication volume requirements of the CG type iterative algorithms. In this chapter, we give algorithms that can be used to minimize these costs. For the purpose of assigning matrix partitions to processors effectively, a two-phase approach is used. In the first phase, the hypergraph partitioning tool PaToH[8] is used to partition the input matrix such that the total number of vector elements that are required to be communicated is minimized. In the second phase, each part is mapped to a processor based on the distances between processors, such that the volume of communications embedded into the reduction operations is tried to be minimized. For this, the iterative improvement heuristics given by Kernighan and Lin [27] is used to minimize the function $\sum_j c(n_j)$, where $c(n_j)$ denotes the cost of net n_j as defined in Section 3.3.

After partitioning the input matrix and assigning them to processors, it is possible to schedule the embedded communications such that they are performed concurrently among different processors. Such a scheduling is important mainly because of the following expectation: *A successful partitioning and mapping of the matrix provides that the processors that have a large number of words to be*

communicated between them are close to each other, and the ones with small number of words are far away. In other words, small packets are expected to require more steps to reach their destinations than the large packets. So, instead of sending a large packet as a whole in one step, it is possible to divide it and send it in multiple steps so that it can be communicated concurrently with other packets. For the aim of such a scheduling, we give a simple algorithm based on a greedy heuristic.

In the subsequent sections, we outline the concepts used in PaToH for hypergraph partitioning, then we briefly explain the KL based iterative improvement heuristics, then finally we give the heuristics we use for scheduling.

4.1 Hypergraph Partitioning

PaToH is a multilevel hypergraph partitioning tool developed by Catalyurek and Aykanat [8]. Here, the K-way hypergraph partitioning problem is solved by recursive bisection method. That is, first a 2-way partition of the input hypergraph is obtained, and then this bipartition is further partitioned in a recursive manner. The multilevel hypergraph bisection algorithm used in PaToH consists of 3 phases: coarsening, initial partitioning, and uncoarsening.

In the coarsening phase, the given hypergraph $\mathcal{H} = \mathcal{H}_0 = (\mathcal{V}_0, \mathcal{N}_0)$ is coarsened into a sequence of smaller hypergraphs $\mathcal{H}_1 = (\mathcal{V}_1, \mathcal{N}_1)$, $\mathcal{H}_2 = (\mathcal{V}_2, \mathcal{N}_2)$, ..., $\mathcal{H}_m = (\mathcal{V}_m, \mathcal{N}_m)$ satisfying $|\mathcal{V}_0| > |\mathcal{V}_1| > |\mathcal{V}_1| > \ldots > |\mathcal{V}_m|$. This coarsening is achieved by coalescing disjoint subsets of vertices of hypergraph $\mathcal{H}_i$ into multinodes such that each multinode in $\mathcal{H}_i$ forms a single vertex of $\mathcal{H}_{i+1}$. The weight of each vertex in $\mathcal{H}_{i+1}$ becomes equal to the sum of its constituent vertices of the respective multinode in $\mathcal{H}_i$. The net set of each vertex in $\mathcal{H}_{i+1}$ becomes equal to the union of the net sets of the constituent vertices of the respective multinode in $\mathcal{H}_i$. Here, multiple pins of a net $n \in \mathcal{N}_i$ in a multinode cluster of $\mathcal{H}_i$ are contracted to a single pin of the respective net $n' \in \mathcal{N}_{i+1}$ of $\mathcal{H}_{i+1}$. Furthermore,

the single-pin nets obtained during this contraction are discarded. The coarsening phase terminates when the number of vertices in the coarsened hypergraph reduces below 100 (i.e. $|\mathcal{V}_m| \leq 100$).

After the coarsest hypergraph $\mathcal{H}_m$ is obtained, the initial partitioning phase is applied. The aim of this phase is to find a bipartition Π_m on the coarsest hypergraph $\mathcal{H}_m$. After that, the multilevel uncoarsening phase starts for $\mathcal{H}_m$. Here, at each level i (for $i = m, m-1, \ldots, 1$), bipartition Π_i found on $\mathcal{H}_i$ is projected back to a bipartition Π_{i-1} on $\mathcal{H}_{i-1}$. The constituent vertices of each multinode in $\mathcal{H}_{i-1}$ is assigned to the part of the respective vertex in $\mathcal{H}_i$. Note that, at this point Π_{i-1} has the same cutsize with Π_i. Then, before the next level of the uncoarsening phase, Π_{i-1} is refined by running a Boundary FM hypergraph partitioning algorithm on $\mathcal{H}_{i-1}$ starting from the initial bipartition Π_{i-1}. These steps continue until a bipartitioning for the fine-grain hypergraph $\mathcal{H}_0$ is obtained.

4.2 Mapping Parts to Processors

Kernighan-Lin (KL) based heuristics are widely used for graph/hypergraph partitioning problems because of their short run-times and good quality results. The KL algorithm is an iterative improvement heuristic originally proposed for graph bipartitioning [27]. The KL algorithm starts from an initial bipartition, then performs a number of passes until it finds a bipartitioning for which a given cost function (e.g. cutsize) has a local minimum.

Here, each pass consists of a sequence of vertex swaps. In the beginning of a pass, the swap gains for each pair of vertices are computed. Note that the swap gain corresponding to the vertices v_k and v_ℓ is equal to the decrease in the global cost function if these two vertices are to be swapped. After that, the vertices for which the swap gain is maximum are swapped, and the swap gains of the remaining vertices are updated. These operations continue until each vertex has been swapped once, so that a sequence $1 \ldots n$ of vertex swaps is obtained. Then, an m value ($m \leq n$) is found such that $G = \sum_i^m g_i$ is maximum, where g_i

denotes the swap gain value in step i of the sequence. At the end of the pass, only the swaps in the sequence $1 \dots m$ are accepted (i.e. the swaps $m+1 \dots n$ are restored); so the net gain of the pass becomes equal to G. Then, a new pass is performed if the net gain G of the current pass is positive.

For part-to-processor mapping problem, we use this heuristic algorithm. For this, we define the cost function as $\sum_j c(n_j)$, where $c(n_j)$ denotes the cost of net n_j as defined in Section 3.3. We also define a different semantics for the vertex swap operations described above. In our case, the swap of vertices v_k and v_ℓ corresponds to swapping the parts assigned to processors P_k and P_ℓ. In the beginning of the algorithm, we choose a random one-to-one mapping of parts to processors. Then a number of passes are performed based on the definitions we have given.

4.3 Communication Scheduling

After the parts are assigned to processors, each processor finds a scheduling of the communications that it shall perform during the CG iterations. Recall that, for 2D meshes there will be four communication phases, one for each direction. In Figure 4.1, a scheduling procedure is given for the phase corresponding to the *left* direction. It is straightforward to extend this procedure also for the other directions. Note that, these operations are performed only once, before the CG algorithm starts. The purpose here is to determine the communication pattern that will be used in each iteration of the CG.

Initially, each processor creates a data structure PACKETS, which stores the information of the vector elements to be sent to each processor. Here, the data contents are stored together with the information of source and destination processors. Note that, a vector element might have multiple destination processors if its corresponding net is connected to more than two parts. After the initialization, the communication pattern is determined for each step of the AABC operation. For this, the number of words to be sent at each step is determined globally,

```
Initialize PACKETS based on the communication requirements
for each step i of the AABC operation
    Globally determine S_i          /* S_i is the max no. of words to be sent at step i */
    s_i = min(S_i, |PACKETS|)       /* |PACKETS| is the no. of words in PACKETS */
    Starting from the packets that have the leftmost destinations
      extract s_i words from PACKETS and put into SENDBUF
    Send SENDBUF to the left neighbour
    Receive RECVBUF from the right neighbour
    for each pckt ∈ RECVBUF
        if a destination of pckt is this processor
            copy data contents of pckt into the local vector
        if there exists a destination of pckt that is not this processor
            insert pckt into PACKETS
```

Figure 4.1: The pseudo code for scheduling the communications in the *left* direction

with the aim of high degree of concurrency. Here, each processor computes an s_i value that indicates the number of vector elements that it will send to its neighbor. While selecting the vector elements, the packets are prioritized according to the rowwise distances of their destination processors. Here, the packets with leftmost destinations are assigned the highest priorities, the reason of which will be described in the subsequent paragraphs. Then s_i number of words are sent to the left neighbour, together with the relevant packet information. Once the information for packets is received from the right processor, the communication pattern is determined according to the destinations of the received packets (i.e. whether to copy into the local vector, or continue transmission).

We determine the maximum number of words S_i to be sent at step i, based on the following considerations. For each processor P_k, there exists a lower bound $s_i^{min}(P_k)$ for the number of words it must send at step i. That is because, the number of steps required to send a packet to a *distant* processor might exceed the number of remaining steps if it is not sent in time. So, a lower bound S_i^{lb} for the globally determined S_i becomes $\max_k (s_i^{min}(P_k))$. Furthermore, there is an upper bound $s_i^{max}(P_k)$ for the number of words that processor P_k can send at step i. This is due to the number of available packets in the data structure *PACKETS* at step i. So, $\min_k (s_i^{max}(P_k))$ becomes the upper bound S_i^{ub} for the global S_i, if all the processors are to send the same amount of data at step i (i.e. highest level

of concurrency is to be obtained). Note that, if S_i is assigned a value smaller than S_i^{ub}, the highest degree of concurrency among processors can be obtained. However, S_i^{lb} is a strict lower bound for S_i, and it is possible that $S_i^{lb} > S_i^{ub}$. In that case, S_i must be assigned a value larger than S_i^{ub}, so the degree of concurrency is reduced among communicating processors. Our approach to assign a value for S_i is greedy in nature: $S_i = \max\left(S_i^{lb}, S_i^{ub}\right)$. Here, the number of words to be sent at each step i is selected as large as possible, as long as the highest degree of concurrency is obtained. However, in the situations that the highest degree is not possible, the deviation among the number of words to be sent by each processor at step i is kept as small as possible (i.e. by selecting $S_i = S_i^{lb}$).

In fact, for this greedy approach be successful, the packets that are to be inserted (via communications) into the local data structures *PACKETS* should be done so as early as possible. It is obvious that such an approach provides higher degrees of freedom while trying to balance the communications at each step. For this purpose, in the procedure given in Figure 4.1, the leftmost processors are given the highest priorities while selecting the packets to be sent. The intuition behind this can be understood by the following example. Assume that leftwise communications are to be scheduled for the ring of processors: $\ldots - P_k - P_{k+1} - P_{k+2} - \ldots$. Assume further that P_{k+2} is required to send two separate packets with destinations P_{k+1} and P_k. Here, if it chooses to send the packet with destination P_{k+1} in the first AABC step, then P_{k+1} will simply copy contents of this packet into its local vector, without adding it to its *PACKETS*. However, if P_{k+2} chooses to send the packet with destination P_k, then P_{k+1} shall insert this packet into its *PACKETS* before the second iteration. So, P_{k+1} will have a higher degree of freedom starting from the second iteration, since it has more number of available packets. Observe that, sending the packets with farther destinations in the earlier AABC steps provides that more packets are available in the processors, starting from the early AABC steps.

Note that the procedure we have given for communication scheduling problem depends on greedy heuristics and does not claim to be optimal. It is possible to devise more complex algorithms for this problem, however we show in Chapter 5 that even this simple heuristic based algorithm gives good results.

Chapter 5

Experimental Results

We have performed experiments to test the validity of the models we have proposed. For this, we have used PaToH [8] as the hypergraph partitioning tool, and implemented the part-to-processor mapping and scheduling algorithms explained in Chapter 4 as preprocessing tools. We have also implemented the parallel coarse grain CG algorithm given in Figure 3.3 using the proposed communication scheme, in which the point-to-point communication operations are embedded into the reduction operations.

For our experiments, we have used the structurally symmetric test matrices given in Table 5.1. As the parallel system, we have used a cluster of personal computers. Each node in the system has Intel Pentium II 400 MHz CPU, 64 MB PC100 RAM, 6GB UDMA IDE hard drive and Intel EtherExpress Pro 10/100 NIC. The interconnection network is a 3COM SuperStack II 3900 smart switch. The operating system used in this system is Debian GNU/Linux distribution. We have used the MPI message passing library to implement the parallel programs.

For the test matrices, the execution times of the serial CG algorithm are given in Table 5.2, together with the execution times of the preprocessing steps. Observe that the FM-based algorithm we have implemented for the part-to-processor mapping problem takes much longer time than the hypergraph partitioning tool

PaToH. The reason is that our main aim here was not to develop a general purpose software tool like PaToH, but only to examine the validity of our models. So, our main concern here was the execution times for the parallel program, not the execution times of the preprocessing steps. However, it is possible to develop a more efficient program for the mapping problem, if it is to be used in real world applications.

To find the average communication requirements given in Table 5.3, we have executed the preprocessing steps 20 times, each with the same parameter values, but with different random seeds. Note that, the communication requirements for the reduction operations are not included in this table. The results are given for each test matrix and each mesh configuration (with different number of processors). Here, point-to-point communication scheme indicates the conventional algorithm, and the embedded communication scheme indicates the proposed algorithm.

Assuming that the computational loads of the processors are balanced, two main factors effect the communication times in the conventional algorithm: *the maximum number of messages sent by a processor* and *the maximum volume of communication handled (sent or received) by a processor*. Note that these two costs are not necessarily cumulative, because it is possible that the processor that sends the maximum number of messages might be different from the processor that handles the maximum volume. Furthermore, even in the same processor, the operation of preparing a message can be performed concurrently with handling a communication that is already started. However, we assume that a processor can not prepare more than one message concurrently, and we assume a single port communication model for the network (i.e. a processor can not send/receive more than one words at the same time). So, we can say that the dominant one of these two factors determines a (not so tight) lower bound for the overall communication cost. For instance, for the test matrix *1138 BUS* and the mesh configuration 6×6, the maximum communication volume is less than twice the maximum number of messages. Assuming that $t_s/t_w >> 2$, the communication time will be dominated by the message start-up costs. So, it is possible to argue that an *average* lower bound for the overall communication time will be 10.4 t_s

for this configuration.

In the embedded communication scheme, remember that no separate point-to-point communication operation is performed for the transmission of vector elements. Instead, each processor sends the relevant vector elements to its neighbor processors in the appropriate steps of the AABC operation. By using the scheduling algorithm given in Chapter 4, each processor determines which vector elements to send in each step, before the CG algorithm starts. We assume that, the communications performed during one step of the AABC operation are done concurrently by different processors. So the extra communication cost for one step of the AABC operation is determined by the processor that handles the maximum communication volume in that step. The overall extra communication cost due to the transmission of vector elements is assumed to be the sum of costs for each step. Based on this consideration, the overall communication cost for this scheme can be modeled as $t_w \sum_i m_i$, where m_i is the maximum communication volume handled by a processor at step i. The "concurrent" column in Table 5.3 gives the corresponding values of $\sum_i m_i$ for each test matrix and each mesh configuration.

To test the practical validity of these considerations, we have executed the conventional and the reformulated parallel CG algorithms for each test matrix. However, since only 24 processors were available for our experiments in the parallel system, we could obtain timing statistics for only the mesh configurations 4×4 and 6×4. The average results obtained by executing the parallel programs 20 times, each for 2500 iterations, are reported in Table 5.4. Here observe that, the execution times for different parts (i.e. reduction, p-p communications, etc.) of each program are reported. The methodology we have used to obtain such data was to execute the programs for only one part (e.g. p-p communications) in all iterations. Since the communication patterns are the same in all iterations, eliminating the computations for measuring the execution times of communication operations does not pose an important problem. However, such an approach eliminates the communication synchronization costs that would have been incurred due to the imbalance of the computational loads among different processors. In fact, such a methodology has the advantage that the effect of

our models can be observed more clearly, independent of other factors such as computational imbalance. Note that in reality, the synchronization costs will be effective in the communication times of both the conventional and the proposed schemes.

For the execution times given in Table 5.4, we have performed the preprocessing steps only once for each matrix and mesh, to obtain the communication requirements in Table 5.5. The "p-p lbnd" column in Table 5.4 denotes the lower bound estimated for the communication times of the conventional scheme due to the start-up costs. That is, each entry in this column is equal to $n_{max}t_s$, where n_{max} is the maximum number of messages handled by a processor, given in the corresponding entry of Table 5.5. To determine a suitable t_s value for the parallel system, we assumed that the p-p communication times for the matrices *GR 30 30*, *1138 BUS*, and *NOS7* are completely determined by the start-up costs, because the maximum communication volumes are too small compared to the maximum number of messages. Based on this assumption, we divided the communication time measured for each matrix and mesh by the maximum number of messages reported for this configuration. We have obtained values very close to each other for each case, which is in accordance with our assumption. As the t_s value to be used for the calculations of lower bounds, we have chosen the minimum one among these values, which is for matrix *NOS7* and mesh 6×4. That is, we have taken $t_s = 1.193/14 = 0.0852msec$. It is possible to observe the values in the columns "p-p comm" and "p-p lbnd" of Table 5.4, and see that the point-to-point communication times are very close to the estimated lower bound values in most cases. So, we can argue that the communication times are mainly dominated by the message start-up costs. Observe that this is the case especially for the matrices with small communication volume requirements (see Table 5.5). However, for the matrices such as *CRE-B* and *CRE-D*, the large communication volume requirements might make the actual communication times deviate from the estimated lower bound values.

It can be observed from Table 5.4 that the reduction operations in the conventional scheme require nearly the same amount of time for different matrices, if the mesh configurations are the same. We can say that the times required for the

AABC of two floating-point words on mesh configurations 4×4 and 6×4 are 0.746 and 1.016 respectively, on the average. However for the model we propose, the time requirements for the reduction operations are larger than these values due to the extra volume of communications embedded into the reduction operations. The "total" column under "OVHD CALCULATED" in Table 5.4 gives the amount of increase in the execution times of these reduction operations. Recall that, we have modeled the extra communication cost in a reduction operation as $t_w \sum_i m_i$, where m_i is the maximum communication volume handled by a processor at step i of the reduction operation. To show the validity of this cost model, we calculate a t_w value for each matrix and mesh, by dividing the corresponding value in the "total" column of Table 5.4 by the value in the "concurrent" column of Table 5.5. These values are reported in the column "per word" of Table 5.4. Observe that, except for the matrices that have very small $\sum_i m_i$ values (i.e. *GR 30 30*, *1138 BUS*, and *NOS7*), the calculated t_w values are close to the given t_w value of the parallel system: $0.5 \mu sec$. This shows that the cost model we have given can be used effectively to estimate the costs of the embedded communications.

Since the number of processors available for our experiments was 24, we could not obtain timing statistics for the mesh configurations 6×6 and 8×8. However, our experiments on the 4×4 and 6×4 meshes have shown that the following two arguments are valid:

- The time requirement for point-to-point communications in the conventional scheme is at least equal to $n_{max} t_s$, where n_{max} is the maximum number of messages sent by a processor.
- The increase in the execution time of the reduction operations in the embedded communications scheme is approximately equal to $t_w \sum_i m_i$, where m_i is the maximum communication volume handled by a processor at step i of the reduction operation.

Based on these arguments, we can estimate the average communication costs of the conventional and the proposed schemes by using the average statistics

reported in Table 5.3. Each graph in Figures 5.1, 5.2, and 5.3 illustrates a comparison of the communication costs for a test matrix and for different mesh configurations. Here, the costs for the point-to-point communications are given as lower bound values. The previous discussions have indicated that the actual communication costs might be larger than these lower bound values, depending on the volume of communications handled. For the embedded communications scheme, we give the estimated communication costs due to the extra communicated vector elements in the reduction operations.

Observe in Table 5.3 that, as the number of processors increases, the maximum volume of communications handled by a single processor decreases in most cases. The reason for this is that, typically the total volume of communications does not increase as much as the increase in the number of processors. So, as the number of processors increases, the volume of communications handled by each processor tend to decrease. This results in a tendency of decreasing in the communication costs of our proposed scheme. A similar tendency is also expected for the volume of communications in the point-to-point communications scheme. However observe in Table 5.3 that, the maximum number of messages to be sent by a processor increases with increasing number of processors in nearly all cases. So, the lower bounds due to the message start-up costs increase, although the communication volumes tend to decrease. This means that, as the number of processors increase, the start-up costs become more and more dominant in the communication costs.

The effect of the embedded communications scheme can be observed in the graphs of Figures 5.1, 5.2, and 5.3. Note that in most cases, the embedded communication costs are smaller than even the lower bound values of point-to-point communications. However, for the matrices that have high communication volume requirements (e.g. *CRE-B, CRE-D, KEN-13*), the conventional scheme seems to perform better than the proposed scheme on small meshes. Despite this, as the mesh sizes increase for these matrices, the start-up costs increase considerably while the embedded communication costs tend to decrease. We can say that the proposed scheme outperforms the conventional scheme in terms of scalability characteristics even for these matrices.

Table 5.1: Properties of test matrices.

matrix name	description	number of rows/cols	number of nonzeros				
			total	avg. per row/col	min. per row/col	max. per row/col	std. dev
SHERMAN3	[12] 3D finite difference grid	5005	20033	4.00	1	7	2.66
KEN-11	[6] linear programming	14694	82454	5.61	2	243	14.54
NL	[9] linear programming	7039	105089	14.93	1	361	28.48
KEN-13	[6] linear programming	28632	161804	5.65	2	339	16.84
CQ9	[9] linear programming	9278	221590	23.88	1	702	54.46
CO9	[9] linear programming	10789	249205	23.10	1	707	52.17
CRE-D	[6] linear programming	8926	372266	41.71	1	845	76.46
CRE-B	[6] linear programming	9648	398806	41.34	1	904	74.69
BCSSTK17	[32] static analyses in structural eng.	10974	208838	39.00	1	150	15.00
BCSSTK24	[32] dynamic analyses in structural eng.	3562	81736	45.00	15	57	11.00
GR 30 30	[32] partial differential equations	900	4322	8.60	4	9	1.00
1138 BUS	[32] power system networks	1138	2596	3.60	2	18	1.80
NOS7	[32] linear equations in structural eng.	729	2673	6.30	4	7	0.72
S3RMT3M3	[32] structural mechanics	5357	106526	39.00	7	48	7.10

Table 5.2: Serial execution times for one iteration of the CGCG algorithm, and the execution times of the preprocessing steps.

name	serial exec. per iter. *(msec)*	mesh	PaToH *(sec)*	mapping *(sec)*
SHERMAN3	3.63	4×4	0.38	0.79
		6×4	0.43	1.77
		6×6	0.46	2.79
		8×8	0.52	20.21
KEN-13	32.57	4×4	2.64	10.77
		6×4	2.89	21.47
		6×6	3.23	49.01
		8×8	3.62	106.60
BCSSTK24	20.05	4×4	2.11	1.06
		6×4	2.12	1.76
		6×6	2.32	10.17
		8×8	2.65	37.03
1138 BUS	0.74	4×4	0.06	0.11
		6×4	0.10	0.18
		6×6	0.08	0.50
		8×8	0.09	1.64
S3RMT3M3	26.18	4×4	2.22	1.07
		6×4	2.50	2.41
		6×6	2.78	6.43
		8×8	3.14	35.37
CO9	33.87	4×4	3.37	5.72
		6×4	3.46	16.85
		6×6	3.65	39.15
		8×8	4.14	151.25
CRE-B	51.23	4×4	7.10	12.40
		6×4	7.14	26.67
		6×6	7.55	81.05
		8×8	8.42	209.40

name	serial exec. per iter. *(msec)*	mesh	PaToH *(sec)*	mapping *(sec)*
KEN-11	15.60	4×4	1.24	4.08
		6×4	1.37	11.03
		6×6	1.50	27.13
		8×8	1.70	58.32
BCSSTK17	53.99	4×4	5.74	2.24
		6×4	6.27	3.92
		6×6	6.88	10.84
		8×8	7.80	64.06
GR 30 30	1.06	4×4	0.10	0.22
		6×4	0.11	0.53
		6×6	0.12	3.39
		8×8	0.15	5.73
NOS7	0.69	4×4	0.08	0.51
		6×4	0.08	1.24
		6×6	0.10	2.65
		8×8	0.10	8.71
NL	15.61	4×4	1.57	7.63
		6×4	1.55	18.50
		6×6	1.70	24.74
		8×8	1.94	62.09
CQ9	30.00	4×4	2.54	3.47
		6×4	3.02	11.86
		6×6	3.06	34.40
		8×8	3.45	132.33
CRE-D	47.28	4×4	6.43	10.82
		6×4	6.79	37.20
		6×6	7.16	42.01
		8×8	8.26	153.84

Table 5.3: Average communication requirements for parallel CGCG algorithm

name	mesh	POINT-TO-POINT COMM. SCHEME				EMBEDDED COMM. SCHEME	
		# of mesgs per proc.		comm. volume		comm. volume	
		avg	max	total	max	total	concurrent
SHERMAN3	4×4	4.6	7.5	1256.2	122.8	1645.4	247.1
	6×4	5.5	9.3	1601.2	110.2	2273.4	235.6
	6×6	6.3	11.4	2019.7	92.4	3058.9	224.4
	8×8	7.1	13.1	2663.7	68.6	4537.9	191.4
KEN-11	4×4	13.6	15.0	9337.2	1715.8	16549.8	2348.8
	6×4	18.0	23.0	10182.0	1438.4	21071.2	2067.9
	6×6	23.1	34.9	11250.6	1214.0	26208.7	1875.8
	8×8	29.7	59.2	13471.5	795.8	36650.6	1519.7
KEN-13	4×4	14.1	15.0	16626.2	3837.5	31258.7	4737.1
	6×4	19.2	23.0	17867.0	2867.9	39310.8	3974.8
	6×6	24.8	35.0	19732.0	2184.8	48880.8	3605.7
	8×8	36.2	63.0	23133.9	1489.1	67563.9	2907.6
BCSSTK17	4×4	3.8	6.8	3579.2	376.5	4517.1	740.0
	6×4	4.3	7.9	4884.5	339.3	6683.6	750.3
	6×6	4.9	8.9	6513.4	285.2	9679.5	728.5
	8×8	5.4	9.9	9483.2	230.0	15849.6	671.5
BCSSTK24	4×4	4.3	7.6	2129.2	191.2	2681.1	376.0
	6×4	4.7	7.7	2909.9	186.6	3907.2	393.1
	6×6	5.0	8.6	3906.6	165.1	5594.1	401.6
	8×8	5.5	9.6	5965.2	148.6	9077.7	401.9
GR 30 30	4×4	4.4	7.1	437.1	39.0	536.8	71.6
	6×4	4.6	7.8	596.0	37.1	819.2	82.3
	6×6	4.9	7.9	778.8	32.0	1146.2	80.3
	8×8	5.4	8.1	1104.8	25.0	1767.1	73.8
1138 BUS	4×4	3.9	7.9	167.4	21.5	218.0	37.8
	6×4	4.1	10.3	218.4	20.1	315.4	37.8
	6×6	4.1	10.4	281.8	17.7	431.9	37.1
	8×8	3.9	9.7	409.5	17.5	688.0	36.5
NOS7	4×4	7.0	10.9	695.3	59.0	934.7	111.8
	6×4	7.8	13.3	862.8	51.3	1246.7	106.9
	6×6	8.4	14.3	1070.5	42.7	1695.9	99.8
	8×8	8.9	15.1	1416.3	32.7	2524.1	88.2
S3RMT3M3	4×4	4.0	6.8	2211.6	204.6	2722.8	384.6
	6×4	4.3	7.2	2948.1	184.1	4049.4	420.2
	6×6	4.5	7.4	3888.3	156.6	5672.5	421.2
	8×8	5.0	8.6	5884.2	145.2	9455.2	412.8
NL	4×4	13.6	15.0	8169.5	897.1	12281.2	1417.6
	6×4	18.2	22.3	9980.2	765.2	16517.7	1408.2
	6×6	22.0	31.0	11744.2	717.0	21321.0	1230.3
	8×8	27.0	45.4	12956.1	497.5	26285.4	937.9
CO9	4×4	12.9	15.0	10905.0	1619.8	17279.0	2556.8
	6×4	16.7	21.9	13466.0	1446.2	23121.2	2467.4
	6×6	19.0	29.5	16702.4	1187.3	30714.6	2194.6
	8×8	23.1	42.2	23473.3	1004.9	46148.7	2074.1
CQ9	4×4	12.2	14.9	9721.6	1513.9	15246.8	2245.0
	6×4	16.2	21.5	12341.8	1350.6	21080.8	2257.4
	6×6	19.2	30.0	14982.2	1072.8	27502.0	2003.1
	8×8	23.6	43.5	21576.5	951.5	41275.1	1933.6
CRE-B	4×4	12.8	15.0	19855.2	4383.9	26374.1	5057.6
	6×4	17.0	23.0	24142.8	3890.1	34846.0	4785.8
	6×6	21.4	34.9	29537.1	3531.2	46623.0	4730.5
	8×8	28.9	60.3	38610.8	2643.3	69977.5	4133.2
CRE-D	4×4	12.4	15.0	18089.4	4043.8	24517.2	4589.1
	6×4	16.7	23.0	22292.4	3746.8	32499.1	4447.5
	6×6	21.0	35.0	27053.2	3172.0	43431.0	4281.0
	8×8	29.4	60.9	36166.1	2450.0	67035.1	3637.1

In the "# of mssgs" column, "avg" and "max" denote the average and maximum number of messages, respectively, sent by a single processor. In the first "comm. volume" column, "total" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled (sent/received) by a single processor. In the "concurrent" column, it is assumed that communications are performed concurrently among different processors, and the actual communication volumes are calculated according to the schedulings of the communications.

Table 5.4: Average execution times of the parallel CGCG algorithms for one iteration

		POINT-TO-POINT COMM. SCHEME				EMBEDDED COMM. SCHEME		OVHD CALCULATED	
name	mesh	rdctn	p-p comm	p-p lbnd	overall	rdctn	overall	total	per word
SHERMAN3	4×4	0.745	0.605	0.597	1.462	0.896	1.178	0.150	5.47×10^{-4}
	6×4	1.018	0.884	0.852	1.935	1.145	1.457	0.129	5.51×10^{-4}
KEN-11	4×4	0.746	1.524	1.278	2.977	1.830	2.693	1.084	4.83×10^{-4}
	6×4	1.014	2.179	1.960	3.609	2.002	2.634	0.986	5.06×10^{-4}
KEN-13	4×4	0.746	1.705	1.278	4.559	3.171	4.887	2.425	4.81×10^{-4}
	6×4	1.016	2.302	1.960	4.305	3.042	4.250	2.026	4.86×10^{-4}
BCSSTK17	4×4	0.745	0.649	0.597	4.622	1.050	4.480	0.304	4.84×10^{-4}
	6×4	1.015	0.857	0.767	3.844	1.388	3.634	0.373	5.14×10^{-4}
BCSSTK24	4×4	0.746	0.738	0.682	2.514	0.985	2.197	0.239	5.98×10^{-4}
	6×4	1.016	0.659	0.597	2.359	1.260	2.099	0.244	6.10×10^{-4}
GR 30 30	4×4	0.747	0.638	0.597	1.354	0.809	0.892	0.063	7.00×10^{-4}
	6×4	1.015	0.623	0.597	1.596	1.074	1.133	0.058	6.67×10^{-4}
1138 BUS	4×4	0.745	0.778	0.767	1.447	0.779	0.836	0.033	8.46×10^{-4}
	6×4	1.016	1.031	1.023	1.891	1.051	1.095	0.035	8.54×10^{-4}
NOS7	4×4	0.747	0.951	0.937	1.665	0.826	0.884	0.080	7.41×10^{-4}
	6×4	1.018	1.193	1.193	2.068	1.097	1.137	0.081	8.27×10^{-4}
S3RMT3M3	4×4	0.745	0.571	0.511	2.725	1.004	2.547	0.258	6.01×10^{-4}
	6×4	1.015	0.627	0.597	2.575	1.239	2.358	0.223	5.52×10^{-4}
NL	4×4	0.747	1.372	1.278	2.972	1.559	2.405	0.813	5.54×10^{-4}
	6×4	1.016	1.995	1.875	3.435	1.850	2.465	0.834	5.18×10^{-4}
CO9	4×4	0.746	1.506	1.278	4.361	1.947	3.974	1.201	4.37×10^{-4}
	6×4	1.016	1.933	1.790	4.260	2.085	3.490	1.069	4.43×10^{-4}
CQ9	4×4	0.745	1.445	1.278	3.929	1.927	3.733	1.181	4.49×10^{-4}
	6×4	1.016	1.946	1.875	4.036	2.012	3.265	0.996	5.00×10^{-4}
CRE-B	4×4	0.745	1.839	1.278	5.796	3.126	6.515	2.380	5.16×10^{-4}
	6×4	1.014	2.502	1.960	5.583	3.346	5.628	2.330	4.97×10^{-4}
CRE-D	4×4	0.745	1.850	1.278	5.499	2.926	5.869	2.180	5.24×10^{-4}
	6×4	1.017	2.513	1.960	5.416	3.324	5.665	2.308	5.32×10^{-4}

In the columns labeled "rdctn", the execution times of the reduction operations in the original and reformulated schemes are given respectively. Similarly, the columns labeled "overall" denote the total execution times for one iteration. In the column "p-p comm", the execution times required for point to point communications are given. The "p-p lbnd" column gives the estimated lower bounds due to the message start-up costs. In the column "OVHD CALCULATED", the increase in the execution times of the reduction operations, due to the embeded communications are given. All the entries in the table are in terms of msec values.

Table 5.5: Actual communication requirements for the configuration used to collect the statistics of Table 5.4

name	mesh	POINT-TO-POINT COMM. SCHEME				EMBEDDED COMM. SCHEME	
		# of mesgs per proc.		comm. volume		comm. volume	
		avg	max	total	max	total	concurrent
SHERMAN3	4×4	4.2	7	1258	130	1566	274
	6×4	5.6	10	1622	111	2366	234
KEN-11	4×4	13.6	15	8464	1904	15802	2243
	6×4	16.6	23	9887	1506	20419	1948
KEN-13	4×4	13.6	15	16714	4010	31005	5038
	6×4	19.4	23	17585	2949	38402	4173
BCSSTK17	4×4	3.9	7	3336	318	4074	628
	6×4	4.2	9	4708	318	6807	725
BCSSTK24	4×4	4.0	8	2142	232	2838	400
	6×4	4.7	7	2914	186	3828	400
GR 30 30	4×4	4.0	7	447	44	606	90
	6×4	4.6	7	590	37	811	87
1138 BUS	4×4	4.1	9	163	22	214	39
	6×4	4.0	12	222	23	299	41
NOS7	4×4	6.9	11	723	64	950	108
	6×4	8.1	14	878	52	1256	98
S3RMT3M3	4×4	4.0	6	2196	192	2904	429
	6×4	4.1	7	3009	202	4080	404
NL	4×4	14.5	15	8261	828	12303	1467
	6×4	17.8	22	10242	899	16761	1609
CO9	4×4	14.1	15	11946	1824	18347	2746
	6×4	17.3	21	12440	1434	22164	2412
CQ9	4×4	12.9	15	11148	1703	16411	2630
	6×4	15.7	22	11415	991	19654	1993
CRE-B	4×4	13.8	15	19686	3586	25991	4616
	6×4	17.1	23	24933	3922	36534	4688
CRE-D	4×4	12.5	15	18052	3720	24044	4158
	6×4	14.9	23	20927	3729	30844	4336

In the "# of mssgs" column, "avg" and "max" denote the average and maximum number of messages, respectively, sent by a single processor. In the first "comm. volume" column, "total" denotes the total communication volume, whereas "max" denotes the maximum communication volume handled (sent/received) by a single processor. In the "concurrent" column, it is assumed that communications are performed concurrently among different processors, and the actual communication volumes are calculated according to the schedulings of the communications.

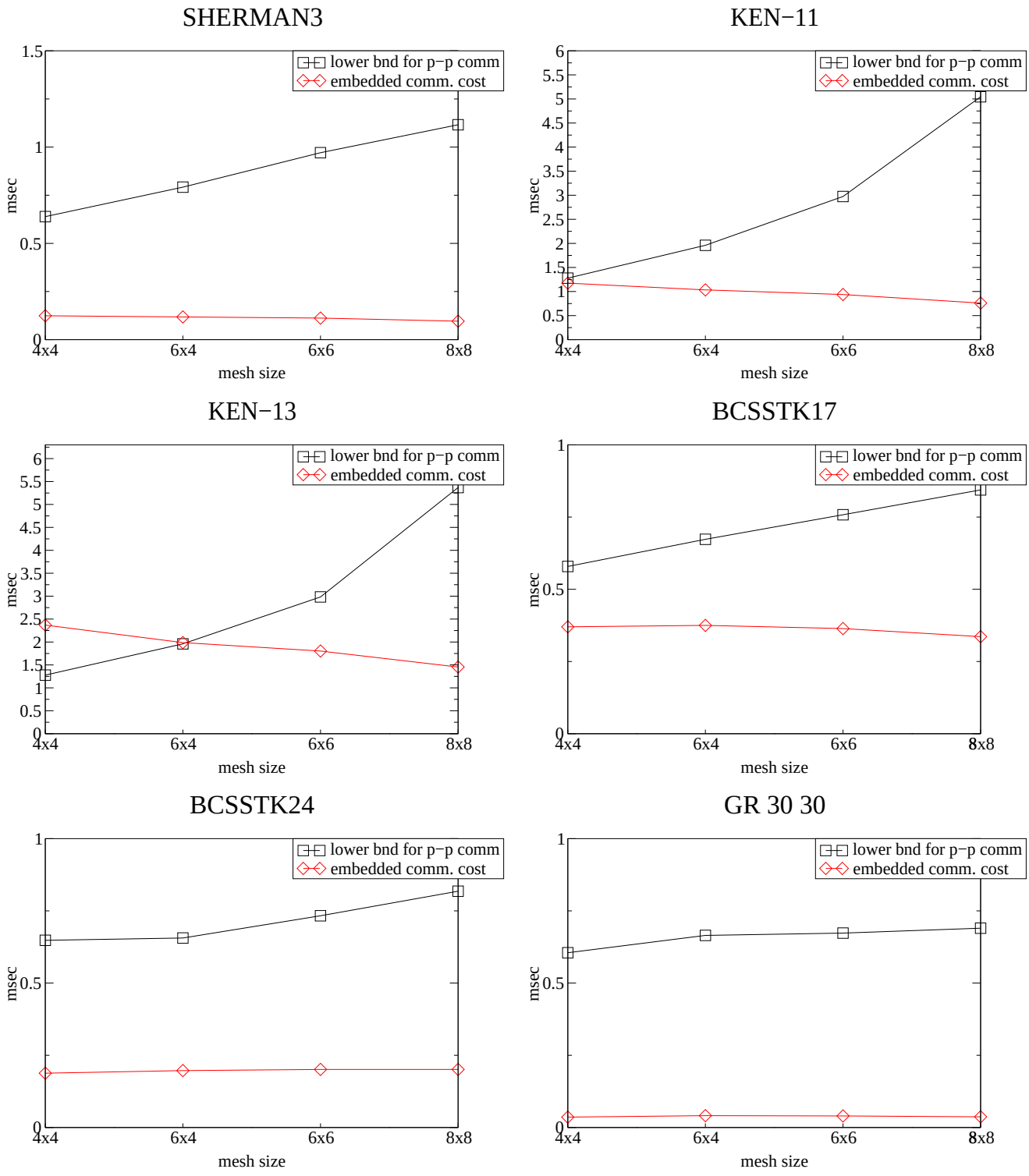

Figure 5.1: Comparison of the conventional and the proposed schemes in terms of estimated communication costs

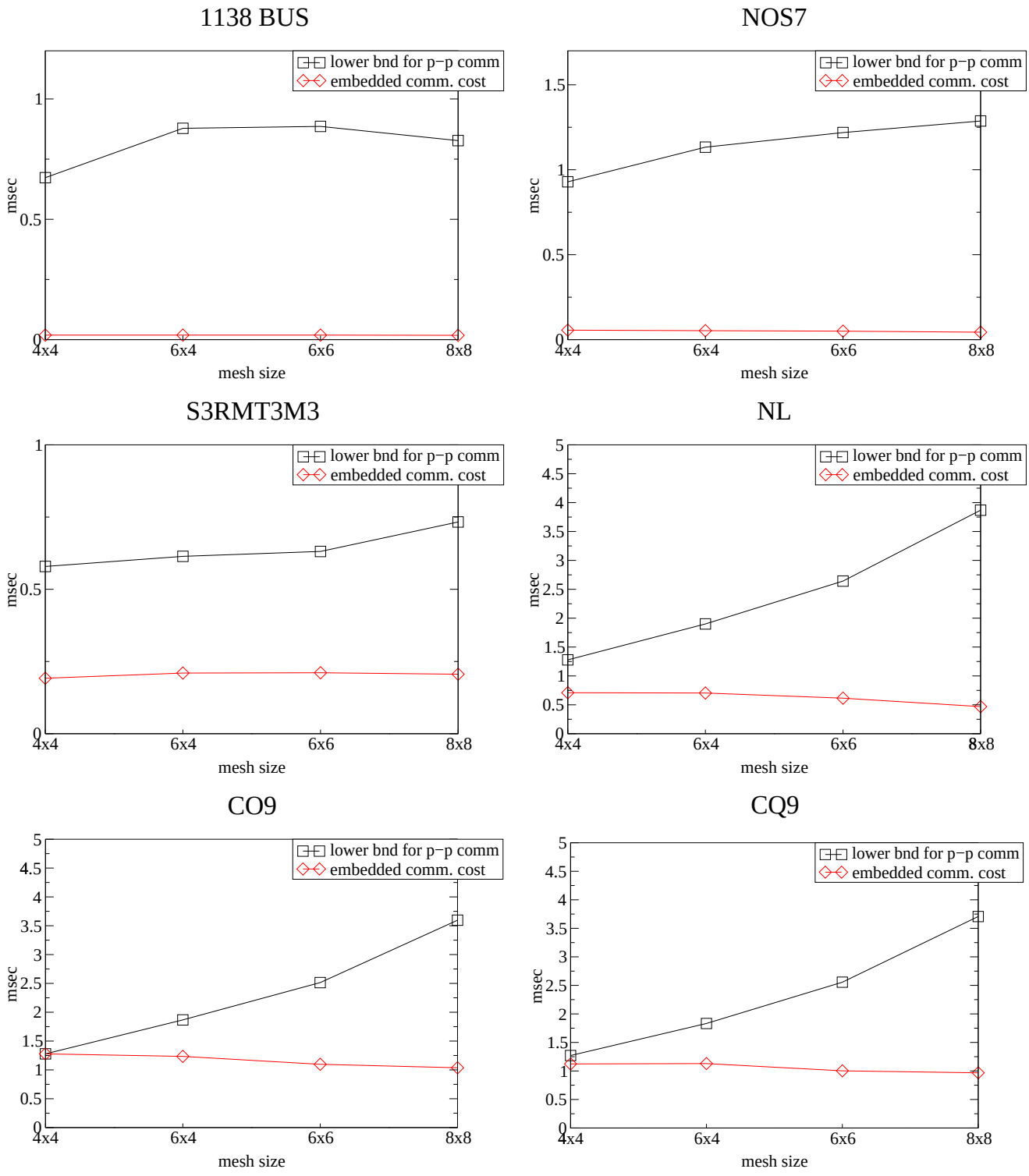

Figure 5.2: Comparison of the conventional and the proposed schemes in terms of estimated communication costs

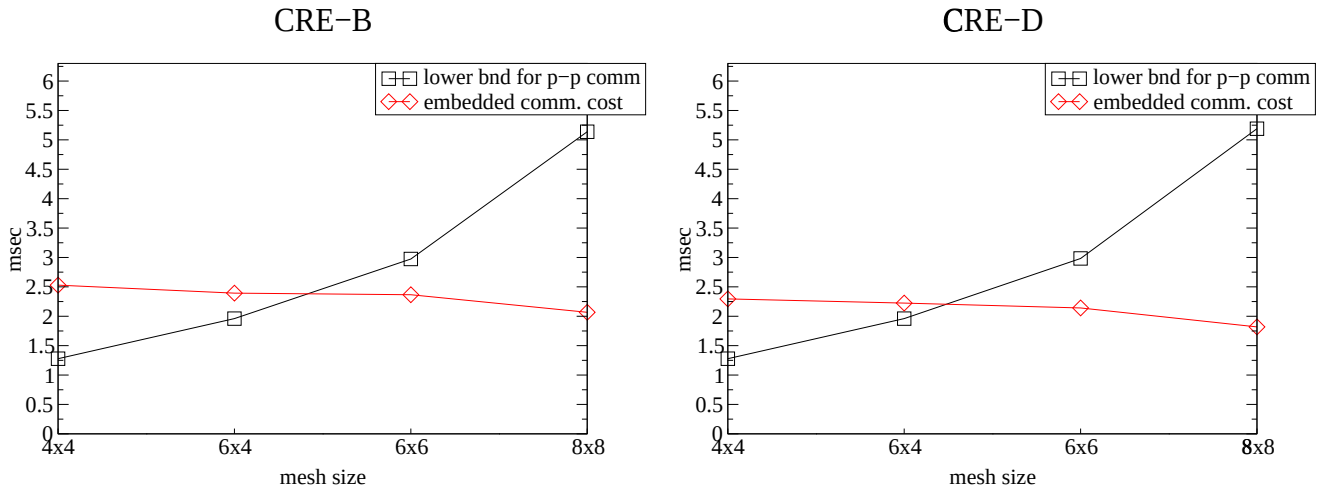

Figure 5.3: Comparison of the conventional and the proposed schemes in terms of estimated communication costs

Chapter 6

Scalability for Multidimensional Meshes

In this thesis, we have given models and algorithms for 2D meshes. However, there exist other network topologies for which the basic communication operations such as AABC require less number of communications. For instance, it is given in [28] that the AABC operation for square 2D meshes with a number of p processors require $2 \times (\sqrt{p} - 1)$ number of communications. However, for the *hypercube* topology with the same number of processors, this number is equal to $\log p$. Especially for the parallel systems with very large number of processors, the 2D meshes might be impractical for such operations. In this chapter, we give a generalization of the embedded communication model so that it is also applicable to d-dimensional meshes. Note that a *hypercube* is a special case of d-dimensional meshes, for which there exist 2 processors in each dimension.

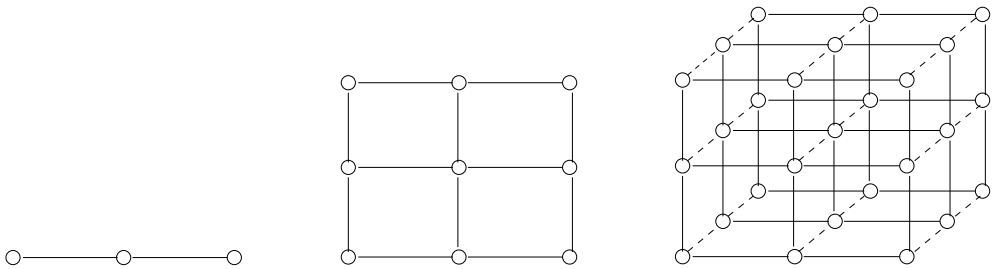

Figure 6.1: (a) 1-dimensional mesh with size 3, (b) 2-dimensional mesh with size 3×3, (c) 3-dimensional mesh with size $3 \times 3 \times 3$. The wrap around connections are not shown for simplicity.

Figure 6.1 gives example multidimensional meshes up to 3 dimensions. Note that, a 1D mesh with size n is assumed to be defined as the ring network that consists of n processors. For $d > 1$, a d-dimensional mesh of size $n_1 \times \ldots \times n_{d-1} \times n_d$ can be created by applying the following recursive steps:

- Create n_d number of $(d-1)$-dimensional meshes, each with size $n_1 \times \ldots \times n_{d-1}$.
- Group the processors that have the same coordinates in the $(d-1)$-dimensional space; and create links between the processors in the same groups such that each group forms a ring topology.

For instance, for the 3D mesh illustrated in Figure 6.1(c), three 2D meshes are created first (the solid lines indicate these meshes). Then, the processors with the same coordinates in the 2D space are connected with each other with the dashed lines. Note that the wrap around connections are not displayed in this figure for the purpose of simplicity.

Based on the steps given above to construct a multidimensional mesh, an AABC operation can be performed recursively on a d-dimensional mesh ($d > 1$) as follows:

- The dimension d of the mesh is ignored, and each processor performs AABC on the $(d-1)$-dimensional mesh that it belongs to.

- The first $(d-1)$ dimensions of the mesh are ignored, and each processor performs AABC for the consolidated message it has obtained in the previous step. Here, each AABC is on the ring that corresponds to dimension d of the mesh.

Observe that, the AABC algorithm we have described in Section 3.2 is the special case of this method for 2D meshes. In fact, this recursive AABC method corresponds to applying the AABC for ring algorithm for each dimension d_i of the mesh in order. So we can state that, the number of communications required for an AABC operation on a d-dimensional mesh of size $n_1 \times \ldots \times n_d$ is equal to $\sum_{i=1}^{d} (n_i - 1)$.

In Section 3.3, we have given a cost model for embedding communications into the AABC operations for 2D meshes. Lemma 3.2 has proposed a cost calculation method for this model. We can generalize this method for d-dimensional meshes based on the AABC operation given above.

For a given compressed hypergraph $\mathcal{H}_C = (\mathcal{V}_C, \mathcal{N}_C)$, assume that the partitions corresponding to the vertices in $\mathcal{V}_C$ are mapped to a d-dimensional mesh $(d > 1)$ such that each v_k has the coordinates $(k_1, \ldots, k_{d-1}, k_d)$. Assume further that the coordinates for the source vertex v_s of net n is given as $(s_1, \ldots, s_{d-1}, s_d)$. Then it is possible to find the cost of net n in terms of communication volume as follows:

1. Ignore dimension d (i.e. treat all partitions as if on the same $(d-1)$-dimensional space), and find the cost for the $(d-1)$-dimensional mesh.
2. For each vertex $v_k \in pins[n]$, assume that there is a source vertex at $(k_1, \ldots, k_{n-1}, s_n)$, and calculate the cost for the ring in dimension d.
3. Add all the costs in steps 1 and 2, and find the total cost for net n.

Note that, for a message originating from v_s and ending at v_k, step 1 corresponds to the route from $(s_1, \ldots, s_{d-1}, s_d)$ to $(k_1, \ldots, k_{d-1}, s_d)$, and step 2 corresponds to the route from $(k_1, \ldots, k_{d-1}, s_d)$ to $(k_1, \ldots, k_{d-1}, k_d)$.

Now that we have given methods to extend the models for 2D meshes to d-dimensional meshes, it is possible to use the embedded communication scheme for the conjugate gradient type iterative solvers on any d-dimensional mesh.

Chapter 7

Conclusions

In this thesis, we have proposed a novel communication scheme for CG type parallel iterative solvers. The purpose here was to avoid the message start-up costs due to the point-to-point communications required in SpMxV computations, through embedding them into the following reduction operations. However, the trade off here was the increase in communication volume. For this reason, we have given a cost model and a methodology to minimize this overhead. We have performed experiments using various test matrices. In the parallel system, the number of available processors was not large enough to observe the effect of our scheme on large processor meshes. For this reason, we have first shown that the cost model we have proposed was practically valid, by using the available processors. Then based on this model, we have estimated the communication costs of the conventional and the proposed schemes on different meshes.

In these experiments we have seen that, the communication costs were dominated by the message start-up costs for the matrices with small communication volume requirements. In these cases, the proposed scheme performed better than the conventional scheme, because it avoided these start-up costs. However, for the matrices with very large communication volume requirements and the meshes with small number of processors, the communication volume overhead introduced by the proposed scheme did not compensate for the decrease in the message start-up costs. On the other hand, we have observed for such matrices that, as the

number of processors increases, the communication costs for the proposed scheme tend to decrease, whereas the costs for the conventional scheme tend to increase. So we can say that, the proposed scheme performs better than the conventional one even for these matrices, if the number of processors is large enough.

It is clear that, the scheme we propose does not perform better than the conventional one in all cases (i.e. for large communication volumes and small number of processors). However, the important point here is that, it is possible to predict its performance before the CG algorithm starts, based on the given cost models and the machine specific parameters: t_s and t_w. So, for an efficient implementation of a CG type parallel iterative algorithm, the scheme to be used can be chosen according to the prediction made in the preprocessing steps. Namely, for the cases in which the message start-up costs are the dominant factors in the communication times, the proposed scheme can be used, and vice versa.

In fact, this thesis proposes an idea and then shows its practical validity for various test matrices. It is also possible to conduct more research on it to obtain better results. For example, a one phase algorithm can be developed instead of the two-phase approach for partitioning and mapping the input matrix to processors. Also, the simple heuristics for the communication scheduling problem can be replaced by more advanced algorithms. We have given a methodology to extend the models given for 2D meshes to multidimensional meshes. Based on this, another research subject might be to devise part-to-processor mapping algorithms for multidimensional meshes and compare the experimental results with 2D meshes.

Bibliography

[1] C. Aykanat, F. Özgüner, F. Ercal, and P. Sadayappan. Iterative algorithms for solution of large sparse systems of linear equations on hypercubes. *IEEE Transaction on Computers*, 37(12), 1988.

[2] C. Berge. *Graphs and Hypergraphs.* American Elsevier, 1976.

[3] T. N. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452, 1993.

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

[5] W. Camp, S. J. Plimpton, B. A. Hendrickson, and R. W. Leland. Massively parallel methods for engineering and science problems. *Communication of ACM*, 37(4):31–41, 1994.

[6] W.J. Carolan, J.E. Hill, J.L. Kennington, S. Niemi, and S.J. Wichmann. An empirical evaluation of the korbx algorithms for military airlift applications. *Operations Research*, 38(2):240–248, 1990.

[7] U. V. Çatalyürek. *Hypergraph Models for Sparse Matrix Partitioning and Reordering.* PhD thesis, Bilkent University, 1999.

[8] U. V. Çatalyürek and C. Aykanat. Hypergraph partitioning based decomposition for parallel sparse matrix vector multiplication. *IEEE Transactions on Parallel and Distributed Systems*, 10(7), 1999.

[9] IOWA Optimization Center. Linear programming problems. ftp://col.biz.uiowa.edu:pub/testprob/lp/gondzio.

[10] C.K. Cheng and Y.C. Wei. An improved two-way partitioning algorithm with stable performance. *IEEE Transactions on Computer-Aided Design*, 10(12):1502–1511, 1991.

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

[12] I.S. Duff, R. Grimes, and J. Lewis. Sparse matrix test problems. *ACM Transactions on Mathematical Software*, 15(1):1–14, 1989.

[13] T. H. Dunigan. Early experiences and performance of the intel paragon. Technical Report ORNL/TM-12194, Oak Ridge National Laboratory, 1993.

[14] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1:237–267, 1976.

[15] A. Gupta. Watson graph partitioning package. Technical Report RC 20453, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1996.

[16] B. Hendrickson and T. G. Kolda. Partitioning sparse rectangular matrices for parallel computations of ax and a^tv. In *Applied Parallel Computing in Large Scale Scientific and Industrial Problems*, 1998.

[17] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26, 2000.

[18] B. Hendrickson and T. G. Kolda. Partitioning nonsquare and nonsymmetric matrices for parallel processing. *SIAM J. Sci. Comput.*, 21:2048–2072, 2000.

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

[20] B. Hendrickson and R. Leland. *The Chaco user's guide, version 2.0.* Sandia National Laboratories, Alburquerquem NM, 87185, 1995.

[21] B. Hendrickson, R. Leland, and R.V. Driessche. Skewed graph partitioning. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing, SIAM*, 1997.

[22] M. Kaddoura, C. W. Qu, and S. Ranka. Partitioning unstructured computational graphs for nonuniform and adaptive environments. *IEEE Parallel and Distributed Technology*, 3(3):63–69, 1995.

[23] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report 98-019, Department of Computer Science, University of Minnesota, 1995.

[24] G. Karypis and V. Kumar. hmetis 1.5: A hypergraph partitioning package. Technical report, Dept. of Computer Science and Engineering, Univ. of Minnesota, 1998.

[25] G. Karypis and V. Kumar. *METIS a Software PAckage for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 3.0.* University of Minnesota, Department of Comp. Sci. and Eng., Army HPC Research Center, Minneapolis, 1998.

[26] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20, 1999.

[27] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49, 1970.

[28] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing - Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., 1993.

[29] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Willey-Teubner, Chichester, U.K, 1990.

[30] D. Luenberger. *Introduction to Linear and Nonlinear Programming.* Addison - Wesley, 1993.

[31] O. C. Martin and S. W. Otto. Partitioning of unstructured meshes for load balancing. *Concurrency: Practice and Experience*, 7(4):303–312, 1995.

[32] Matrix market. Mathematical and Computational Sciences Division, Information Technology Laboratory, National Institute of Standards and Technology, http://math.nist.gov/MatrixMarket.

[33] F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proceedings of the Scalable High Performance Comput. Conference, IEEE*, pages 486–493, 1994.

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

[35] Y.C. Wei and C.K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Transactions on Computer-Aided Design*, 10(7):911–921, 1991.