THE UNIVERSITY OF MICHIGAN

COMPUTING RESEARCH LABORATORY

MINIMIZATION OF INTERPROCESSOR COMMUNICATION FOR PARALLEL COMPUTATIONS ON AN SIMD MULTICOMPUTER

William Sai-fong Wu

CRL-TR-24-84

MARCH 1984

Room 1079, East Engineering Building
Ann Arbor, Michigan 48109
USA
Tel: (313) 763-8000

MINIMIZATION OF INTERPROCESSOR COMMUNICATION FOR PARALLEL COMPUTATIONS ON AN SIMD MULTICOMPUTER

by
William Sai-fong Wu

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
1984

Doctoral Committee:

Professor Keki B. Irani, Chairman
Professor Donald A. Calahan
Assistant Professor Janice M. Jenkins
Professor Norman R. Scott
Professor Alfred C-T Wu

ABSTRACT

MINIMIZATION OF INTERPROCESSOR COMMUNICATION
FOR PARALLEL COMPUTATIONS ON AN SIMD MULTICOMPUTER

by

William Sai-fong Wu

Chairman : Keki B. Irani

Interprocessor communication is an important aspect of parallel processing. Studies have shown that data communication can be a major cause of performance degradation of a parallel algorithm. This thesis is concerned with minimization of delay due to interprocessor data communication during the execution of a parallel algorithm on an SIMD multicomputer interconnected with any multistage interconnection network from a class of functionally equivalent networks.

The interprocessor communication cost minimization problem for the omega network in the equivalent class are presented. For the class of important permutations with which this thesis is concerned, a representation scheme is developed. This class of permutations includes the class of bit-permute-complement studied often in the literature. Based on the representation scheme and the omega network, algorithms are developed that determine, for a given parallel procedure, the mapping and remapping of data into physical memories so that the communication cost is minimized. Algorithms are also developed that determine, for every arithmetic expression of a given procedure, the alignment of operands for every binary operation so that the communication cost is minimized. For example, the communication costs of a lower triangular matrix inversion program and a fast Fourier transform program are minimized using both approaches. For each of the approaches, a saving of one-third of the original communication cost is achieved for the matrix inversion program while a saving of three-fifths

is achieved for the fast Fourier transform program. The algorithms developed are proved to be applicable for any SIMD multicomputer interconnected with any network from the equivalent class.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE

FIGURE

LIST OF TABLES

TABLE

CHAPTER 1

INTRODUCTION

Continuing advances in VLSI technology have resulted in low cost computing capabilities that make multicomputer systems more attractive as a practical way to increase effective computing power without inordinate increase in cost. Multicomputer systems consisting of more than 500 processors have been made feasible and proposed [LuBa80, Schw80]. However, unless parallel algorithms that use these multicomputer systems effectively are found, the systems will not be cost effective[Ager77].

Until a few years ago, computer scientists were primarily concerned with the computing aspects of parallel processing. Communication aspects of the processing usually receive little attention. The communication aspects are becoming at least as important as the computing aspects, if not more, as more and more proposed or built interconnection networks for the processing elements are not capable of realizing at all or realizing efficiently some of the communication patterns. Gentleman [Gent78] shows that communication time can be greater than the computation time that is based on data dependencies alone. He shows that considering data dependencies alone, the execution time for computing the matrix product of two $N \times N$ matrices is O(log$N$), but the communication delays limit the total execution time to O(N) on a mesh connected multicomputer. Possible tradeoffs between computation and communication are demonstrated through parallel algorithms developed for Boolean matrix multiplication [AgLi78]. The parallel algorithms developed are found to execute in $O(\frac{N^2}{\log N})$ and O(log N) time using N and $\frac{N^3}{\log N \log \log N}$ processors. The tradeoffs show that computation

and communication should be taken into consideration in mapping algorithms onto multicomputer systems. Indeed, by properly arranging the data onto a circularly connected network of N processors, [IrCh80] shows that the complexity of the Jacobi algorithm for a $N \times N$ matrix can be reduced from $O(N^2)$ to $O(N)$. This thesis investigates the problem of mapping parallel algorithms efficiently onto a multicomputer system interconnected with any multistage interconnection network from a class of functionally equivalent networks.

The structure of the multicomputer system that this proposal is based on is an SIMD (Single Data Multiple Data Stream [Flyn72]) computer as shown in figure 1.1. It consists of a central processor with its own private memory, $N = 2^n$ processing elements each with its own private memory, and an interconnection network that provides data paths among the processing elements (PEs). Data communication between PEs is performed by executing routing instructions. Given a data communication pattern, depending on the interconnection network, one or more routing instructions for the network may be needed to realize the communication pattern. The total execution time of a parallel algorithm thus consists of the computation time and the communication time. This thesis is concerned with efficient execution of an arbitrarily given parallel algorithm on a multicomputer system through minimization of the communication delays of the algorithm.

1.1. Motivation and Objectives

The execution of a parallel algorithm can be considered to proceed in a sequence of alternating computation and communication stages. During a computation stage, active processors perform transformations on available data, generating new results. During the communication stage, one or more data are transferred between processors in preparation for the next computation stage.

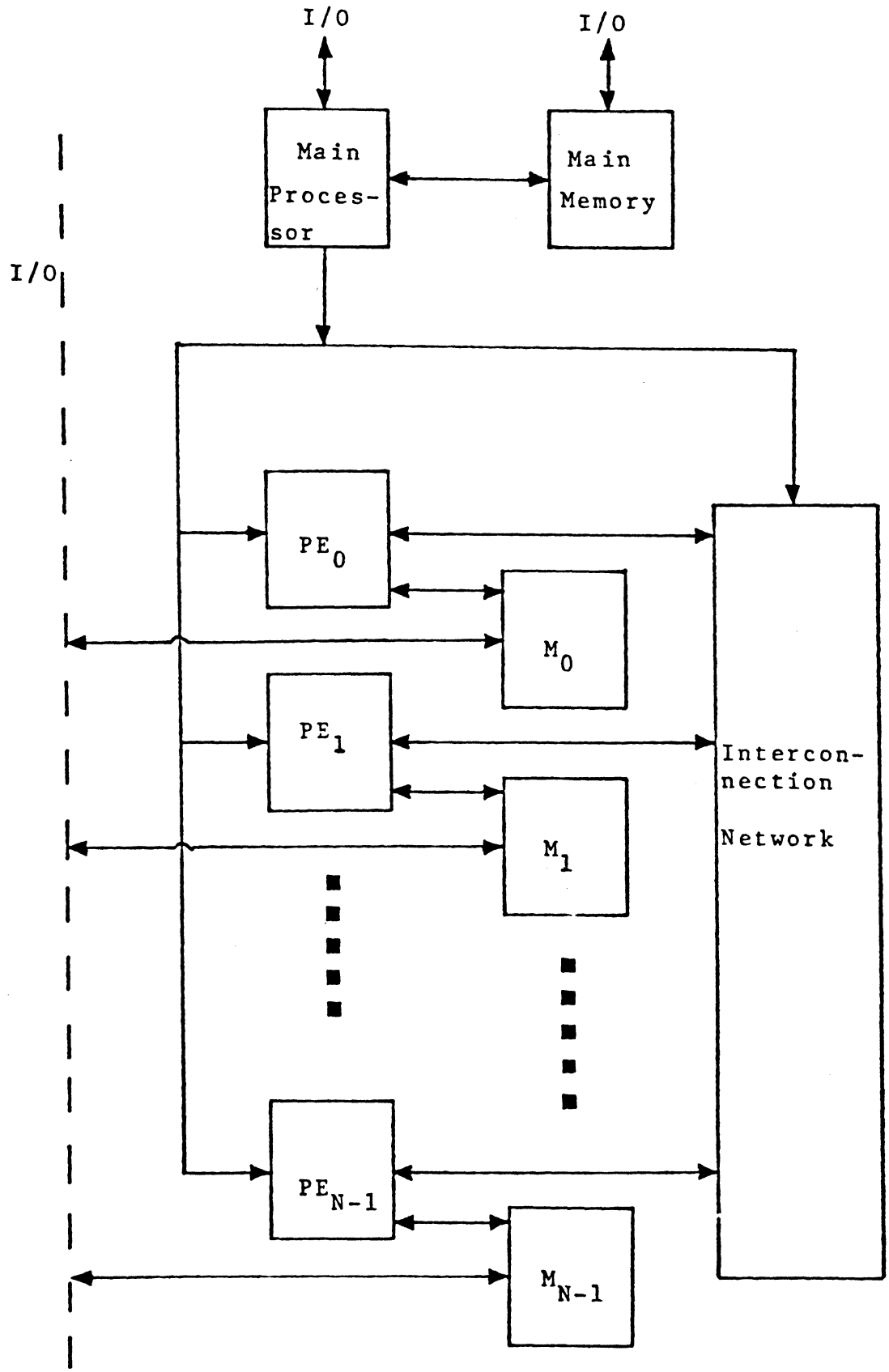


Figure 1.1 Model of an SIMD multicomputer.

As Chen [Chen81] has pointed out, while the processing speed of a computation stage can readily be determined from the capability of the processing elements, the processing speed of a communication stage depends on the organization of the entire system, and thus is much more complicated. The communication process is dependent on many factors, including the interconnection network, the storage scheme for the data and the data transfer functions performed. For instance, consider a parallel assignment statement (PAS).

$$A(i) \leftarrow D(i+2) + A(i) * (B(i) - C(i)), \qquad 0 \leq i \leq N-1.$$

It consists of the following cycles :

1.Computation stage :

$T(i) \leftarrow A(i) * (B(i) - C(i))$ (T is a temporary vector)

Communication stage :

align $T(i)$ and $D(i+2)$

2.Computation stage :

$A(i) \leftarrow D(i+2) + T(i)$

Clearly, alignment of vectors D and T can be achieved in different ways, each causing a different communication delay (number of routing steps). For example, we can move D to T or move T to D. The way the vectors are stored also affects the data transfer function. Communication delay of a data transfer function may again vary, depending on the type of network used. Thus, for a given algorithm and a given machine, a careful study of the relationship among the computational structure, the communication requirements and the machine structure is necessary for minimizing the total communication cost [Ager77].

At the hardware level, a cross-point array interconnection network is obviously the best choice for minimizing the communication costs. However, as N increases, its cost-effectiveness decreases rapidly and becomes impractical[Kuck78]. It is also possible to design

an interconnection network that matches the communication requirement of a given algorithm. However, it is often the case that it is too costly or its usefulness is limited to only specific problems.

The requirements of cost-effectiveness along with high performance have led to proposals of many blocking multistage interconnection networks [Park80, WuFe78, WuFe79, Prad79]. Such network of size N (N input $\times$ N output) consists of log N stages each comprising $\frac{N}{2}$ elementary 2 input $\times$ 2 output, two-state switches. Each stage is preceded or followed by a fixed wiring pattern that connects it to the adjacent stage or the outside. The number of admissible permutations on such network is $\sqrt{N^N}$, which is a small fraction of $N!$ possible permutations for $N>8$. However, researchers [Lawr75,Peas77] have demonstrated that many important permutations of parallel algorithms can be realized in one pass through some of these networks. In this thesis, we will be concerned with a class of networks that are functionally equivalent to the omega network. We will denote this class of networks by Γ and will define it in chapter 5.

Once, a good interconnection network is chosen, other means must be sought to improve the communication delays of parallel algorithms. Now, the need of data communication arises when the operands of some binary operations are not aligned. It is thus clear that communication cost is affected by the manner in which data are stored in the memories. This has led to the development of software techniques [IrCh82, KuSt77, Kuhn79, Kuhn80] to properly distribute the data over memories to reduce communication costs. Kung & Stevenson [KuSt77], Kuhn [Kuhn80], Bohkari [Bohk81] and Irani & Chen [IrCh82] have demonstrated that to improve the performance of parallel algorithms, storing the data properly is sometimes a more flexible tool for providing better solutions than hardware, whose capability is often limited by cost and complexity. However, most of their works are limited to multicomputer systems with single stage interconnection networks of limited capabilities, for example, the mesh connected [KuSt77], the perfect shuffle connected [Kuhn80] and the circularly connected

multicomputer systems[IrCh82].

The global objective of this thesis is thus to develop methodologies for minimizing interprocessor communication delays in parallel computations on an SIMD multicomputer interconnected with a blocking multistage interconnection network in Γ. It embodies the following general objectives:

1. Characterization of a network in Γ.

2. Characterization of the class of communication patterns with which we will be concerned.

3. Development of methodologies that will analyze an arbitrarily given algorithm and obtain an optimal or near optimal solution to the interprocessor communication minimization problem for an SIMD multicomputer interconnected with the network.

4. Demonstration of how the methods developed can be used for any multistage interconnection network in Γ.

The communication problem with which this thesis is concerned is first formulated in [IrCh82] and is defined formally in the following section.

1.2. The Communication Cost Minimization Problem

The communication cost of a parallel algorithm is dependent on the algorithm and the multicomputer system onto which it is to be mapped. Characterization of the communication cost minimization problem hence involves specifications of parallel algorithms, interconnection networks, and the relationship between the two. The notion of data transfer at the logical and the physical level is illustrated.

Specification of Parallel Algorithms

Let m denote the logical memory and assume that the parallel algorithm uses $2^n = N$ logical memories, $m_0, m_1, m_2, \ldots, m_{N-1}$. Also assume that the vectors in the algorithm are of size N and their elements are indexed from 0 to $N-1$, and that the ith element of a vector, for example, B, i.e., $B(i)$, is stored in logical memory m_i.

The communication requirement of a parallel algorithm is specified by a sequence of k logical data transfers, P_j, $0 \leq j \leq k-1$. Let $\{0,1,\ldots,N-1\}_m$ denote the index set of the logical memory, then a logical transfer (or data transfer at the algorithm level), P_j, for a particular data is a partial function mapping the index set $\{0,1,\ldots,N-1\}_m$ into itself. That is, P_j sends data from memories m_i to $m_{P_j(i)}$, for all i to which P_j assigns values. For instance, consider the expression $u(i-1) \times v(i+1)$. Since $u(i-1)$ and $v(i+1)$ are not stored in the same logical memory, logical transfers are needed to align the two vector operands before multiplication can be executed. We can move either u to v or v to u. Suppose v is to be moved to u for alignment. That is, transfer operation is applied to variable v only, so that after the operation, the desired computation can be carried out at u. The logical transfer P_j for vector v may then be written as a permutation,

$$P_j = \begin{pmatrix} 0 & 1 & 2 & \ldots & N-1 \\ N-2 & N-1 & 0 & \ldots & N-3 \end{pmatrix}.$$

This transfer P_j is, at the logical level, a uniform shift of distance -2, i.e., $P_j(i) = (i-2) \bmod N$. However, P_j is not unique. A logical transfer for vector u with shift distance +2, or a logical transfer for v with shift distance -1, and one for u with shift distance +1 can also be used to align the operands.

The notion of where the operands shall be aligned can be formalized as follows. Consider the expression $u(P_u^{-1}(i)) \times v(P_v^{-1}(i))$, instead of aligning u with v or v with u, we can align them according to a bijection mapping $\{0,1,\ldots,N-1\}_m$ onto $\{0,1,\ldots,N-1\}_m$. By this, we

mean that if w is the result vector of the binary operation on u and v, and A_w is the bijection, then the binary operation for u and v is performed in processor $A_w(i)$, and the result is stored in memory $m_{A_w(i)}$, $0 \leq i \leq N\text{-}1$. The logical transfers for vectors u and v, in this case, are $P_{uw} = A_w P_u$ and $P_{vw} = A_w P_v$, respectively. For instance, consider the binary operation $u(i-1) \times v(i+1)$ again. If $A_w =$ identity function, then $P_{uw}(i) = i + 1 \ mod\,(N) = P_u$ and $P_{vw}(i) = i - 1 \ mod\,(N) = P_v$. If $A_w(i) = h + i \ mod\,(N)$, then $P_{uw}(i) = i + 1 + h \ mod\,(N)$ and $P_{vw}(i) = i - 1 + h \ mod\,(N)$. Henceforth, such bijections, A_w's will be called alignment functions. If temporary vector variables w_i is to be aligned to w_j, the logical transfer is given by $A_{w_j} A_{w_i}^{-1}$.

Specification of Interconnection Networks

Let M denote the physical memory with N memory modules, $M_0, M_1, \ldots, M_{N-1}$, which are connected by an interconnection network. The network is defined by a set of admissible network functions, $\underline{Q} = \{Q_0, Q_1, \ldots, Q_k\}$, each element of the set being a bijection on the index set $\{\ 0,1,2,\ldots,N\text{-}1\ \}_M$. When a routing instruction associated with an admissible network function Q_j, is executed, data is sent from data memory modules M_i to $M_{Q_j(i)}$ for all the "active" memories M_i through direct data paths of the network. In other words, each admissible network function designates a transfer that can be physically realized in one routing step (one pass through the network).

Let P be an arbitrary partial function on the the index set $\{\ 0,1,2,\ldots,N\text{-}1\ \}_M$ and that i specifies a transfer of some data from M_i to $M_{P(i)}$. Zero or more routing instructions will be needed to realize P, i.e. zero or more passes through the network before the data can reach the destination memories. Hence, we can associate with each interconnection network, a distance function D on a set of partial functions such that

$$D(P) = \begin{cases} \textit{minimum number of routing steps to realize } P, \textit{ if } P \textit{ can be realized.} \\ \infty \quad \textit{if } P \textit{ cannot be realized.} \end{cases}$$

Clearly, if P is an identity function, no transfer is necessary. Therefore, $D(P) = 0$. If P is an element of the set of admissible network functions, Q, of a network, then $D(P) = 1$ with respect to the network. If $D(P) = 1$ for a network, then we define P as an admissible permutation of the network.

Mapping Data onto Memories

Consider a parallel algorithm using N logical memories that is to be executed on a machine with N PEs and an interconnection network with a distance function D. To execute the algorithm, we map the "logical" memories, m, onto the "physical" memories, M. A data mapping function for a data variable v is a bijection F_v mapping $\{ 0,1,...,N-1 \}_m$ onto $\{0,1,...,N-1\}_M$, i.e., the ith component of v in m_i is stored in memory module $M_{F_v(i)}$, $0 \leq i \leq N-1$.

Consider two vectors u and v, let F_u and F_v be their corresponding data mapping functions. Suppose u and v have to be aligned for certain operation, for example, $u(P_u^{-1}(i)) \times v(P_v^{-1}(i))$. One way to align the two vector operands is to move u to v so that after the operation the desired computation can be carried out at v. Let $P_{uv} = P_v^{-1}P_u$ be the function describing the logical transfer (permutation) of moving u to v. At the logical level, element $u(F_u^{-1}(i))$ is moved from $m_{F_u^{-1}(i)}$ to $m_{P_{uv}F_u^{-1}(i)}$. At the physical level, however, the data movement is from M_i to $M_{F_vP_{uv}F_u^{-1}(i)}$. The actual permutation to be realized by the network is then given by $F_vP_{uv}F_u^{-1}$, and the communication cost is thus given by $D(F_vP_{uv}F_u^{-1})$. Figure 1.2 illustrates the notions of logical transfer, physical transfer, and data mapping function. Identity functions are usually the first choice for data mapping functions. However, if the data mapping functions F_u and F_v are both identity functions, the network transformation to be realized becomes P_{uv} = transfer at the algorithm or logical level, which may not be realizable in one routing step. Hence, alternate data mapping functions for u or v or both may have to be used to make $F_vP_{uv}F_u^{-1}$ admissible.

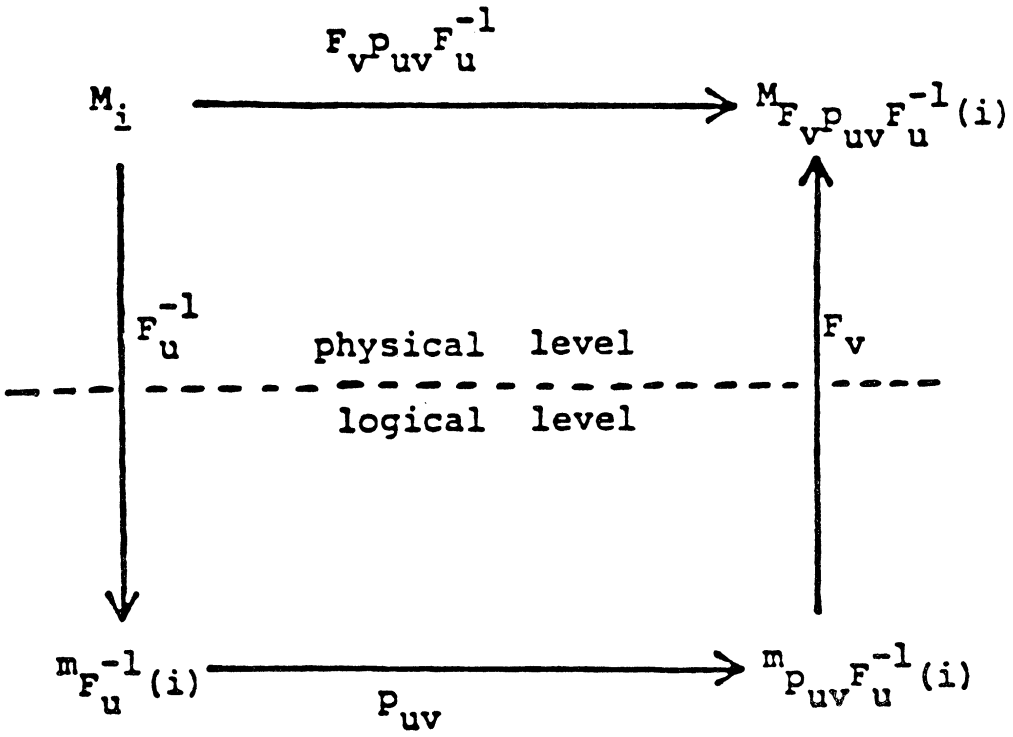


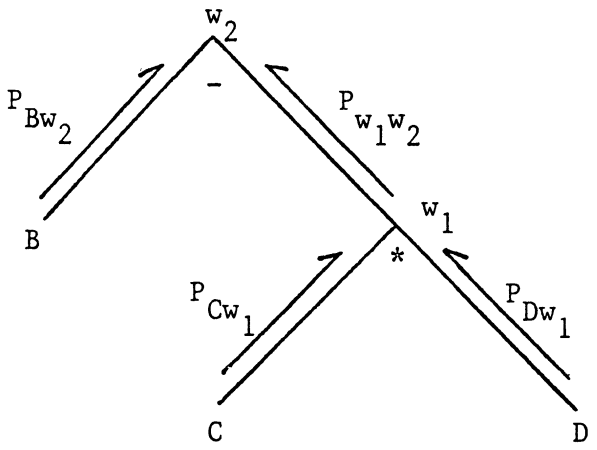
Figure 1.2 Logical and physical data transfers

Having specified the algorithm, the network and the relationship between data transfer and data mapping functions, we shall now formulate the interprocessor communication minimization problem.

The Minimization Problem

For any binary operation, the way operands are aligned determines the logical transfer functions for the two operands. Instead of aligning variable u with v or v with u for a binary operation involving v and u (as was the case in the previous section), we can perform the binary operation in processor $A_w(i)$ and store the result in memory $m_{A_w(i)}$, $0 \leq i \leq N-1$. There is no loss of generality if the resulting vector is called $w(A_w(i))$. In other words, the data mapping function F_w is an identity function. The total communication cost for the binary operation is then $D(P_{uw}F_u^{-1})+D(P_{vw}F_v^{-1})$, where the logical transfers P_{uw} and P_{vw} are A_wP_u and A_wP_v, respectively.

The above result can be generalized to a parallel expression, S, with variables $v_1, v_2, \ldots, v_k$. Let E be an expression tree for S. For example, for $B(i+1) - C(i-3)*D(i+1)$, E is as shown in figure 1.3.



where w_1 and w_2 are the interna
nodes of the tree and the partia results.

Figure 1.3 An expression tree for $B(i+1)-C(i-3)*D(i+1)$.

Now, let $w_0, w_1, \ldots, w_t$ be the internal nodes of E, where each w_i represents the partial result of some binary operation on variables and/or other internal nodes. For a given expression tree E, which specifies the order of computation of S, the alignment of operands for every operation from which the logical transfer functions can be specified must be determined. Thus, for a statement S, the communication cost is

$$\sum_{\substack{v_i: \text{ variable (leaf node)} \\ w_j: \text{ partial result (internal node)} \\ w_j = \text{parent}(v_i)}} D(P_{v_i w_j} F_{v_i}^{-1}) \; + \sum_{\substack{w_i, w_j: \text{ internal nodes} \\ w_j = \text{parent}(w_i)}} D(P_{w_i w_j}),$$

where $parent(x)$ denotes the parent node of x in E.

The above indicates that the total communication cost depends on the data mapping functions F_{v_i}'s, and the transfer functions $P_{v_i w_j}$'s and $P_{w_i w_j}$'s. The transfer functions in turn depend on the expression tree E, as well as the alignment of operands. If the operations are commutative, the tree may not be unique too.

The complete communication cost minimization problem for a parallel algorithm can hence be stated as follows :

Given a parallel algorithm involving variables with logical index set $\{0,1,...,N-1\}_m$, an interconnection network for physical memories with index set $\{0,1,...,N-1\}_M$ and the distance function D, determine the following :

1. an expression tree for every parallel assignment statement in the given algorithm;

2. alignments of operands for binary operations in every expression tree in 1; and

3. a data mapping function F_v for every variable v involved in the parallel statement in 1,

such that the value of the expression

$$\sum_{\substack{\text{statements of the given algorithm}}} \sum_{\substack{v_i : \text{variable (leaf node)} \\ w_j : \text{partial result (internal node)} \\ w_j = \text{parent}(v_i)}} D(P_{v_i w_j} F_{v_i}^{-1}) + \sum_{\substack{w_i, w_j : \text{internal nodes} \\ w_j = \text{parent}(w_i)}} D(P_{w_i w_j}) \qquad (1.1)$$

is minimized.

A data mapping is defined to be static if the data mapping function, F_{v_i} for each variable, v_i, in the algorithm does not change throughout the algorithm.

[Kuhn80] and [IrCh82] have shown that by determining when and how to perform a remapping during the execution of a parallel algorithm, the new cost, the communication cost plus the cost of remapping itself, can be less than the cost of using only static data mapping. For nonstatic mapping, we will determine a data mapping function for each logical transfer of every vector variable in the algorithm. That is, given a sequence of k logical transfers for v, we will determine a sequence of data mapping functions, $\hat{S}_v = F_{v,1}, F_{v,2}, \ldots, F_{v,k}$, where $F_{v,i}$ is the data mapping function for the ith logical transfer of vector v. The remapping cost for v is then given by

$$\sum_{i=1}^{k} D(F_{v,i+1}F_{v,i}^{-1}).$$

For nonstatic mapping, the sum of remapping cost for all the vectors is to be included into the above communication cost formulation (equation 1.1). Only step 3 of the above cost minimization problem has to be changed. That is, instead of determining a data mapping function F_v for every variable v, we determine a sequence of data mapping functions $F_{v,1}, F_{v,2}, \ldots, F_{v,k}$ for v.

Clearly, the total communication cost is dependent on the alignment functions, the mapping and remapping functions, and the logical transfer functions. Furthermore, since the expression tree for a statement may not be unique, the choice of expression trees may affect the communication cost. The general communication cost minimization problem is complex In fact, even the static data mapping problem, where alignment functions and expression trees are fixed, can be shown computationally equivalent to the bandwidth reduction problem [KuSt77,Bokh81] which is NP-complete. Hence, in this thesis, we shall restrict ourselves to a specific algorithm environment. We shall assume that the expression trees are fixed. This is often desirable, for example, in problems where computation reordering may cause loss of sig-

nificant digits, overflow or underflow, etc.. We shall also assume that the amount of computations in a parallel program is fixed. For example, if there is a loop in the program, the number of times the loop will be repeated is fixed and known during compile time. Assuming all alignment functions are identity functions, we will develop algorithms for optimum or near-optimum nonstatic data mapping functions. Assuming all data mapping functions are known or fixed, we will also develop algorithms for finding optimum or near-optimum alignment functions.

1.3. Review of Previous Works

Data mapping is concerned with the manner in which data are stored. Its objective is to reduce communication costs between processors and processors or processors and memories during the execution of parallel algorithms. Even though a restricted version of the data mapping problem called mapping problem was first formulated by Kung and Stevenson [KuSt77] and the general communication cost minimization problem was then formulated by Irani and Chen [IrCh82],the notion of data mapping functions for special cases can actually be traced back to earlier literature. For example, in [Lawr75], [Budn71] and [Swan74], skewed array storage schemes for conflict-free access to rows, columns, forward and backward diagonals and square blocks are described. In [Peas77], data mapping functions for performing FFT and matrix multiplication efficiently on an indirect binary n-cube microprocessor array are described. However, no general technique had been reported.

In formulating the mapping problem, Kung and Stevenson [KuSt77] model the communication requirement of a parallel program as a sequence of k logical transfers, $P_j, 0 \leq j \leq k-1$. The minimization problem is then to find a single mapping function, F, for all the vector variables in the parallel program such that

$$\sum_{j=0}^{k-1} D(FP_jF^{-1}) \text{ is minimized.}$$

For $P_0 = P_1 = \cdots = P_{k-1} = P$, they have shown that this type of algorithm (type I) can always be executed optimally, even on the simplest network, namely, the circularly connected network. Type I algorithm can be executed in at most $4k$ routing steps on a circularly connected network. Matrix transposition and polynomial evaluation as described in [Ston71] are examples of such an algorithm. It is also pointed out in [KuSt77] that the mapping problem for this type of algorithms corresponds to a graph numbering problem.

For algorithms using different transfers (type II), Kung and Stevenson [KuSt77] pointed out that the minimization problems become the well known bandwidth or profile reduction problems encountered in sparse matrix computations. Since no exact solution for the above general problems can be easily obtained, in [ThKu77] and [KuSt77], an approximate technique for the bitonic sort problem (type II algorithm) on a mesh connected network is described. The technique determines an optimum data mapping function for the most expensive transfer. The approach, however, is not desirable. The optimum data mapping function for a particular transfer may not be optimum for the entire algorithm. It may make the total cost higher as the optimal data mapping function may be costly for other transfers. Nevertheless, the routing time is reduced from $O(\sqrt{n} \log n)$ to $O(\sqrt{n})$.

In [Bokh81] the mapping problem is shown to be computationally equivalent to the graph isomorphism problem, and similar to the quadratic assignment problem. Since they are usually solved approximately by heuristic algorithms, Bokhari [Bokh81] describes a heuristic algorithm in solving the mapping problem that arises when solving structural problems on the finite element machine (FEM), an array of processors under development at NASA Langley Research Center.

In [Kuhn79, Kuhn80] efficient mapping of several numerical algorithms (type II algorithms) on shuffle-exchange network is described. Efficient mapping in his sense, is to make

the communication complexity of an algorithm to be equal to or less than the computational complexity of the algorithm. The notion of nonstatic mapping is introduced to map parallel algorithms that cannot be efficiently mapped by static mappings. A mapping is defined to be static if the mapping of data to memory does not change throughout the algorithm. Hence, instead of minimizing

$$\sum_{j=0}^{k-1} D(FP_jF^{-1}),$$

as in [KuSt77], Kuhn [Kuhn80] tries to minimize

$$\sum_{j=0}^{k-1} D(F_{j+1}P_jF_j^{-1}).$$

Non-static mapping is shown to be useful for some numerical algorithms considered by Kuhn. However, no systematic method for finding the mapping functions, F_j, $0 \leq j \leq k-1$, is described.

In [IrCh82], the general problem of communication cost minimization (as described in section 1.2) is formulated. Based on the cycle structures of an interconnection network and two arbitrarily given permutations, a mapping methodology for the permutations is described.

A heuristic algorithm to obtain an optimal or near-optimal solution for the problem of mapping a particular class of algorithms onto a circularly connected network is also described in [IrCh82]. The only restriction to the algorithms considered is that it consists of uniform-shift type logical transfers only. Their approach consists of the following steps. First, at the logical level, an optimal sequence L of logical transfers is obtained (by determining alignment functions and a computation ordering for each expression tree), assuming that identity mapping function is used. Then, based on L, they determine for every data v a data mapping function F_v. Since F_v is optimal only with respect to L and may not be optimal in a global sense, a heuristic that makes appropriate adjustments so that the resulting data mapping functions become optimal or near-optimal is then used. The notion of nonstatic data map-

ping is also considered and a method for finding an optimal remapping schedule is developed. However, it is optimum with respect to a particular L only.

Previous efforts regarding to the interprocessor communication problems have primarily been concerned with solutions to special cases or on fast heuristics. The most general cases that have been studied so far are reported in [IrCh82]. However, the study is mainly for a circularly connected network that is considerably different and less powerful than the class of multistage interconnection networks that is to be studied here.

1.4. Outline of Thesis

In chapter 2, a network in Γ, the omega network, on which chapters 3 and 4 are based, is characterized. Based on the characterization, the class of important permutations that this thesis assumes is described. The characterization also leads to a method for solving for a data mapping function for a single vector variable. The method is demonstrated by finding a data mapping function that enables bit reversal permutation as well as perfect shuffle permutation admissible on an omega network. In chapter 3, permutations and data mapping functions are characterized. Identity alignment functions are assumed and algorithms are developed that determine, for a given parallel procedure, the mapping and remapping of data into physical memories so that the total communication cost is minimized. In chapter 4, permutations and alignment functions are characterized. Identity data mapping functions are assumed and algorithms are developed that determined, the alignment of operands for every binary operation of expression trees so that the total communication cost is minimized. In chapter 5, Γ, the class of multistage interconnection networks that are functionally equivalent to the omega network is defined. The algorithms developed in chapters 3 and 4 are then proved applicable for any network in Γ.

CHAPTER 2

THE OMEGA NETWORK AND THE ALGORITHM ENVIROMENT

In this chapter, the set of admissible network functions, defining the omega network in Γ is described. A representation scheme for a class of communication patterns is developed. Based on the representation scheme, a general method for finding data mapping function for a class of simple problems is described. The method is demonstrated by finding a data mapping function for bit reversal permutation (BRP) and perfect shuffle permutation (PSP) so that they can both be realizable in one pass through an omega network. The data mapping function is proved to be good for omega network of any size, and thus proving Chen's[Chen82] conjecture that for BRP and PSP, such data mapping function can always be obtained for omega network of any size.

The class of permutations that the method can apply to is limited by the representation scheme. It includes one of the five families of "Frequently Used Bijections" collected in [Lenf78] only. However, it will be shown that the representation scheme can be extended to represent a larger class of permutations that we will be designating by Δ[IrWu82]. This larger class of permutations is assumed throughout this thesis. By this, we mean that all logical transfers for the vector variables of parallel algorithms are assumed to be in Δ. If a parallel algorithm uses other permutations besides logical transfers in Δ, we shall consider the minimization problem for vector variables involved in logical transfers in Δ only.

2.1. The Omega Network

Since the omega network proposed by Lawrie[Lawr75] has been extensively studied in the past[Sieg78,WuFe79], we will give a brief description of the network and then characterize Ω, the set of admissible network functions defining the omega network.

The topology of a $2^n \times 2^n$ omega network consists of $N = 2^n$ input lines, N output lines and n stages. The network input lines and network output lines are both numbered in binary notation from 0 to N–1 as shown in figure 2.1.

The stages are also numbered from 0 to n–1, from right to left, as shown in the figure. Each of these stages consists of a perfect shuffle interconnection of N lines and $\frac{N}{2}$ switches. The switches of each stage are numbered in binary notation from 0 to $\frac{N}{2}$–1 as in figure 2.1.

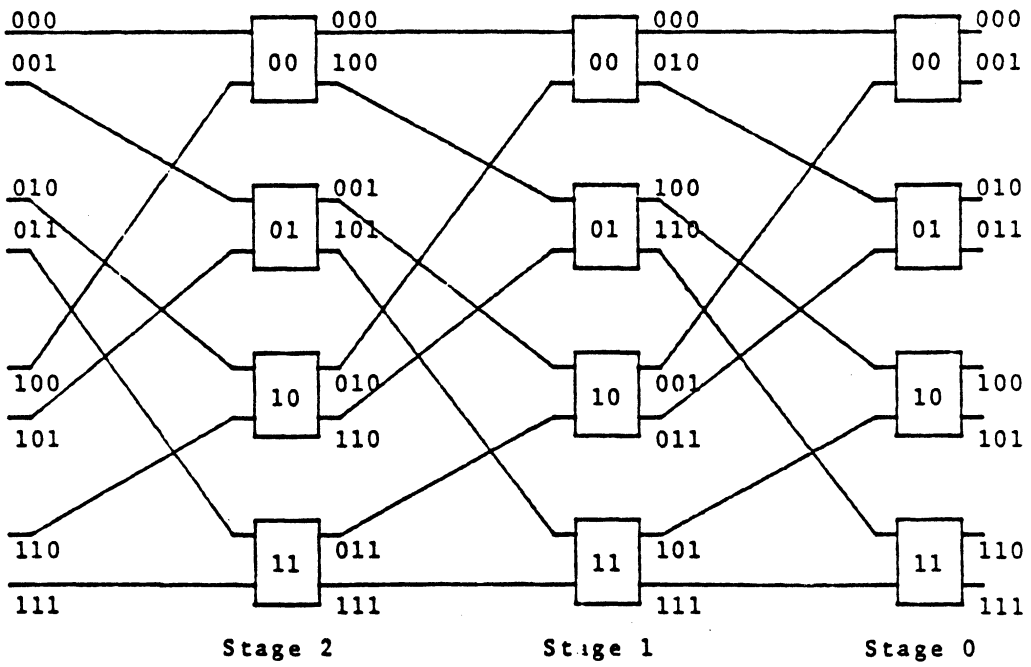


Figure 2.1 A $2^3 \times 2^3$ omega network.

For our purpose, a switch is not capable of broadcasting. It may either send its inputs straight through or interchange the inputs i.e. a switch may operate in one of two modes namely, "direct" or "crossed".

The input lines of a stage are called switch input lines and the output lines of a stage are called switch output lines (figure 2.2). The indexing of all these lines is carried through the network, assuming all switches operate in the direct mode. That is, if lines (network input lines or switch input lines) i and j are paired together on a switch on the left, the lines (network output lines or switch output lines) incident on the right are also indexed as i and j, with i connected to i and j to j when the switch operates in the direct mode. The switch output line k of stage i is connected to switch input line k of stage i-1, $1 \leq i \leq n-1, 0 \leq k \leq N-1$. The network input lines are the same as the switch input lines at stage $n-1$. The network output lines are the same as the switch output lines at stage 0.

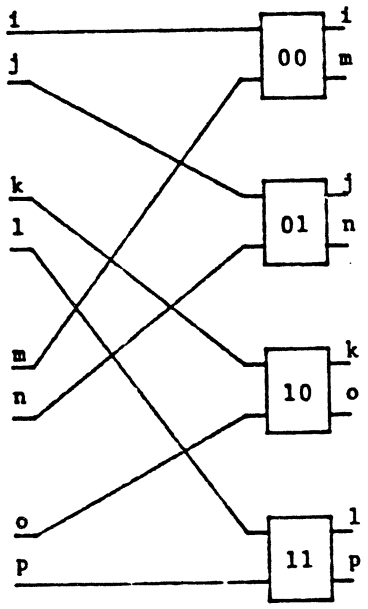


Figure 2.2 A single stage for a $2^3 \times 2^3$ omega network.

Let l be the index of a switch input line and let it be identified by an n-dimensional vector $(l_{n-1},l_{n-2}, \ldots ,l_1,l_0)$, where

$$l=2^{n-1}l_{n-1} + 2^{n-2}l_{n-2}+.....2l_1 + l_0.$$

At stage i, the switch input lines that differ only in their components l_i are paired together as inputs to a switch, for $0 \leq i \leq n-1$. Two switch input lines $(l_{n-1}, \ldots ,l_{i+1},1,l_{i-1},...,l_0)$ and $(l_{n-1}, \ldots ,l_{i+1},0,l_{i-1},...,l_0)$ connected to a switch at stage i can be connected to switch output lines $(l_{n-1}, \ldots ,l_{i+1},0,l_{i-1}, \ldots ,l_1,l_0)$ and $(l_{n-1}, \ldots ,l_{i+1},1,l_{i-1}, \ldots ,l_1,l_0)$, respectively, at the output if the switch operates in the crossed mode. They will be connected to switch output lines $(l_{n-1}, \ldots ,l_{i+1},1,l_{i-1}, \ldots ,l_1,l_0)$ and $(l_{n-1}, \ldots ,l_{i+1},0,l_{i-1}, \ldots ,l_1,l_0)$, respectively, at the output if the switch is set to the direct mode. Since there are n stages, each component l_i of all network input lines, $0 \leq i \leq n-1$, is affected at some point and can be changed by setting appropriate switches to the exchange mode. Hence, the mapping of a network input line $l1$ to a network output line $l2$ can be regarded as an n-step process in which the ith component of the network input line $l1$ is changed to the ith component of the network output line $l2$ at step (or stage) i, for $0 \leq i \leq n-1$.

Let x & y be the indices of a network input and a network output,respectively. A permutation, P, is then a one-to-one function mapping x into y for all integers in the range $0 \leq x \leq 2^n-1$ and $0 \leq y \leq 2^n-1$, i.e. $y=P(x)$. Let $x_{n-1}x_{n-2} \cdots x_0$ and $y_{n-1}y_{n-2} \cdots y_0$ denote the binary representations of x & $y \in \{0,1,...,2^{n-1}\}$, respectively. Then the function representing the permutation, P, can be described by a set of functions $p_i, 0 \leq i \leq n-1$, where

$$y_i = p_i(x_{n-1},x_{n-2},...,x_0).....(2.1).$$

If P is a mapping that is linear in the components of the index, then we can write

$$\underline{y} = \mathbf{P}\underline{x},$$

where $\underline{x} = \left[x_{n-1}, x_{n-2}, x_{n-3}, \ldots, x_0\right]^T$,

$\underline{y} = \left[y_{n-1}, y_{n-2}, y_{n-3}, \ldots, y_0\right]^T$,

$\mathbf{P}$ is a $n \times n$ nonsingular binary matrix,

and modulo 2 arithmetic is used.

In what follows, a permutation will be denoted by a capital letter. If a permutation that is denoted by a capital letter has a nonsingular binary matrix representation, then the matrix will be denoted by the corresponding capital letter in boldface. Unless otherwise stated, modulo 2 arithmetic will be used for all matrix operations. We will use $\mathbf{L}$ to denote an arbitrary lower unit triangular matrix and $\mathbf{U}$ to denote an arbitrary upper unit triangular matrix. By unit matrix, we mean a matrix in which all the coefficients on the main diagonal are ones. The following theorem is due to Parker[Park80] and is based in Pease's earlier work[Peas77].

Theorem 2.1

A permutation P is an element of Ω, iff for every i, $0 \leq i \leq n-1$, y_i can be expressed as

$$y_i = x_i \oplus f_i\,(y_{n-1}, y_{n-2}, \ldots, y_{i+1}, x_{i-1}, x_{i-2}, \ldots, x_0).$$

where f_i is a boolean function and $\oplus$ stands for exclusive-or operation (equivalent to modulo 2 arithmetic).

Proof :

Consider a switch node at stage i of the omega network. This node switches a pair of lines whose indices differ only in the ith components of their binary representations. The less

significant components of the index have not been modified by any switch node at stage $k, k<i$. Hence f_i maps the less significant components of x, i.e., $x_{i-1}, \ldots, x_0$ and the more significant components of y, i.e. $y_{n-1}, y_{n-2}, \ldots, y_{i+1}$, into $\{0,1\}$.

The fact that y_i must be equal to $x_i \oplus f_i$ instead of the more general form $(g_i x_i \oplus f_i)$, reflects the fact that if one input line to a switch node is switched, the other is also switched.

Corollary 2.1

A permutation P, that is linear in the components of the indices, is in Ω iff it can be expressed in the form

$$\mathbf{L}\underline{y} = \mathbf{U}\underline{x}.$$

□

Since the inverse of a lower unit triangular matrix is also a lower unit triangular matrix, for the linear case, $\underline{y}$ can be written as :

$$\underline{y} = \mathbf{LU}\underline{x}.$$

Hence, a linear mapping P with permutation matrix $\mathbf{P}$ is admissible through a $2^n \times 2^n$ omega network iff it can be decomposed into an $\mathbf{L}$ and a $\mathbf{U}$ such that $\mathbf{P} = \mathbf{LU}$.

2.2. A Method for Finding Data Mapping Functions

In what follows, we will limit ourselves to data mapping functions and permutations that are linear in the components of the indices. That is, we will consider permutations and data mapping functions that can be represented in $n \times n$ nonsingular binary matrix form only.

Consider a data mapping function F_v for a vector variable v. The relationship between logical and physical memory is as follows :

$$\underline{x} = \mathbf{F}_v \underline{x}_L,$$

where $\underline{x}_L$ is an n-dimensional vector that is the binary representation of the index of the logical memories of v,

$\underline{x}$ is an n-dimensional vector that is the binary representation of the index of the physical memories, where components of v are respectively stored.

A logical transfer P_j for v can be expressed by a matrix, $\mathbf{P}_j$, such that

$$\underline{y}_L = \mathbf{P}_j \underline{x}_L \text{.....(2.2)},$$

where $\underline{y}_L$ is an n-dimensional vector that is the binary representation of the index of the logical memories of u, where v will be stored after the permutation.

Assuming identity data mapping function for u,

$$\underline{y} = \underline{y}_L,$$

where $\underline{y}$ is an n-dimensional vector that is the binary representation of the indices of the physical memories, where components of u are respectively stored.

The physical transfer relating the physical memories where v and w are stored is given by

$$\underline{y} = \mathbf{P}_j [\mathbf{F}_v]^{-1} \underline{x}.$$

If $\mathbf{P}_j[\mathbf{F}_v]^{-1}$ is **LU** decomposable, then with data mapping function, F_v, the logical transfer P_j is admissible through an omega network with the data mapping function, $\mathbf{F}_v$. Given k permutations, P_j, $0 \leq j \leq k$, for a vector v, we will consider, in the next section, the problem of how to find a data mapping function, F_v, if there exists one, for P_j, $0 \leq j \leq k$, such that $\mathbf{P}_j \mathbf{F}_v^{-1}$, $0 \leq j \leq k$, are **LU** decomposable. The method will be described by finding a data mapping function, $\mathbf{F}_v$, that makes PSP and BRP both admissible to an omega network.

2.2.1. Data Mapping for PSP and BRP

Perfect shuffle permutation (PSP) and bit reversal permutation (BRP) are two important data communication patterns in many numerical problems, e.g., Fast Fourier Transform. However, although perfect shuffle interconnection is used in every stage of an omega network, both PSP and BRP are not omega network admissible. In what follows, it will be shown that a systematic method can be used to find a data mapping function such that PSP (P_s) and BRP (P_b) for vector variable v are simultaneously admissible in one pass through an omega network. In fact, a general solution that is good for omega network of any size is found. Thus proving the conjecture[Chen81] that for permutations BRP and PSP, a data mapping function can always be obtained for omega network of any size such that BRP and PSP will be admissible in one pass.

Definition 2.1 :

Let P_s be a perfect shuffle permutation on the set {0,1,...,N-1 }, where $N = 2^n$ and n is a positive integer. Let $(x_{n-1}x_{n-2}\cdots x_1x_0)$ be the binary expansion of an element x of the set. Then the PSP of x is given by

$$P_s(x) = (x_{n-2}x_{n-3}\cdots x_0x_{n-1}).$$

In other words, $P_s(x)$ is obtained by cyclically shifting the binary expansion of x one place to the left. The corresponding $n \times n$ binary matrix $\mathbf{P}_s$ is

$$\begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

□

Definition 2.2 :

Let P_b be a bit reversal permutation on the set {0,1,...,N-1 }, where $N = 2^n$ and n is a positive integer. Let $(x_{n-1}x_{n-2} \cdots x_1x_0)$ be the binary expansion of an element x of the set. Then the BRP of x is given by

$$P_b\,(x) = (x_0x_1 \cdots x_{n-2}x_{n-1}).$$

In other words, $P_b\,(x)$ is obtained by reversing the order of the bits of the binary expansion of x. The corresponding $n \times n$ binary matrix $\mathbf{P}_b$ is

$$\begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix}$$

□

Let F be a data mapping function with which both PSP and BRP can be realized in one pass. Then $\mathbf{P}_s[\mathbf{F}]^{-1}$ and $\mathbf{P}_b[\mathbf{F}]^{-1}$ in $\underline{y} = \mathbf{P}_s[\mathbf{F}]^{-1}\underline{x}$ and $\underline{y} = \mathbf{P}_b[\mathbf{F}]^{-1}\underline{x}$, respectively, are decomposable into $\mathbf{LU}'$ s. Let the $n \times n$ binary matrix $[\mathbf{F}]^{-1}$ be as shown below

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-2} & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-2} & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-2} & a_{2,n-1} \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ a_{n-3,0} & a_{n-3,1} & a_{n-3,2} & \cdots & a_{n-3,n-2} & a_{n-3,n-1} \\ a_{n-2,0} & a_{n-2,1} & a_{n-2,2} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}$$

The matrix product $\mathbf{P}_s[\mathbf{F}]^{-1}$ is then

$$\begin{bmatrix} a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-2} & a_{1,n-1} \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-2} & a_{2,n-1} \\ a_{3,0} & a_{3,1} & a_{3,2} & \cdots & a_{3,n-2} & a_{3,n-1} \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ a_{n-2,0} & a_{n-2,1} & a_{n-2,2} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \\ a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-2} & a_{0,n-1} \end{bmatrix}$$

The matrix product $\mathbf{P}_b[\mathbf{F}]^{-1}$ is then

$$\begin{bmatrix} a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \\ a_{n-2,0} & a_{n-2,1} & a_{n-2,2} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-3,0} & a_{n-3,1} & a_{n-3,2} & \cdots & a_{n-3,n-2} & a_{n-3,n-1} \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ a_{2,0} & a_{2,1} & a_{2,2} & \cdots & a_{2,n-2} & a_{2,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{1,n-2} & a_{1,n-1} \\ a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,n-2} & a_{0,n-1} \end{bmatrix}$$

Since $\mathbf{P}_s[\mathbf{F}]^{-1}$ and $\mathbf{P}_b[\mathbf{F}]^{-1}$ are **LU** decomposable, we will let $\mathbf{P}_s[\mathbf{F}]^{-1}$ and $\mathbf{P}_b[\mathbf{F}]^{-1}$ be decomposed into $\mathbf{L}_s\mathbf{U}_s$ and $\mathbf{L}_b\mathbf{U}_b$, respectively. Let $l_{s\,i,j}$, $u_{s\,i,j}$, $l_{b\,i,j}$ and $u_{b\,i,j}$, for $0 \leq i, j < n$, be the elements of the matrices $\mathbf{L}_s$, $\mathbf{U}_s$, $\mathbf{L}_b$ and $\mathbf{U}_b$, respectively. Then a system of $2n^2$ equations with $2(n^2-n) + n^2$ variables can be set up. For instance, consider $a_{1,0}$

$$a_{1,0} = \left(\sum_{j=1}^{n} l_{s\,0,j} u_{s\,j,0}\right) \mod (2),$$

$$a_{1,0} = \left(\sum_{j=1}^{n} l_{b\,n-2,j} u_{b\,j,0}\right) \mod (2).$$

In this example, we can eliminate $a_{1,0}$ immediately by equating the equations. Hence,

$$\left(\sum_{j=1}^{n} l_{s\,0,j} u_{s\,j,0} + \sum_{j=1}^{n} l_{b\,n-2,j} u_{b\,j,0}\right) \mod (2) = 0.$$

Solving this set of equations will produce F. Obviously, F does not exist if there is any contradiction between the equations. In general, for m different communication patterns, we will have mn^2 equations with $m(n^2 - n) + n^2$ variables. There is a limit to the number of different communication patterns that can be made simultaneously admissible. Clearly, the limit is dependent on the communication patterns required.

2.2.2. A Date Mapping for BRP and PSP on Omega Network

The system of nonlinear equations derived in the last section has been solved for different values of n. Observing the characteristics of the data mapping functions thus obtained, a general solution that is good for omega network of any size is deduced. For a $2^n \times 2^n$ omega network, the data mapping function **F** is determined to be

$$f_{i,j} = \begin{cases} 1 & \text{if } j = n-1-i \bmod(n) \\ 1 & \text{if } j = i+1 \qquad \text{for } \frac{n}{2} \le i \le n-2 \ (n \text{ even}) \text{ or } \frac{n-1}{2} \le i \le n-2 \ (n \text{ odd}), \\ 0 & \text{else} \end{cases}$$

where $0 \le i, j \le n-1$.

As an example, for $n = 5$, **F** is

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The corresponding mapping expressed in the convention used in [Chen81] is:

(0,18,12,30,4,22,8,26,2,16,14,28,6,20,10,24,1,19,13,31,5,23,9,27,3,17,15,29,7,21,11,25).

The corresponding $\mathbf{P}_s[\mathbf{F}]^{-1}$ is

$$p_{i,j} = \begin{cases} 1 & \text{if } j=n-2-i \bmod(n) \\ 1 & \text{if } j=i \qquad \text{for } i< \textit{floor of } \frac{n}{2}. \\ 0 & \textit{else} \end{cases}$$

For $n = 5$, $\mathbf{P}_s[\mathbf{F}]^{-1}$ is

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

The corresponding $\mathbf{L}_s\mathbf{U}_s$ matrix is given by

$$l_{s\,i,j} = \begin{cases} 1 & \text{if } j=i \\ 1 & \text{if } i=n-2-j \text{ for } j< \textit{floor of } \frac{n}{2}. \\ 0 & \textit{else} \end{cases}$$

$$u_{s\,i,j} = \begin{cases} 1 & \text{if } j=i \\ 1 & \text{if } j=n-2-i \text{ for } i< \textit{floor of } \frac{n}{2}. \\ 0 & \textit{else} \end{cases}$$

For $n = 5$,

$$\mathbf{L}_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$$\mathbf{U}_s = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

$\mathbf{P}_b[\mathbf{F}]^{-1}$ is simply equal to $\mathbf{L}_g$ and hence

$$\mathbf{L}_b = \mathbf{L}_g,$$

$$\mathbf{U}_b = \mathbf{I},$$

where $\mathbf{I}$ is a $n \times n$ identity matrix with its diagonal elements equal to 1.

In what follows, we will prove that $\mathbf{F}$ is indeed a general solution. With data mapping function $\mathbf{F}$, BRP and PSP will both be admissible in one pass through an omega network.

Lemma 2.1

Let $\mathbf{B}$ be a $n \times n$ nonsingular binary matrix and $b_{p,q}$ be the element in the pth row and qth column of $\mathbf{B}$ such that,

$$b_{p,q} = \begin{cases} 1 & \text{if } q = n-1-p \bmod(n) \\ 1 & \text{if } q = p-1 \\ 0 & \text{else} \end{cases} \qquad \text{for } 1 \le p \le \frac{n}{2} - 1 \ (n \text{ even}) \text{ or } 1 \le p \le \frac{n-1}{2} \ (n \text{ odd}).$$

Then $\mathbf{B}$ is the inverse of $\mathbf{F}$.

Proof :

We are going to prove that $\mathbf{FB} = \mathbf{I}$. We will first define two matrices, $\mathbf{C}$ and $\mathbf{D}$ such that $\mathbf{C} + \mathbf{D} = \mathbf{F}$. We will then prove that $\mathbf{CB} + \mathbf{DB} = \mathbf{FB} = \mathbf{I}$.

Let $c_{i,j}$ and $d_{i,j}$ be the elements in the ith row and jth column of $\mathbf{C}$ and $\mathbf{D}$, respectively, and they are defined as below,

$$c_{i,j} = \begin{cases} 1 & \text{if } j = n-1-i \bmod(n) \\ 0 & \text{else} \end{cases}$$

$$d_{i,j} = \begin{cases} 1 & \text{if } j=i+1 \quad \text{for } \frac{n}{2} \le i \le n-2 \text{ (} n \text{ even) or } \frac{n}{2} \le i \le n-2 \text{ (} n \text{ odd).} \\ 0 & \text{else} \end{cases}$$

Consider **CB**, the matrix multiplication is equivalent to replacing p in the definition of $b_{p,q}$ with $n-1-i$. For $q=n-1-p$, we will have $q=i$, and for $q=p-1$, we will have $q=n-i-2$. The limits, 1, $\frac{n}{2}-1$ and $\frac{n-1}{2}$ become $n-2$, $\frac{n}{2}$ and $\frac{n-1}{2}$. Let $cb_{i,q}$ be the element in the ith row and qth column of **CB**. Then

$$cb_{i,q} = \begin{cases} 1 & \text{if } q=i \\ 1 & \text{if } q=n-i-2 \text{ for } \frac{n}{2} \le i \le n-2 \text{ (} n \text{ even) or } \frac{n-1}{2} \le i \le n-2 \text{ (} n \text{ odd).} \\ 0 & \text{else} \end{cases}$$

The matrix multiplication of **D** and **B** is again equivalent to

1. replacing p in $q=n-1-p$ (in the definition of **B**) with $i+1$, for $\frac{n}{2} \le i \le n-2$ (n *even*) *or* $\frac{n-1}{2} \le i \le n-2$ (n *odd*)

2. replacing p in $q=p-1$ with $i+1$, for i that satisfies the following two requirements simultaneously :

   a. $\frac{n}{2} \le i \le n-2$ (n *even*) or $\frac{n-1}{2} \le i \le n-2$ (n *odd*)

   b. $1 \le i+1 \le \frac{n}{2}-1$ (n *even*) or $1 \le i+1 \le \frac{n-1}{2}$ (n *old*)

Obviously, the two requirements in 2 cannot be satisfied simultaneously. The element, $db_{i,j}$, in the ith row and qth column of **DB** is

$$db_{i,q} = \begin{cases} 1 & \text{if } q=n-i-2 \quad \text{for } \frac{n}{2} \le i \le n-2 \text{ (} n \text{ even) or } \frac{n-1}{2} \le i \le n-2 \text{ (} n \text{ odd).} \\ 0 & \text{else} \end{cases}$$

$\mathbf{CB} + \mathbf{DB} = \mathbf{FB} = \mathbf{I}$. Therefore, $\mathbf{B}$ is the inverse of $\mathbf{F}$.

□

Lemma 2.2

$\mathbf{P}_b\mathbf{F}^{-1}$ is **LU** decomposable.

Proof :

Now, matrix $\mathbf{C}$ in lemma 2.1 is actually a bit reversal permutation (BRP). Since $\mathbf{B} = \mathbf{F}^{-1}$, the matrix product $\mathbf{CF}^{-1} = \mathbf{CB}$ is given in the previous lemma. If we examine $\mathbf{CB}$ carefully, it can be seen that $cb_{i,j} = 0$, for all $i < j$. Furthermore, $cb_{i,j} = 1$, for $i = j$, $0 \leq i \leq n-1$. Therefore, $\mathbf{CB}$ is a lower triangular unit matrix. Clearly, $\mathbf{P}_b\mathbf{F}^{-1}$ is **LU** decomposable, and $D(\mathbf{P}_b\mathbf{F}^{-1}) \leq 1$.

□

Let $p_{i,j}$ be the element in the ith row and jth column of the matrix $\mathbf{P}_g$. Then,

$$p_{i,j} = \begin{cases} 1 & \text{if } j=i+1 \qquad \text{for } 0 \leq i \leq n-2 \\ 1 & \text{if } j=n-1-i \bmod(n) \quad \text{for } i=n-1 \\ 0 & \textit{else} \end{cases}$$

Matrix multiplication of $\mathbf{P}_g$ and $\mathbf{F}^{-1}$ is again equivalent to

1. replacing p in $q=n-1-p$ (in the definition of $b_{p,q}$) with $i+1$, for $0 \leq i \leq n-2$

2. replacing p in $q=n-1-p$ with $n-1-i$, for $i=n-1$

3. replacing p in $q=p-1$ with $i+1$, for i that satisfies the following requirements simultaneously

   a. $1 \leq i+1 \leq \frac{n}{2}-1$ (n even) or $1 \leq i+1 \leq \frac{n-1}{2}$ (n odd),

b. $0 \leq i \leq n\text{-}2$.

The limits 1, $\frac{n}{2}-1$ and $\frac{n-1}{2}$ now become 0, $\frac{n}{2}$ - 2 and $\frac{n-3}{2}$. Let $pb_{i,q}$ be the element in the ith row and qth column of $\mathbf{P}_\theta\mathbf{F}^{-1}$. Then

$$pb_{i,q} = \begin{cases} 1 & \text{if } q=n\text{-}2\text{-}i \quad \text{for } 0 \leq i \leq n\text{-}2 \\ 1 & \text{if } q=i \quad \text{for } 0 \leq i \leq \frac{n}{2}-1 \text{ (} n \textit{ even} \text{) } \textit{or } 0 \leq i \leq \frac{n-3}{2} \text{ (} n \textit{ odd} \text{).} \\ 1 & \text{if } j=n\text{-}1 \quad \text{for } i=n\text{-}1 \\ 0 & \textit{else} \end{cases}$$

Lemma 2.3

$\mathbf{P}_\theta\mathbf{F}^{-1}$ is **LU** decomposable.

Proof :

Let **E** be a matrix with its element in the ath row and bth column defined as follows.

$$e_{a,b} = \begin{cases} 1 & \text{if } b=n\text{-}2\text{-}a \quad \text{for } \frac{n}{2} \leq a \leq n\text{-}2 \text{ (} n \textit{ even} \text{) } \textit{or } \frac{n-1}{2} \leq a \leq n\text{-}2 \text{ .} \\ 0 & \textit{else} \end{cases}$$

Now, multiply **E** by $\mathbf{P}_\theta\mathbf{F}^{-1}$. This corresponds to

1. replacing i in $q = n\text{-}2\text{-}i$ (in the definition of $pf_{i,q}$) with $n\text{-}2\text{-}a$ for a that satisfies the following two requirements simultaneously

   a. $\frac{n}{2} \leq a \leq n\text{-}2$ (n even) or $\frac{n-1}{2} \leq a \leq n\text{-}2$ (n odd)

   b. $0 \leq n\text{-}2\text{-}a \leq n\text{-}2$

2. replacing i in the equation $q = i$ with $n\text{-}2\text{-}a$ for a that satisfies the following requirements simultaneously

a. $\frac{n}{2} \leq a \leq n-2$ (n even) or $\frac{n-1}{2} \leq a \leq n-2$ (n odd)

b. $0 \leq n-2-a \leq \frac{n}{2}-1$ (n even) or $0 \leq n-2-a \leq \frac{n-3}{2}$ (n odd)

Since $e_{n-1,h} = 0$, we do not have to consider $pf_{n-1,n-1}$ at all. Let $ep_{a,q}$ be the element of $\mathbf{EP}_s\mathbf{F}^{-1}$ in the ith row and jth column. Then

$$ep_{a,q} = \begin{cases} 1 & \text{if } q=a \quad \text{for } \frac{n}{2} \leq a \leq n-2 \text{ (} n \text{ even) or } \frac{n-1}{2} \leq a \leq n-2 \text{ (} n \text{ odd)} \\ 1 & \text{if } q=n-2-a \quad \text{for } \frac{n}{2} \leq a \leq n-2 \text{ (} n \text{ even) or } \frac{n-1}{2} \leq a \leq n-2 \text{ (} n \text{ odd)} . \\ 0 & \text{else} \end{cases}$$

Now, let $\mathbf{I} + \mathbf{E} = \mathbf{L}_t$. They form a lower trianguiar unit matrix with its element in the ath row and bth column defined as follows.

$$l_{a,b} = \begin{cases} 1 & \text{if } b=a \\ 1 & \text{if } b=n-2-a \quad \text{for } \frac{n}{2} \leq a \leq n-2 \text{ (} n \text{ even) or } \frac{n-1}{2} \leq a \leq n-2 \text{ (} n \text{ odd)} . \\ 0 & \text{else} \end{cases}$$

Clearly, $\mathbf{IP}_s\mathbf{F}^{-1} = \mathbf{P}_s\mathbf{F}^{-1}$. Now, $\mathbf{L}_t\mathbf{P}_s\mathbf{F}^{-1} = \mathbf{EP}_s\mathbf{F}^{-1} + \mathbf{P}_s\mathbf{F}^{-1}$. Let $\mathbf{U}_t = \mathbf{EP}_s\mathbf{F}^{-1} + \mathbf{P}_s\mathbf{F}^{-1}$. Then its elements $u_{a,b}$ are defined as follows.

$$u_{a,b} = \begin{cases} 1 & \text{if } b = a \\ 1 & \text{if } b=n-2-a \quad \text{for } 0 \leq a \leq \frac{n}{2}-2 \text{ (} n \text{ even) or } 0 \leq a \leq \frac{n-3}{2} \text{ (} n \text{ odd)} . \\ 0 & \text{else} \end{cases}$$

Now, $\mathbf{U}_t$ is an upper triangular unit matrix. Since $\mathbf{L}_t\mathbf{P}_s\mathbf{F}^{-1} = \mathbf{U}_t$, $\mathbf{P}_s\mathbf{F}^{-1} = \mathbf{L}_t^{-1}\mathbf{U}_t$. Therefore, $\mathbf{P}_s\mathbf{F}^{-1}$ is $\mathbf{LU}$ decomposable.

□

Theorem 2.2

There always exists a data mapping function for BRP and PSP, so that both will be admissible to an omega network of any size. One such data mapping function is **F** whose element in the *i*th row and *j*th column is given by

$$f_{i,j} = \begin{cases} 1 & \text{if } j = n\text{-}i\text{-}1 \\ 1 & \text{if } j = i+1 \quad \text{for } \frac{n}{2} \le i \le n\text{-}2 \text{ (} n \text{ even) or } \frac{n-1}{2} \le i \le n\text{-}2 \text{ (} n \text{ odd) .} \\ 0 & \text{else} \end{cases}$$

Proof :

Based on lemma 2.2 and 2.3, with data mapping function, **F**, BRP and PSP are admissible in one pass through an omega network of any size.

□

2.2.3. Applications for Other Networks

A network, N_1, is functionally equivalent[WuFe79, Prad79] to the omega network iff

$$\underline{Q_1} = R_1 \Omega R_2,$$

where $\underline{Q_1}$ is the set of admissible network permutations for N_1,

R_1 and R_2 are permutations.

Any permutation in $\underline{Q_1}$ can be decomposed into three functions, a permutation R_1, an admissible network permutation in Ω and a permutation R_2. If R_1 and R_2 are linear in the components of the indices, then a permutation, P, that is linear in the components of the index is in $\underline{Q_1}$ if and only if **P** is decomposable into a $\mathbf{R_1L}$ and an $\mathbf{UR_2}$. Given some permutations that are linear in the components of the indices, a data mapping function for these

permutations on N_1, if there exists one, can be determined by solving the system of nonlinear equations that can be set up similarly as in section 2.2.1.

2.3. A Larger Class of Permutations

Since once the desired permutations are put into matrix forms, **F**, if it exists, can be found systematically with the algorithm described. The applicability of the algorithm is clearly limited to those cases where a permutation can be expressed as a linear combination of the components of the index.

The class of communication patterns that can be expressed in a nonsingular binary matrix form is a small percentage of the $N!$ possible input to output per nutations. In fact, the percentage decreases rapidly as N increases. Fortunately, as Lenfant [Lenf78] pointed out, he set of needed permutations is usually small, as compared to the set of $N!$ possible permutations. He collected Frequently Used Bijections (FUBs) into five families. By extending our approach of representing a permutation in a matrix form, the method can now be used for three families of Lenfant's FUBs. In fact, this larger class of permutations, which we designate by Δ, contains also the class of bit-permute-complement studied in [NaSa77,81A,81B]. We now describe Δ[IrWu82].

Let $P:(x_{n-1}, x_{n-2}, \ldots, x_0) \rightarrow (y_{n-1}, y_{n-2}, \ldots, y_0)$ be any mapping that is linear in the components of x. Then it can be expressed in binary matrix form as shown below.

$$\mathbf{P} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & \cdots & a_{0,n-2} & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-2} & a_{1,n-1} \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}$$

where $a_{i,j} \in \{0,1\}$.

Let $k_{n-1}k_{n-2}\cdots k_0$ denote the binary representation of k, $0\leq k\leq 2^n-1$. Let θ_k be a bijection defined by

$$\theta_k : (x_{n-1}, x_{n-2}, \ldots, x_0)\rightarrow(x_{n-1}, x_{n-2}, \ldots, x_0)\oplus k,$$

where $(x_{n-1}, x_{n-2}, \ldots, x_0)\oplus k$ denotes bit-per-bit exclusive-or of $(k_{n-1}, k_{n-2}, \ldots, k_0)$ with $(x_{n-1}, x_{n-2}, \ldots, x_0)$.

Definition 2.3 :

Let $P_{\oplus k} = \theta_k \cdot P$ be the left composition of θ_k with P. Then Δ, the set of permutations with which we will be concerned, is defined by

$$\Delta = \{P_{\oplus k} \mid 0\leq k\leq 2^n-1 \}.$$

□

Clearly, $P_{\oplus k}$ cannot be expressed in the matrix form described earlier except when k=0. However, a $2^{n+1}\times 2^{n+1}$ omega network can be partitioned into two $2^n\times 2^n$ omega network by not allowing any exchange at the first stage of the network[Smit78,Sieg78]. This is equivalent to requiring the first row vector of a $(n+1)\times(n+1)$ permutation matrix for the network to be [1 0 0...0]. The $(n+1)\times(n+1)$ matrix below now specifies two permutations, P and $P_{\oplus k}$, each of which passes through one of the two subnetworks of order $2^n\times 2^n$.

$$\dot{\mathbf{P}} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ k_{n-1} & a_{0,0} & a_{0,1} & \cdots & a_{0,n-2} & a_{0,n-1} \\ k_{n-2} & a_{1,0} & a_{1,1} & \cdots & a_{1,n-2} & a_{1,n-1} \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdots & \cdot & \cdot \\ k_0 & a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}$$

In other words, $\dot{\mathbf{P}}$ is admissible through a $2^{n+1}\times 2^{n+1}$ omega network iff $P_{\oplus k}$ and P are admissible through a $2^n\times 2^n$ omega network. Therefore, we can use a $(n+1)\times(n+1)$ nonsingular binary matrix to represent any $P_{\oplus k}$ (including k =0). In this form, theorem 2.1 states that a permutation $P_{\oplus k}$ is admissible through a $2^n\times 2^n$ omega network iff its corresponding $(n+1)\times(n+1)$ matrix is decomposable into **LU**, where the first row of **U** is [1 0 0...0].

Let $\dot{\mathbf{F}}$ be a $(n+1)\times(n+1)$ nonsingular binary matrix. For $\dot{P}[\dot{F}]^{-1}$ to specify two communication patterns that correspond to **P** and $\mathbf{P}_{\oplus k}$ with alternate mappings, so that each communica ion pattern passes through one of the two $2^n\times 2^n$ subnetworks, the first row of $\dot{P}[\dot{F}]^{-1}$ shall be equal to the row vector [1 0 0... 0]. This, in turn, requires that the first row of $\dot{F}$ be [1 0 0 $\cdots$ 0]. If $\dot{\mathbf{F}}$ is the data mapping matrix for the data mapping function

$$\dot{F}: (x_n,x_{n-1},x_{n-2},\ldots,x_0)\rightarrow(y_n,y_{n-1},y_{n-2},\ldots,x_0),$$

then the mapping functions for $P_{\oplus k}$ and P are

$$\dot{F}_1 : (1,x_{n-1},x_{n-2},\ldots,x_0)\rightarrow(1,y_{n-1},y_{n-2},\ldots,x_0) \text{ and}$$

$$\dot{F}_0 : (x_n,x_{n-1},x_{n-2},\ldots,x_0)\rightarrow(0,y_{n-1},y_{n-2},\ldots,x_0), \text{ respectively.}$$

Henceforth, in this report, a permutation in Δ will be represented by a $(n+1)\times(n+1)$ binary matrix whose 1st row is [1 0 0...0] even when $k = 0$. Furthermore, when we talk about an upper triangular unit matrix, we are concerned with a $(n+1)\times(n+1)$ upper triangular unit matrix whose first row is [1 0 0...0]. Since every permutation in Δ is **LUL** decomposable [Peas77], at most two passes through a $2^n\times 2^n$ omega n etwork are needed to realize anyone of them. Furthermore, if the permutation is an identity function, no routing through the network is necessary. Therefore, if D is the distance function associated with the $2^n\times 2^n$ omega network, then $0\leq D(P)\leq 2$, for all P in Δ. Since we are assuming a $2^n\times 2^n$ omega network for the rest of this report, unless otherwise stated, the distance function D will be associated with the $2^n\times 2^n$ omega network.

The cardinality of Δ can be computed in many ways. Based on the following lemma we can determine the cardinality of Δ.

Lemma 2.4 :

There are $\prod_{i=0}^{n-1}(2^n - 2^i)$ nonsingular $n \times n$ binary matrices.

Proof :

The lemma is to be proved by mathematical induction. A binary matrix is nonsingular if every row of it is linearly independent of all the other rows in the matrix. There are 2^n different combinations of 0's and 1's. Excluding the row of all zeros, there are $(2^n - 2^0)$ different rows.

In choosing the first row, there are $(2^n - 2^0)$ choices. There are 2^1-1 rows that are lir early dependent on the first row. Hence, for the second row, we have $(2^n - 2^1)$ choices (excluding the row of all zeros). There are $(2^n - 2^0)(2^n - 2^1)$ possible combinations of the first two rows.

Now, assume there are $\prod_{i=0}^{j-2}(2^n - 2^i)$ possible combinations of the first j-1 rows. There are $2^{j-1}-1$ rows that are linearly dependent on any one of such combinations. Hence, for the jth rows, we have $(2^n - 2^{j-1})$ choices. There are $\prod_{i=0}^{j-1}(2^n - 2^i)$ possible combinations of the first j rows. Therefore, there are $\prod_{i=0}^{n-1}(2^n - 2^i)$ nonsingular $n \times n$ binary matrices.

□

Lemma 2.5 :

The cardinality of Δ is $\prod_{i=1}^{n}(2^{n+1} - 2^i)$.

Proof :

There are $\prod_{i=0}^{n}(2^{n+1} - 2^i)$ nonsingular $(n+1)\times(n+1)$ nonsingular binary matrix. But the first row of a matrix in Δ has to be [1, 0,...., 0]. The cardinality of Δ is then $\prod_{i=0}^{n}(2^{n+1} - 2^i)$ divided by $(2^{n+1} - 2^0)$. That is, $\prod_{i=1}^{n}(2^{n+1} - 2^i)$.

□

For large N, the cardinality of Δ is only a small percentage of the possible $N!$ permutations. Nevertheless, many of the permutations encountered in parallel algorithms are in this class. Table I lists several of the more popular permutations in the class of bit-permute-complement (BPC).

Table I Example Permutations in BPC

Permutations[1]	$\underline{y} = (y_{n-1}, y_{n-2}, \ldots, y_0) = P(\underline{x})$
Matrix Transpose	$(x_{\frac{n}{2}-1}, \ldots, x_0, x_{n-1}, \ldots, x_{\frac{n}{2}})$
Bit Reversal	$(x_0, x_1, x_2, \ldots, x_{n-1})$
Vector Reversal	$(x_{n-1}, x_{n-2}, \ldots, x_0) \oplus (2^n-1)$
Perfect Shuffle	$(x_0, x_{n-1}, x_{n-2}, \ldots, x_1)$
Unshuffle	$(x_{n-2}, x_{n-3}, \ldots, x_0, x_{n-1})$
Shuffle Row Major	$(x_{n-1}, x_{\frac{n}{2}-1}, x_{n-2}, x_{\frac{n}{2}-2}, \ldots, x_{\frac{n}{2}}, x_0)$
Bit Shuffle	$(x_{n-1}, x_{n-3}, \ldots, x_1, x_{n-2}, x_{n-4}, \ldots, x_0)$

Further characterization of Δ in later chapters leads to efficient algorithms for networks in Γ, a subset of the set of all networks that are functionally equivalent to the omega network. Nevertheless, the method developed in this chapter demonstrates the general concept

[1] $\underline{x} = (x_{n-1}, x_{n-2}, \ldots, x_0)$

of data mapping and is indicative of how complex the problem can possibly become if the multicomputer system has a network that is not in Γ.

CHAPTER 3

NON-STATIC DATA MAPPING

We shall now develop techniques for determining non-static data mappings for the vector variables of a given parallel program such that its total communication cost can be minimized.

We will assume that alignment functions of any expression tree are identity functions. Furthermore, we will assume that the logical transfers for binary operations on temporary variables are identity functions. If the logical transfer for a temporary variable is not identity, then it can be transformed to identity through simple recursive transformations. For instance, in figure 3.1 -A, logical transfers for aligning vectors a and b to w_1 are P_a and P_b, respectively. Similarly, logical transfers for c, d, w_1, w_2 and w_3 are P_c, P_d, P_1, P_2 and P_3, respectively. Through transformations, the logical transfers for w_1, w_2 and w_3 become P_3P_1, P_3P_2 and **I**, respectively [1] (figure 3.1 -B). Figure 3.1 -C can be obtained by applying the transformation again.

For the beginning, we will also work under the restriction that there is no precedence relationship between expression trees, i.e., leaf variables are not the results of other expression trees. Earlier assumptions plus this restriction allows us to develop an algorithm that guarantees minimum cost. The algorithm also sets the stage for the development of a heuristic algorithm for a given parallel program where precedence relationship between its expression trees exists.

---

[1] **I** stands for identity function

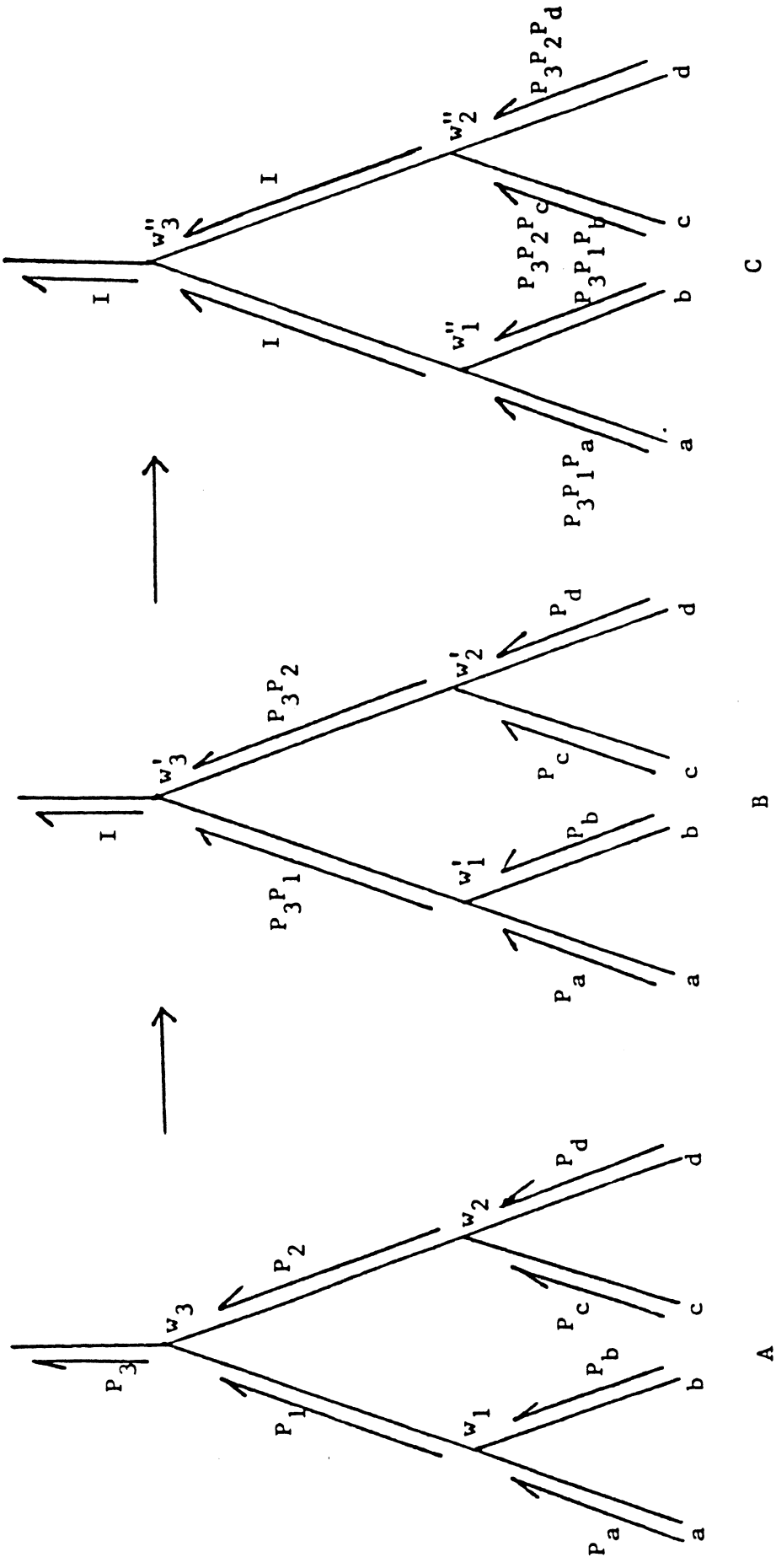


Figure 3.1 Example for recursive transformation.

Under the above constraints, the communication cost function of a parallel program is given by

$$\left[\begin{array}{l} \sum_{\substack{\text{statements}\\ \text{of the}\\ \text{given}\\ \text{algorithm}}} \quad \sum_{\substack{v_i:\ \text{variable}\\ w_j:\ \text{partial}\\ \text{result}\\ w_j = \text{parent}\\ (v_i)}} D(P_{v_i w_j} F_{v_i,h}^{-1}) \end{array}\right] + \sum_{\substack{v_i:\ \text{variable}\\ F_{v_i,h}:\ \text{data mapping}\\ \text{function}}} D(F_{v_i,h+1} F_{v_i,h}^{-1})$$

where $F_{v_i,h}$ is the data mapping function for the hth logical transfer of vector variable v_i.

To minimize the cost function, one therefore, has to determine for each vector v an ordered sequence of data mapping functions. Since there is no precedence relationship among expression trees, determination of an optimum sequence of data mapping functions, $\hat{S}_v$, for any variable, v, does not change any logical transfer for any other variables. Hence, the minimization problem for a program with j vector variables can be decomposed into j independent subproblems, each specifying the communication requirements of a single variable.

For a vector v, all its communication requirements are specified by a sequence of logical transfers, $\hat{P}_{v,k} = P_{v,1}, P_{v,2}, \cdots, P_{v,k}$. The method for solving for an optimum sequence of data mapping functions, $\hat{S}_v = F_{v,1}, F_{v,2}, \ldots, F_{v,k}$, for v so that the total communication cost,

$$\sum_{h=1}^{k} D(P_{v,h} F_{v,h}^{-1}) + \sum_{h=2}^{k} D(F_{v,h} F_{v,h-1}^{-1})$$

is minimized, can be applied to each of the j subproblems. The set of j sequences of data mapping functions thus obtained is a solution to the minimization problem. Since the algorithm that we will be describing deals with the variable v only, subscript v will be dropped from $F_{v,h}$ and $P_{v,h}$ from now on.

3.1. Logical Transfers and Data Mapping Functions

Characterization of the logical transfers and data mapping functions are needed for describing the algorithm. The following two basic lemmas can be obtained easily from linear algebra[Hers64]. Hence, we merely state them here.

Lemma 3.1

The **LU** decomposition of a nonsingular binary matrix is unique.

□

Lemma 3.2

The set of all lower triangular unit matrices and the set of all upper triangular unit matrices are groups under matrix multiplication (modulo 2).

□

We will designate the set of all lower triangular unit matrices and the set of all upper triangular matrices by $\underline{\mathbf{L}}$ and $\underline{\mathbf{U}}$, respectively. Two matrices, $\mathbf{P}_1$ and $\mathbf{P}_2$, are in the same coset of $\underline{\mathbf{R}}$, $\underline{\mathbf{R}} = \underline{\mathbf{L}}$ or $\underline{\mathbf{R}} = \underline{\mathbf{U}}$, iff $\mathbf{P}_1$ and $\mathbf{P}_2$ are in $\underline{\mathbf{R}}\mathbf{P}_1 = \{\mathbf{R}_x\mathbf{P}_1 \mid \mathbf{R}_x \in \underline{\mathbf{R}}\}$.

Theorem 3.1

Let $D(P_iF_i^{-1}) \leq 1$. If $\mathbf{P}_j \in \underline{\mathbf{L}}\mathbf{P}_i$, then $D(P_jF_i^{-1}) \leq 1$.

Proof :

Let $\mathbf{P}_i\mathbf{F}^{-1} = \mathbf{LU}$. Then $\underline{\mathbf{L}}\mathbf{P}_i\mathbf{F}_i^{-1} = \underline{\mathbf{L}}\mathbf{U}$. Therefore, if $\mathbf{P}_j \in \underline{\mathbf{L}}\mathbf{P}_i$, then $\mathbf{P}_j\mathbf{F}_i^{-1} \in \underline{\mathbf{L}}\mathbf{U}$ and $D(P_jF_i^{-1}) \leq 1$.

□

Theorem 3.2

Given $\mathbf{P}_1$, if $D(P_1F_1^{-1}) \leq 1$ and $\mathbf{F}_2 \in \underline{\mathbf{U}}\mathbf{F}_1$, then $D(P_1F_2^{-1}) \leq 1$.

Proof :

If $\mathbf{P}_1\mathbf{F}_1^{-1} = \mathbf{LU}$, then $\mathbf{P}_1\mathbf{F}_1^{-1}\underline{\mathbf{U}} = \mathbf{L}\underline{\mathbf{U}}$. Therefore, if $\mathbf{F}_2 = \mathbf{U}_i\mathbf{F}_1$, then $\mathbf{P}_1\mathbf{F}_1^{-1}\mathbf{U}_i^{-1} \in \underline{\mathbf{LU}}$, and $D(P_1F_2^{-1}) \leq 1$.

□

Theorem 3.3

The set of all data mapping functions $\underline{F}_{all,h}$, that makes logical transfer P_h in the sequence $\hat{P}_k$ realizable in one or zero pass is given by

$$\underline{F}_{all,h} = \{\mathbf{U}_i\mathbf{L}_j\mathbf{P}_h | \mathbf{L}_j \in \underline{\mathbf{L}} \ \& \ \mathbf{U}_i \in \underline{\mathbf{U}}\}.$$

The cardinality of $\underline{F}_{all,h}$ is 2^{n^2}.

Proof :

Let $\mathbf{F}_i = \mathbf{ULP}_h$. Then $(\mathbf{P}_h\mathbf{F}_i^{-1}) = \mathbf{P}_h(\mathbf{ULP}_h)^{-1} = \mathbf{L}^{-1}\mathbf{U}^{-1}$ is $\mathbf{LU}$ decomposable. Therefore, if $\mathbf{F}_i = \mathbf{ULP}_h$, then $\mathbf{F}_i \in \underline{\mathbf{F}}_{all}$.

Let $\mathbf{F}_i \in \underline{F}_{all}$. Then $\mathbf{P}_h\mathbf{F}_i^{-1} = \mathbf{L}_a\mathbf{U}_b$ and $\mathbf{F}_i = \mathbf{U}_b^{-1}\mathbf{L}_a^{-1}\mathbf{P}_h$.

Therefore,

$$\underline{F}_{all} = \{ \mathbf{U}_r\mathbf{L}_s P_h \mid \mathbf{U}_r \in \underline{\mathbf{U}} \ \& \ \mathbf{L}_s \in \underline{\mathbf{L}} \}.$$

Since there are $2^{\frac{(n^2-n)}{2}}$ $\mathbf{U}$'s and $2^{\frac{(n^2+n)}{2}}$ $\mathbf{L}$'s, $|\underline{F}_{all,h}| = 2^{n^2}$.

□

The following two lemmas and theorem form the basis for theorem 3.5 that establishes the worst case communication cost for non-static data mapping.

Lemma 3.3

Given two logical transfers, P_h and P_{h+1}, for vector variable v, there exists a data mapping function that makes both logical transfers simultaneously realizable in one pass.

Proof :

Now, every nonsingular binary matrix is **LUL** decomposable[Peas77]. Let $\mathbf{P}_h\mathbf{P}_{h+1}^{-1} = \mathbf{L}_r\mathbf{U}_s\mathbf{L}_t$ and let $\mathbf{F}_a = \mathbf{L}_t\mathbf{P}_{h+1}$. Then $D(P_hF_a^{-1}) \leq 1$ and $D(P_{h+1}F_a^{-1}) \leq 1$.

□

Lemma 3.4

Every nonsingular binary matrix is **ULU** decomposable.

Proof:

Let ρ be the bit reversal permutation matrix. Then $\rho\rho = I$, and $\rho = \rho^{-1}$. It can be easily shown that $\rho\mathbf{L}\rho = \mathbf{U}$, and $\rho\mathbf{U}\rho = \mathbf{L}$. Now, every nonsingular binary matrix is **LUL** decomposable [Peas77]. Consider an arbitrary matrix, $\mathbf{P} \in \Delta$. Let $\rho\mathbf{P}\rho = \mathbf{L}_r\mathbf{U}_s\mathbf{L}_t$. Then

$$\mathbf{P} = \rho\mathbf{L}_r\rho\rho\mathbf{U}_s\rho\rho\mathbf{L}_t\rho.$$

But $\rho\mathbf{L}_r\rho$, $\rho\mathbf{L}_t\rho \in \underline{\mathbf{U}}$, and $\rho\mathbf{U}_s\rho \in \underline{\mathbf{L}}$.

Therefore, every nonsingular binary matrix is **ULU** decomposable.

□

Theorem 3.4

Let $\hat{S} = F_1, F_2, \ldots, F_k$ be a sequence of data mapping functions for v. Let $F_i = F_{i+1} = \cdots = F_{i+h}$, $2 \leq i, i+h \leq k$, $F_{i-1} \neq F_i$, and $D(P_g F_i^{-1}) \leq 1$, $i \leq g \leq i+h$. If the remapping cost $D(F_i F_{i-1}^{-1}) = 2$, then we can always find a F_{new} to replace all of the $F_i, F_{i+1}, \ldots, F_{i+h}$ such that $D(P_g F_{new}^{-1}) \leq 1$, $i \leq g \leq i+h$, and the remapping cost is $D(F_{new} F_{i-1}^{-1}) \leq 1$.

Proof :

Let the remapping function, $\mathbf{F}_i \mathbf{F}_{i-1}^{-1}$ be decomposable into $\mathbf{U}_r \mathbf{L}_g \mathbf{U}_l$. Let $F_{new} = \mathbf{U}_r^{-1} \mathbf{F}_i$. Clearly, $D(F_{new} F_{i-1}^{-1}) \leq 1$. Furthermore, based on theorem 3.2, $D(P_g F_{new}^{-1}) \leq 1, i \leq g \leq i+h$.

□

Using the results of the above two lemmas and the theorem 3.4, the following theorem can be proved.

Theorem 3.5

Given k different logical transfers for variable v, the resulting communication cost after data mapping is given by

$$k\text{-}1 \leq cost \leq \frac{3}{2} k\text{-}1 \quad (k = \text{even}),$$

$$k\text{-}1 \leq cost \leq \frac{3k\text{-}1}{2} \quad (k = \text{odd}).$$

Proof:

Case : the lower bound

Since there are k different logical transfers, the total communication cost clearly cannot be less than k-1.

Case : the upper bound

Based on lemma 3.3, for every two permutations, we can always find a data mapping function that makes both permutations simultaneously realizable in one pass. If k is even, then we will have at most $\frac{k}{2}$ different data mapping functions or $\frac{k}{2}-1$ remappings. Based on theorem 3.4, the remapping cost between any two different consecutive data mapping functions can be less than or equal to one. Therefore, if k is even, the resulting communication cost is less than $k + \frac{k}{2}-1 = \frac{3}{2}k-1$.

If k is odd, then we will have at most $\frac{(k-1)}{2}+1$ different data mapping functions or $\frac{(k-1)}{2}$ remappings. Therefore, if k is odd, the resulting communication cost is less than or equal to $k + \frac{(k-1)}{2}$.

□

3.2. Characterization of Optimum Sequences

The algorithm that we will be describing shortly, involves partial enumeration of possible partial solutions. Rather than enumerating all sequences to find an optimum sequence, the algorithm generates possible subsequences of optimum sequences. Conceptually, during its hth step, subsequences of length h are formed by appending each subsequence from the h-1th step with data mapping functions that belong to F_h, a set of data mapping functions that will be defined later. The subsequences thus formed, are called intermediate subsequences. Intermediate subsequences that can lead to an optimum sequence are saved for the next step, while the rest are eliminated. The subsequences that are saved for the h+1th step are called candidate subsequences. A set of candidate subsequences of length h is denoted by SEQ_h. In what follows, we will describe rules for eliminating some of the intermediate subsequences from consideration. We will also describe F_h, the set of data mapping functions for step h of the algorithm.

Some basic notations and definitions for the algorithm are needed.

Notation 3.1 :

A sequence of data mapping functions for variable v is designated by a hat over a capital letter S. For example, $\hat{Si} = F_{1,i}, F_{2,i}, \ldots, F_{k,i}$ is a sequence of data mapping functions for variable v. A subsequence, $\hat{Si}_{a,b}$, of $\hat{Si}$ is given by $F_{a,i}, F_{a+1,i}, \ldots, F_{b,i}$. If a=1, then $\hat{Si}_{a,b}$ is the prefix of $\hat{Si}$ and is designated by $\hat{Si}_b$. Note : $\hat{Si} = \hat{Si}_k$.

□

Notation 3.2 :

The cost of a subsequence, $\hat{Si}_{a,b}$ of $\hat{Si}$, is denoted by either $D_{a,b}(\hat{Si})$ or $D_{a,b}(\hat{Si}_{a,b})$, and is given by

$$D_{a,b}(\hat{Si}_{a,b}) = D_{a,b}(\hat{Si}) = \sum_{h=a}^{b} D(P_h F_{h,i}^{-1}) + \sum_{h=a+1}^{b} D(F_{h,i} F_{h-1,i}^{-1}).$$

□

Notation 3.3 :

Let $\underline{S}_h$ be a set of subsequences of data mapping functions of length h. Then $Min(\underline{S}_h)$ is the cost of the minimum cost subsequence in $\underline{S}_h$.

□

Definition 3.1 :

A sequence $\hat{Si}$ is optimum if there does not exist another sequence $\hat{Sj}$, such that

$$D_{1,k}(\hat{Si}) > D_{1,k}(\hat{Sj}).$$

□

The following theorems determine the conditions under which an intermediate subsequence of length h is to be eliminated. We will consider two intermediate subsequences $\hat{S1}_h$ and $\hat{S2}_h$, where $\hat{S1}_h \neq \hat{S2}_h$. Given a sequence $\hat{S1}$ having $\hat{S1}_h$ as its subsequence, we will

examine conditions under which we can always find $S2$ that contains $S2_h$ such that $D_{1,k}(\hat{S1}) \geq D_{1,k}(\hat{S2})$. Since we are seeking an optimum solution only, these are conditions under which $\hat{S1}_h$ can be eliminated.

Theorem 3.6

Let $\hat{S1}_h$ be the prefix of $\hat{S1}$. Given a subsequence $\hat{S2}_h$ such that $D_{1,h}(\hat{S1}_h) - D_{1,h}(\hat{S2}_h) \geq 2$, we can always find $\hat{S2}$ containing $\hat{S2}_h$ such that $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

Proof :

The communication cost associated with $\hat{S1}$ is given by

$$D_{1,k}(\hat{S1}) = D_{1,h}(\hat{S1}_h) + D_{h,k}(\hat{S1}) + D(F_{h+1,1}F_{h,1}^{-1}).$$

Let $\hat{S2}_h$ be a subsequence of $\hat{S2}$ and $\hat{S2}_{h+1,k} = \hat{S1}_{h+1,k}$. The communication cost associated with $\hat{S2}$ is then given by

$$\begin{aligned} D_{1,k}(\hat{S2}) &= D_{1,h}(\hat{S2}_h) + D_{h,k}(\hat{S2}) + D(F_{h+1,2}F_{h,2}^{-1}) \\ &= D_{1,h}(\hat{S2}_h) + D_{h,k}(\hat{S1}) + D(F_{h+1,1}F_{h,2}^{-1}). \end{aligned}$$

Therefore,

$$\begin{aligned} D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) &= D_{1,h}(\hat{S1}_h) - D_{1,h}(\hat{S2}) + D(F_{h+1,1}F_{h,1}^{-1}) - D(F_{h+1,1}F_{h,2}^{-1}) \\ &\geq 2 + D(F_{h+1,1}F_{h,1}^{-1}) - D(F_{h+1,1}F_{h,2}^{-1}). \end{aligned}$$

But $0 \leq D(F_{h+1,1}F_{h,1}^{-1}), D(F_{h+1,1}F_{h,2}^{-1}) \leq 2$. Therefore,

$$D(F_{h+1,1}F_{h,1}^{-1}) - D(F_{h+1,1}F_{h,2}^{-1}) \geq -2, \text{ and } D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0.$$

Corollary 3.6.1

The candidate set, $\underline{SEQ}_h$, can therefore, be partitioned into two disjoint sets, $\underline{SEQ}_h^1$ and $\underline{SEQ}_h^0$, where $\hat{Si}_h \in \underline{SEQ}_h^0$ iff $D_{1,h}(\hat{Si}_h) = Min(\underline{SEQ}_h)$ and $\hat{Si}_h \in \underline{SEQ}_h^1$ iff $D_{1,h}(\hat{Si}_h) = Min(\underline{SEQ}_h) + 1$.

□

Theorem 3.7

Let $\hat{S1}_h$ be the prefix of $\hat{S1}$. Given a subsequence $\hat{S2}_h$ such that $F_{h,1} = F_{h,2}$ and $D_{1,h}(\hat{S1}_h) - D_{1,h}(\hat{S2}_h) \geq 0$, we can always find $\hat{S2}$ containing $\hat{S2}_h$ such that $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

Proof :

The communication cost associated with $\hat{S1}$ is given by

$$D_{1,k}(\hat{S1}) = D_{1,h}(\hat{S1}) + D_{h+1,k}(\hat{S1}) + D(F_{h+1,1}F_{h,1}^{-1}).$$

Let $\hat{S2}_h$ be a subsequence of $\hat{S2}$ and $\hat{S2}_{h+1,k} = \hat{S1}_{h+1,k}$. Then

$$\begin{aligned} D_{1,k}(\hat{S2}) &= D_{1,h}(\hat{S2}) + D_{h+1,k}(\hat{S2}) + D(F_{h+1,2}F_{h,2}^{-1}) \\ &= D_{1,h}(\hat{S2}) + D_{h+1,k}(\hat{S1}) + D(F_{h+1,1}F_{h,1}^{-1}). \end{aligned}$$

Therefore, $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

□

Theorem 3.8

Let $\hat{S1}_h$ be the prefix of $\hat{S1}$. Given a subsequence $\hat{S2}_h$ such that $\mathbf{F}_{h,1}\mathbf{F}_{h,2}^{-1} = \mathbf{L}_r\mathbf{U}_s$, $\mathbf{F}_{h,1} \neq F_{h,2}$ and $D_{1,h}(\hat{S1}_h) - D_{1,h}(\hat{S2}_h) \geq 1$, we can always find $\hat{S2}$ containing $\hat{S2}_h$ such that $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

Proof :

The communication cost associated with $\hat{S1}$ is given by

$$D_{1,k}(\hat{S1}) = D_{1,h}(\hat{S1}) + D_{h+1,k}(\hat{S1}) + D(F_{h+1,1}F_{h,1}^{-1}).$$

Let $\hat{S2}_h$ be a subsequence of $\hat{S2}$ and $\hat{S2}_{h+1,k} = \hat{S1}_{h+1,k}$, Then

$$\begin{aligned} D_{1,k}(\hat{S2}) &= D_{1,h}(\hat{S2}) + D_{h+1,k}(\hat{S2}) + D(F_{h+1,2}F_{h,2}^{-1}) \\ &= D_{1,h}(\hat{S2}) + D_{h+1,k}(\hat{S1}) + D(F_{h+1,1}F_{h,2}^{-1}). \end{aligned}$$

Therefore,

$$D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 1 + D(F_{h+1,1}F_{h,1}^{-1}) - D(F_{h+1,1}F_{h,2}^{-1}).$$

If $D(F_{h+1,1}F_{h,1}^{-1})=0$, then since $D(F_{h+1,1}F_{h,2}^{-1}) = D(F_{h,1}F_{h,2}^{-1}) = 1$, $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$. If $D(F_{h+1,1}F_{h,1}^{-1}) > 0$, then since $D(F_{h+1,1}F_{h,2}^{-1}) \leq 2$, $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

Therefore, if $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$ and $F_{h,1}F_{h,2}^{-1}$ is **LU** decomposable, then $D_{1,k}(\hat{S1}) - D_{1,k}(\hat{S2}) \geq 0$.

□

It will be inefficient to generate intermediate subsequences of length h from candidate subsequences of length h–1 by appending every one of the candidate subsequences with every permutation in Δ. Based on the following two theorems, we determine the set of data mapping functions, $\underline{F_h}$, that have to be appended to candidate subsequences in $\underline{SEQ_{h-1}}$ to guarantee an optimum solutions. Let $\underline{last_h}$, $1 \leq h \leq k-1$, be a set of data mapping functions such that F_a is in $\underline{last_h}$, iff F_a is the last mapping function of a candidate subsequence in $\underline{SEQ}_{h-1}$. Note : $\underline{last_1} = \emptyset$. Let $Map_h = \underline{last_{h-1}} \bigcap (\underline{F_{all,h-1}} \bigcup \underline{F_{all,h-2}} \bigcup \underline{F_{all,h-3}})$, where $\underline{F_{all,b}} = \emptyset$ if b is less than or equal to zero. The first of the following two theorems shows that it is sufficient for $\underline{F_h}$ to be equal to $\underline{last_{h-1}} \bigcup \underline{F_{all,h}}$. The second theorem shows that it is sufficient for $\underline{F_h}$ to be equal to $\underline{Map_h} \bigcup \underline{F_{all,h}} \subseteq \underline{last_{h-1}} \bigcup \underline{F_{all,h}}$.

Theorem 3.9

Let $\hat{Si}_{h-1}$ be a candidate subsequence in $\underline{SEQ_{h-1}}$ and $F_{h-1,i}$ be the last data mapping function in $\hat{Si}_{h-1}$. $\hat{Si}_{h-1},F_a \notin \underline{SEQ_h}$ if $F_a \neq F_{h-1,i}$ and $F_a \notin \underline{F_{all,h}}$.

Proof:

Let $\hat{Sj}_h$ be formed by appending F_a to $\hat{Si}_{h-1}$. Then

$$D_{1,h}(\hat{Sj}_h) = D_{1,h-1}(\hat{Si}_{h-1}) + D(P_hF_a^{-1}) + D(F_aF_{h-1,i}^{-1}).$$

Let $\hat{Si}_h$ be formed by appending $F_{h-1,i}$ to $\hat{Si}_{h-1}$. Then

$$D_{1,h}(\hat{Si}_h) = D_{1,h-1}(\hat{Si}_{h-1}) + D(P_hF_{h-1,i}^{-1}).$$

Therefore,

$$D_{1,h}(\hat{Sj}_h) - D_{1,h}(\hat{Si}_h) = D(P_hF_a^{-1}) + D(F_aF_{h-1,i}^{-1}) - D(P_hF_{h-1,i}^{-1}).$$

Since $F_a \neq \underline{F}_{all,h}$,

$$D(P_hF_a^{-1}) = 2,$$

and

$$D_{1,h}(\hat{Sj}_h) - D_{1,h}(\hat{Si}_h) = 2 + D(F_aF_{h-1,i}^{-1}) - D(P_hF_{h-1,i}^{-1}).$$

If $D(F_aF_{h-1,i}^{-1}) = 2$, then $D_{1,h}(\hat{Sj}_h) - D_{1,h}(\hat{Si}_h) \geq 2$. By theorem 3.6, $\hat{Sj}_h$ cannot be a candidate subsequence. If $D(F_aF_{h-1,i}^{-1}) = 1$, then $D_{1,h}(\hat{Sj}_h) - D_{1,h}(\hat{Si}_h) \geq 1$. By theorem 3.8, $\hat{Sj}_h$ cannot be a candidate subsequence. Therefore, $\hat{Sj}_h \notin \underline{SEQ}_h$.

Corollary 3.9.1

If $F_a \notin \underline{last}_{h-1} \bigcup \underline{F}_{all,h}$, then for all $\hat{St}_{h-1} \in \underline{SEQ}_{h-1}$, $\hat{St}_{h-1}F_a \notin \underline{SEQ}_h$. Therefore, it is sufficient for $\underline{F}_h = \underline{last}_{h-1} \bigcup \underline{F}_{all,h}$.

Corollary 3.9.2

If $\hat{Si}_h \in \underline{SEQ}_h$, $F_{h,i} \notin \underline{F}_{all,h}$, then $F_{h,i} = F_{h-1,i}$.

□

$\underline{F}_h$ according to theorem 3.9 may still grow larger and larger, with increasing h, until it equals Δ. The following theorem shows that that is not the case.

Theorem 3.10

Let $\hat{Si}_{h-1} = \hat{Si}_{h-4},F_{h-3,i},F_{h-2,i},F_{h-1,i}$ be a candidate subsequence in $\underline{SEQ}_{h-1}$. If $F_{h-1,i} \notin \underline{F}_{all,h} \bigcup \underline{F}_{all,h-1} \bigcup \underline{F}_{all,h-2} \bigcup \underline{F}_{all,h-3}$, then $\hat{Si}_{h-1}, F_{h-1,i} \notin \underline{SEQ}_h$.

Proof :

If $\hat{Si}_{h-1}$ is a candidate subsequence, then $\hat{Si}_{h-2}$, which is a subsequence of $\hat{Si}_{h-1}$, is a candidate subsequence. Now, $F_{h-1,i} \notin \underline{F}_{all,h-1}$. By corollary 3.9.2, $F_{h-2,i} = F_{h-1,i}$. Similarly, $\hat{Si}_{h-3}$, $\hat{Si}_{h-4}$ are candidate subsequences and their last mapping functions, $F_{h-3,i}$ and $F_{h-4,i}$ are equal to $F_{h-1,i}$.

Now, since $F_{h-1,i} \notin \underline{F}_{all,h} \bigcup \underline{F}_{all,h-1} \bigcup \underline{F}_{all,h-2} \bigcup \underline{F}_{all,h-3}$, $D(P_r F_{r,i}^{-1}) = 2$, $h-3 \leq r \leq h$. Let $\hat{Si}_h = \hat{Si}_{h-1},F_{h-1,i}$. Then,

$$D_{1,h}(\hat{Si}_h) = D_{1,h-4}(\hat{Si}_{h-4}) + \sum_{r=h-3}^{h} D(P_r F_{r,i})$$
$$= D_{1,h}(\hat{Si}_{h-4}) + 8.$$

Based on lemma 3.3 and theorem 3.4, we can always find another intermediate subsequence $\hat{Sj}_h = \hat{Si}_{h-4},F_{h-3,j},F_{h-2,j},F_{h-1,j},F_{h,j}$ such that $F_{h-3,j} = F_{h-2,j}$, $F_{h-1,j} = F_{h,j}$, $D(P_r F_{r,j}^{-1}) \leq 1$, $h-3 \leq r \leq h$, and the remapping cost $D(F_{h-3,j}F_{h-4,i}^{-1})$ and $D(F_{h-1,j}F_{h-2,j}^{-1})$ are less than or equal to one. Therefore, $D_{1,h}(\hat{Sj}_h) \leq D_{1,h-4}(S_{i_{h-1}}) + 6$.

Since $D_{1,h}(\hat{Si}_h) - D_{1,h}(\hat{Sj}_h) \geq 2$, by theorem 3.6, $\hat{Si}_h = \hat{Si}_{h-1},F_{h-1,i} \notin \underline{SEQ}$.

Corollary 3.10.1

For an optimal solution it is sufficient that, $\underline{F}_h = \underline{Map}_h \bigcup \underline{F}_{all,h}$.

Proof:

Since, if $\hat{Si}_{h-1}$, $F_{h-1,i} \in \underline{SEQ}_h$, then $F_{h-1,i} \in \underline{F}_{all,h} \bigcup \underline{F}_{all,h-1} \bigcup \underline{F}_{all,h-2} \bigcup \underline{F}_{all,h-3}$ and $F_{h-1,i} \in \underline{F}_{all,h} \bigcup \underline{last}_{h-1}$ (corollary 3.9.1), it is sufficient for

$$\begin{aligned}\underline{F}_h &= (\underline{F}_{all,h} \bigcup \underline{last}_{h-1}) \bigcap (\underline{F}_{all,h} \bigcup \underline{F}_{all,h-1} \bigcup \underline{F}_{all,h-2} \bigcup \underline{F}_{all,h-3}) \\ &= \underline{F}_{all,h} \bigcup (\underline{last}_{h-1} \bigcap (\underline{F}_{all,h-1} \bigcup \underline{F}_{all,h-2} \bigcup \underline{F}_{all,h-3})) \\ &= \underline{F}_{all,h} \bigcup \underline{Map}_h.\end{aligned}$$

□

Since, $\underline{F}_h \subseteq \bigcup_{t=h-3}^{h} \underline{F}_{all,t}$ the set of data mapping functions that the algorithm will be examining will not grow larger and larger, with increasing h, until it equals Δ. In fact, it is bound by $4 \times |\underline{F}_{all}|$, where $|\underline{F}_{all}| = 2^{n^2}$.

3.3. Non-Static Data Mapping Algorithms

From theorem 3.9, we can also conclude that if F_a is not in $\underline{F}_{all,h}$, but in $\underline{Map}_h$, then F_a has to be appended only to the candidate subsequence whose last mapping function is equal to F_a. Therefore, instead of adding each data mapping functions in $\underline{F}_{all,h} \bigcup \underline{Map}_h$ to every subsequence in $\underline{SEQ}_{h-1}$, we only have to add each data mapping functions in $\underline{F}_{all,h}$ to every subsequence in $\underline{SEQ}_{h-1}$ and add $F_{h-1,i}$ in $\hat{Si}_{h-1}$ to each subsequence $\hat{Si}_{h-1}$ in $\underline{SEQ}_{h-1}$. Hence, instead of producing at most $|\underline{SEQ}_{h-1}| \times |\underline{F}_{all,h} \bigcup \underline{Map}_h|$ intermediate subsequences at step h of the algorithm, we have to produce only at most $|\underline{SEQ}_{h-1}| \times |\underline{F}_{all,h}| + |\underline{Map}_h|$ intermediate subsequences.

Actually not all the $|\underline{SEQ}_{h-1}| \times |\underline{F}_{all,h}| + |\underline{Map}_h|$ intermediate subsequences have to be generated and evaluated. We will first define the following notation and then describe the observations behind such claim.

Notation 3.4 :

The upper triangular unit matrices are $\mathbf{U}_i$, $1 \leq i \leq 2^{\frac{(n^2-n)}{2}}$. $\mathbf{U}_1 = \mathbf{I}$. Similarly, the lower triangular unit matrices are $\mathbf{L}_i$, $1 \leq i \leq 2^{\frac{(n^2+n)}{2}}$. $\mathbf{L}_1 = \mathbf{I}$.

□

The following observations can be used in reducing the actual number of intermediate subsequences that have to be formed.

1. Let C be the cost of the lowest cost intermediate subsequence of length h formed thus far in the partial enumeration algorithm. Let $F_i \in \underline{F_h}$. If $Min(\underline{SEQ}^i_{h-1}) + D(P_h F_i^{-1}) > C + 1$, then $\underline{SEQ}^i_{h-1}, F_i \notin \underline{SEQ_h}$ (Theorem 3.6).

2. The lower bounds of the costs of intermediate subsequences formed by appending a mapping function, $F_a \in \underline{F_h}$, to subsequences in $\underline{SEQ}^i_{h-1}$, $i = 0,1$, are $l_0 = D(P_h F_a^{-1}) + Min(\underline{SEQ}^0_{h-1})$ and $l_1 = D(P_h F_a^{-1}) + Min(\underline{SEQ}^1_{h-1})$, respectively. Once an intermediate subsequence with cost, l_i is obtained by appending F_a to a subsequence in $\underline{SEQ_{h-1}}$, we do not have to append F_a to any subsequences in $\underline{SEQ}^i_{h-1}$ (Theorem 3.7). Furthermore, if such intermediate subsequence has a cost of l_0, we do not have to append F_a to any subsequences in $\underline{SEQ}^0_{h-1} \bigcup \underline{SEQ}^1_{h-1}$ (Theorem 3.7).

3. Let $F^{-1}_{j,h-1}$ be the last data mapping function of $\hat{Sj}_{h-1}$. If $\mathbf{F}_i \mathbf{F}^{-1}_{j,h-1} = \mathbf{U}_s$, where $\mathbf{F}_i \in \underline{F_{all,h}}$, but $\mathbf{F}_i \notin \underline{\mathbf{U}P_h}$, then we have to append $\mathbf{U}_s^{-1} F_i$ to $\hat{Sj}_{h-1}$ only. Appending any other $\mathbf{F}_r \in \underline{\mathbf{U}\mathbf{F}_i}$ to $\hat{Sj}_{h-1}$ is not necessary, for they will be eliminated (Theorem 3.8).

4. Let $F_{j,h-1}$ be the last data mapping function of $\hat{Sj}_{h-1}$. Let $\underline{S_{ULU}}$ be the set of all **ULU** decompositions of $F_i F^{-1}_{j,h-1}$, where $F_i \in \underline{F_h}$, but $\mathbf{F}_i \notin \underline{\mathbf{U}P_h}$. We have to form intermediate subsequence $\hat{Sj}_{h-1}, U_r F_i$ iff the **ULU** decomposition $\mathbf{U}_r \mathbf{L}_s \mathbf{U}_t \in \underline{S_{ULU}}$. Appending any other $U_w F_i$ to $\hat{Sj}_{h-1}$ is not necessary, for they will be eliminated (Theorem 3.8).

The algorithm described below incorporates the observations made and will produce an optimum sequence of data mapping functions.

Algorithm I :

MAIN PROGRAM
BEGIN
Let $\underline{SEQ}_1^0 = P_1$. Let $\underline{S}_1 = \underline{F}_{all,1}$ - $\underline{\mathbf{LP}}_1$.
Reduced $\underline{S}_1$ according to theorem 3.8.
FOR $h = 2$ TO k DO
BEGIN
Form $\underline{F}_{all,h}$. Partition it into $2^{\frac{(n^2+n)}{2}}$ equivalence classes.
They are the cosets $\underline{UL}_iP_h$, $1 \leq i \leq 2^{\frac{(n^2+n)}{2}}$.

$C = large\_number$; /* cost of minimum subsequence of length h */
$\underline{SEQ}_h^0 = \underline{SEQ}_h^1 = \underline{Temp} = \emptyset$;

/* Form intermediate subsequences, keep them in $Cand[1{:}2^{\frac{(n^2-n)}{2}}]$ and their corresponding cost in $Lmin[1{:}2^{\frac{(n^2-n)}{2}}]$ */

Form_Subsequence ($\underline{SEQ}_{h-1}^0$, $\underline{UL}_1P_h$, $Lmin[]$, $Cand[]$, C, h, 1);
Include $Cand[]$ into $\underline{Temp}$;

Form_Subsequence ($\underline{SEQ}_{h-1}^1$, $\underline{UL}_1P_h$, $Lmin[]$, $Cand[]$, C, h, 1);
Include $Cand[]$ into $\underline{Temp}$;

FOR j=2 TO $2^{\frac{(n^2+n)}{2}}$ DO
BEGIN
Form_Subsequence ($\underline{SEQ}_{h-1}^0$, $\underline{UL}_jP_h$, $Lmin[]$, $Cand[]$, C, h, j);
Include $Cand[]$ into $\underline{Temp}$;
END;

j=2;
DONE = FALSE;
WHILE (NOT ((min($\underline{SEQ}_{h-1}^1$) + 1) > C+1) AND NOT DONE DO (observation 1)
BEGIN

```
    Form_Subsequence (SEQ^1_{h-1}, UL_jP_h, Lmin[], Cand[], C, h, 1);
    Include Cand[] into Temp;
    j=j+1;
                    (n^2+n)
    IF (j > 2      2       ) THEN DONE = TRUE;
  END;

  /* Remove some intermediate subsequences and partition the set into two sets */
  REDUCE (Temp, SEQ^0_h, SEQ^1_h, h);

  /* Append subsequence with its last data mapping function if needed */
  IF { (F_h - F_all,h) ≠ ∅ } AND NOT(Min(SEQ^0_{h-1}) + 2 > C+1) THEN ( observation 1 )
  BEGIN
    FOR EVERY Ŝi_{h-1} ∈ SEQ^0_{h-1} and F_{h-1,i} ∈ F_h - F_all,h APPEND
        F_{h-1,i} to Ŝi_{h-1} to form Ŝi_h.
    IF D_{1,h}(Ŝi_h) = C, THEN
      Include it in SEQ^0_h
    ELSE
      IF (D_{1,h}(Ŝi_h) = C+1) AND ( it cannot be eliminated by theorem 3.8 )) THEN
        Include it in SEQ^1_h;
  END;

  /* Append subsequence with its last data mapping function if needed */
  IF { (F_h - F_all,h) ≠ ∅ } AND NOT(Min(SEQ^1_{h-1})+2 > C+1) THEN ( observation 1 )
  BEGIN
    FOR EVERY Ŝi_{h-1} ∈ SEQ^1_{h-1} and F_{h-1,i} ∈ F_h - F_all,h APPEND
        F_{h-1,i} to Ŝi_{h-1} to form Ŝi_h.
    IF D_{1,h}(Ŝi_h) = C, THEN
      Include it in SEQ^0_h
    ELSE
      IF (D_{1,h}(Ŝi_h) = C+1) AND ( it cannot be eliminated by theorem 3.8 )) THEN
        Include it in SEQ^1_h;
    END;
    ELIMINATE subsequences in SEQ^1_h according to theorem 8;
  END;
END.

/* This procedure forms intermediate subsequence if necessary */
PROCEDURE Form_Subsequence (SET, MPS, Lmin[], Cand[], C, h, t)
```

BEGIN

$Lmin[1:2^{\frac{(n^2-n)}{2}}]=large\_number;$

WHILE { (NOT every $\hat{S}_{h-1} \in \underline{SET}$ has been used) AND

$(Lmin[j] > Min(\underline{SET}) + D(P_h P_{h-1}^{-1} L_t^{-1})$, for some $1 \leq j \leq 2^{\frac{(n^2-n)}{2}}$)} DO (observation 2)

BEGIN

Let $\hat{S}_{h-1}$ be a sequence in $\underline{SET}$ that has not been used.

Let F be the last data mapping of $\hat{S}_{h-1}$. Let $F_0 = L_t P_h$.

IF $t = 0$ THEN

BEGIN

IF $(Lmin[0] > Min(\underline{SET}) + D(F_0 F^{-1})$ THEN

BEGIN

$Lmin[0]=Min(\underline{SET}) + D(F_0 F^{-1})$

$Cand[0] = \hat{S}_{h-1}$ appended with F_0;

C= minimum of C and $Lmin[0]$;

END;

END;

IF $(\mathbf{F_0 F^{-1}}) = \mathbf{U}_j \in \underline{\mathbf{U}}$ THEN (observation 3)

BEGIN

IF $(Lmin[j] > Min(\underline{SET}) + D(P_h F_0^{-1} U_j^{-1}))$ THEN (observation 2)

BEGIN

$Lmin[j]=Min(\underline{SET}) + D(P_h F_0^{-1} U_j^{-1})$;

$Cand[j] = \hat{S}_{h-1}$ appended with $U_j F_0$;

C= minimum of C and $Lmin[j]$;

END;

END

ELSE

BEGIN

IF $NOT(Min(\underline{SET}) + D(P_h F_0^{-1})+1 \geq C+1)$ THEN (observation 1)

BEGIN

FOR j=1 TO $2^{\frac{(n^2-n)}{2}}$ DO (observation 3)

BEGIN

FORM $\underline{S}_{ULU}$, the set of all **ULU** decomposition of $F_0 F^{-1}$; (observation 4)

FOR every $U_j L_r U_t \in \underline{S}_{ULU}$ DO

BEGIN

IF $(Lmin[j] > Min(\underline{SET}) + D(P_h F_0^{-1} U_j^{-1}) + 1$ THEN

BEGIN

$Lmin[j] = Min(\underline{SET}) + D(P_h F_0^{-1} U_j^{-1}) + 1$

$Cand[j]$=$\hat{S}$ appended with $U_j F_0$;

C= minimum of C and $Lmin[j]$;

END;

END;

END;

END;

Let $\hat{S}$ be another sequence in $\underline{SET}$, that has not been worked on yet.

END;

END.

/* This procedure include intermediate subsequences into the set of Candidate subsequences */

PROCEDURE REDUCE ($\underline{T}$, $\underline{SET}^0$, $\underline{SET}^1$, h)

BEGIN

Let $C = Min(\underline{T})$;

FOR every $\hat{S} \in \underline{T}$ DO

BEGIN

IF $D_{1,h}(\hat{S}) = C$ THEN

put $\hat{S}$ into $\underline{SET}_0$

ELSE

IF $D_{1,h}(\hat{S}) = C+1$ THEN

put $\hat{S}$ into $\underline{SET}^1$ if they cannot be eliminated by theorem 8.

END;

END.

□

Based on previous theorems, it is clear that the algorithm produces an optimum sequence of data mapping functions. Clearly, the complexity of the algorithm is dependent on the number of intermediate subsequence formed. For each $F_a \in \underline{F}_{all,h}$, at most $|\underline{SEQ}_{h-1}|$ intermediate subsequences with F_a as its last mapping function will be formed. But, based on theorem 3.7, there could only be one candidate subsequence with F_a as its last data mapping function. Therefore, the maximum number of candidate subsequences formed at step h is $|\underline{Map}_h + \underline{F}_{all,h}|$. But

$$|\underline{Map}_h| = |\underline{last}_{h-1} \cap (\underline{F}_{all,h-1} \cup \underline{F}_{all,h-2} \cup \underline{F}_{all,h-3})| \leq 3C,$$

where $C = 2^{n^2}$.

Therefore, $|\underline{Map}_h + \underline{F}_{all,h}| \leq 4C$. At step $h+1$, there are at most $|\underline{Map}_h + \underline{F}_{all,h}| \times |\underline{F}_{all,h+1}| + |\underline{Map}_h| \leq 4C^2 + 3C$ intermediate subsequences. Given k transfers for vector v, we will have $kC(4C + 3)$ intermediate subsequences. Since n remains constant for a fixed network, the complexity of the algorithm is $O(k)$.

A Heuristic Algorithm

The problem of determining optimum data mapping functions for a parallel program where precedence relationship between its vector variables exists is complex. For such parallel programs, a heuristic algorithm that is based on the optimum algorithm developed in this chapter has been developed and is described.

Consider a parallel program consisting of two expression trees only (figure 3.2). Clearly, expression tree #2 in the figure is dependent on results from expression tree #1. To determine optimum sequences of data mapping functions for C or D before determining data mapping functions for A, initial data mapping function for A has to be assumed. However, the data mapping function assumed for A may not be the initial data mapping function of an optimum sequence for A. Hence, an optimum sequence of data mapping functions for A shall be determined before optimum sequences for C and D are determined. More often than not, a set of optimum sequences for A will be produced by algorithm I. Thus, for the parallel program the problem of determining which one will lead to an optimal solution, if it ever will, is complex. Hence, a heuristic algorithm for finding a near-optimal solution has been developed.

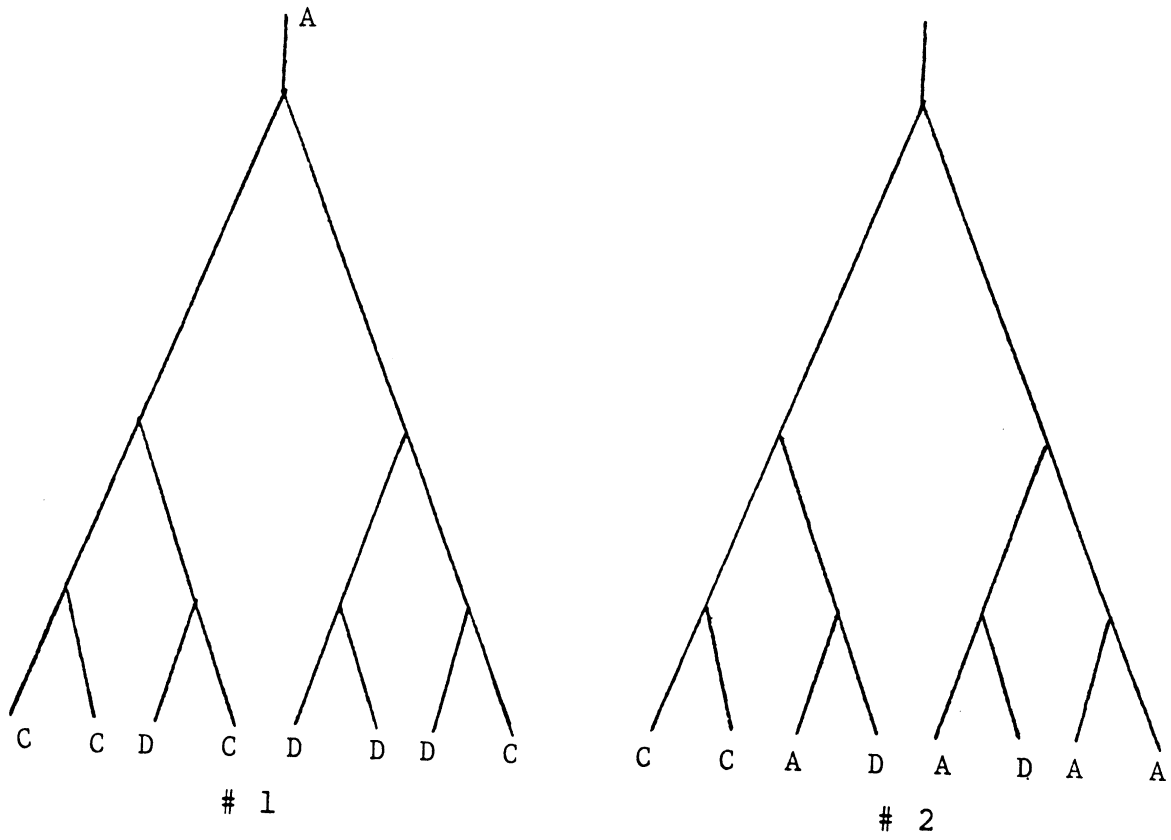


Figure 3.2 Expression trees of a parallel program.

Algorithm II

1. If the result vector, v, of an expression tree is dependent on its previous value, assign a new vector variable to replace the result vector. Note : for each iteration of a program loop where such dependency exists, we have to assign a new vector variable to replace each such result vector.

2. Determine the precedence relationships between the vector variables and put them in a sequence, $v_1, v_2, \ldots, v_z$, such that logical transfers for v_j is not dependent on any variables v_i, $i > j$.

3. For j=1 to z DO

BEGIN

Determine optimum sequences for v_j.

Pick *limit* number of optimum sequences for v_j and for each of them

determine optimum sequences for other variables as follows :

For i=j+1 TO z DO

BEGIN

Determine logical transfers for v_i.

Determine optimum sequences for v_i.

Arbitrarily pick an optimum sequence for v_i.

END;

Keep the optimum sequence for v_j that leads to the total lowest cost

for all variables.

If $j \neq z$, based on data mapping functions for v_j determine logical transfers for v_{j+1}.

END;

□

The above heuristic has been tested on a number of example problems, its performance has been compared with that of a total enumeration algorithm and it is found that in most cases, the heuristic generates optimal or near-optimal solutions.

3.4. Examples

In this section, we will consider two parallel algorithms, the inversion of a lower triangular matrix and the radix-2 fast Fourier transform (FFT). These two examples are representative of two classes of diverse, important computational requirements, namely, matrix operations, discrete spectrum analysis and related processes.

In section 3.4.1, we will present a parallel algorithm for lower triangular matrix inversion and a corresponding optimum solution. In section 3.4.2, we will present the parallel FFT algorithm and a corresponding optimum solution.

3.4.1. Lower Triangular Matrix Inversion

In this section, we will consider the inversion of an 8×8 lower triangular matrix on an SIMD multicomputer system with 16 processing elements. We will assume that in any operation, one or more processing elements can be disabled or set to produce zeros. We will designate a vector variable of 16 elements by a capital letter. We will also designate the *i*th element of a vector by its corresponding small letter and a subscript *i*.

The inversion algorithm uses partitioning[Boro75,Hell78] to invert a $r \times r$ lower triangular matrix, **V**. It is based on the following relationship between **V** and its inverse $\mathbf{V}^{-1}$. If

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & 0 \\ \mathbf{V}_3 & \mathbf{V}_2 \end{bmatrix},$$

where $\mathbf{V}_1$, $\mathbf{V}_2$ are $\frac{r}{2} \times \frac{r}{2}$ lower triangular matrices and $\mathbf{V}_3$ is a $\frac{r}{2} \times \frac{r}{2}$ matrix, then

$$\mathbf{V}^{-1} = \begin{bmatrix} \mathbf{V}_1^{-1} & 0 \\ -\mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1} & \mathbf{V}_2^{-1} \end{bmatrix}.$$

The inversion algorithm hence proceeds in stages. It will first determine $\mathbf{V}_1^{-1}$ and $\mathbf{V}_2^{-1}$, and then $\mathbf{V}_2^{-1}\mathbf{V}_3$ and $\mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1}$.

In our example, the elements of **V** are stored in three vectors, *A*, *D*, and *L*, and they are to be operated on in that order. Their relationship with **V** is as depicted in figure 3.3.

$$\mathbf{V} = \begin{bmatrix} a_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a_2 & a_3 & 0 & 0 & 0 & 0 & 0 & 0 \\ d_8 & d_9 & a_{12} & 0 & 0 & 0 & 0 & 0 \\ d_{10} & d_{11} & a_{14} & a_{15} & 0 & 0 & 0 & 0 \\ l_0 & l_1 & l_4 & l_5 & a_8 & 0 & 0 & 0 \\ l_2 & l_3 & l_6 & l_7 & a_{10} & a_{11} & 0 & 0 \\ l_8 & l_9 & l_{12} & l_{13} & d_0 & d_1 & a_4 & 0 \\ l_{10} & l_{11} & l_{14} & l_{15} & d_2 & d_3 & a_6 & a_7 \end{bmatrix}.$$

Vector elements that are not shown are zeroes, e.g. $d_{12} = 0$.

Figure 3.3 Relationship between **V**, **A**, **D** and **L**.

For convenience, we name the submatrices of the matrix, **V**, as in figure 3.4.

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_1 & 0 \\ \mathbf{V}_3 & \mathbf{V}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_0 & 0 & 0 & 0 \\ \mathbf{D}_2 & \mathbf{A}_1 & 0 & 0 \\ \mathbf{L}_0 & \mathbf{L}_1 & \mathbf{A}_2 & 0 \\ \mathbf{L}_2 & \mathbf{L}_3 & \mathbf{D}_0 & \mathbf{A}_3 \end{bmatrix}$$

$\mathbf{A}_i'$ s, $\mathbf{D}_i'$ s & $\mathbf{L}_i'$ s are 2×2 matrices.

Figure 3.4 Relationship between **V** and its submatrices.

Some more definitions are needed.

Definition 3.2 :

Let # denote an exchange operator. Let Y and Z be two vector variables. Then Y # Z means that if the ith processing element is enabled, then the ith element of vectors Y and Z are swapped.

□

Definition 3.3 :

P_a is a permutation defined by

$$P_a:(x_3,x_2,x_1,x_0)\rightarrow(x_3,x_2,x_0,x_1).$$

□

Definition 3.4 :

P_b is a permutation defined by

$$P_b:(x_3,x_2,x_1,x_0)\rightarrow(x_2,x_3,x_0,x_1).$$

□

We shall first describe how the inverses of $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{A}_3$ are formed and how they are stored in a new vector C. The relationship between the elements of C, $c_i, 0 \leq i \leq 15$, and the submatrices is as shown below

$$\mathbf{A}_0^{-1} = \begin{bmatrix} c_0 & 0 \\ c_2 & c_3 \end{bmatrix}$$

$$\mathbf{A}_1^{-1} = \begin{bmatrix} c_{12} & 0 \\ c_{14} & c_{15} \end{bmatrix}$$

$$\mathbf{A}_2^{-1} = \begin{bmatrix} c_8 & 0 \\ c_{10} & c_{11} \end{bmatrix}$$

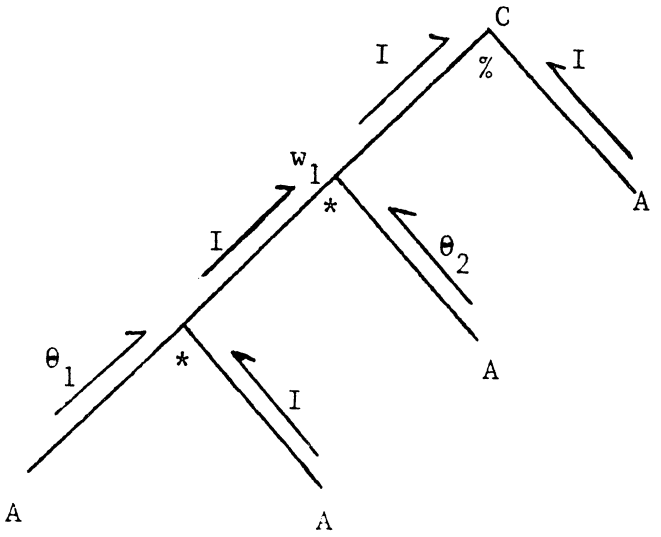
$$\mathbf{A}_3^{-1} = \begin{bmatrix} c_4 & 0 \\ c_6 & c_7 \end{bmatrix}$$

Figure 3.5 Relationship between $\mathbf{A}_i'$ s and C.

The following is a description of the procedure for forming the inverses of the 2×2 matrices.

(1) The reciprocals of the diagonal terms of the matrices are formed and stored in their original location in A. (They are equal to c_0, c_3, c_{12}, c_{15}, c_8, c_{11}, c_4, c_7).

(2) The off-diagonal terms, a_2, a_{14}, a_{10} and a_6, of the matrices are multiplied with the diagonal terms of their inverses (figure 3.6), for example, $c_2 = -c_3 a_2 c_0$. The products are then combined with the diagonal terms of their inverses, $\mathbf{A}_0^{-1}, \mathbf{A}_1^{-1}, \mathbf{A}_2^{-1}, \mathbf{A}_3^{-1}$ to give a new vector C.



Only processor elements with off-diagonal terms multiply.
% - diagonal and off-diagonal terms of the inverses combined to form C.

Figure 3.6 Forming the off-diagonal terms of $\mathbf{A}_i^{-1}$ s.

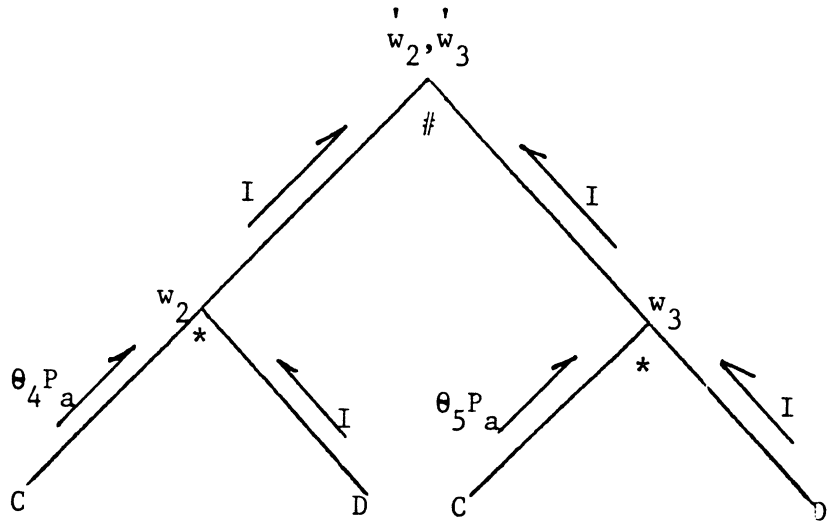
Having the inverses of $\mathbf{A}_0$, $\mathbf{A}_1$, $\mathbf{A}_2$ and $\mathbf{A}_3$ formed, we can now proceed to compute $\mathbf{A}_1^{-1}\mathbf{D}_5$ and $\mathbf{A}_3^{-1}\mathbf{D}_0$ to give a new vector $\mathbf{E}$, where

$$\mathbf{A}_3^{-1}\mathbf{D}_0 = \begin{bmatrix} e_0 & e_1 \\ e_2 & e_3 \end{bmatrix},$$

and

$$\mathbf{A}_1^{-1}\mathbf{D}_2 = \begin{bmatrix} e_8 & e_9 \\ e_{10} & e_{11} \end{bmatrix},$$

as in figure 3.7.



$\theta_4 \cdot P_a$ - forms all product terms of the diagonal elements.
$\theta_5 \cdot P_a$ - forms all the product terms of the off-diagonal elements.
- interchange of the vector elements in processors 1, 2, 9 and 10.

Figure 3.7 Forming all the product terms of $\mathbf{A}_1^{-1}\mathbf{D}_5$ and $\mathbf{A}_3^{-1}\mathbf{D}_0$.

The elements of $\mathbf{A}_3^{-1}\mathbf{D}_0$ and $\mathbf{A}_1^{-1}\mathbf{D}_5$ are then formed by adding $\dot{w}_2$ and $\dot{w}_3$ together as the expression tree in figure 3.8.

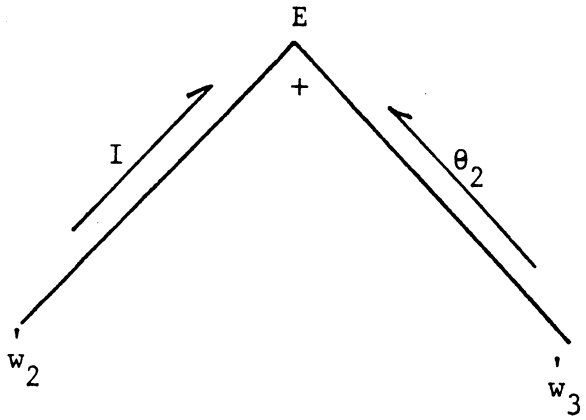
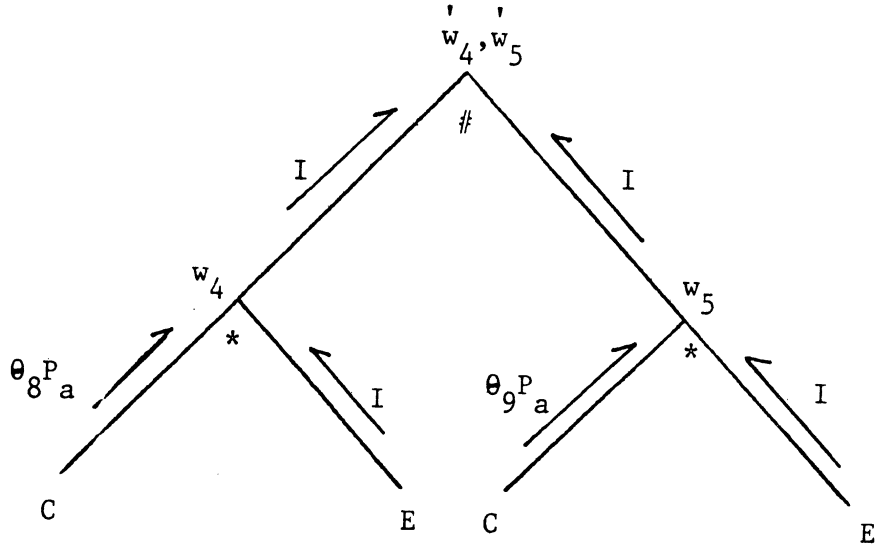


Figure 3.8 Adding product terms to form elements of $\mathbf{A}_3^{-1}\mathbf{D}_0$ and $\mathbf{A}_1^{-1}\mathbf{D}_5$.

$\mathbf{A}_1^{-1}\mathbf{D}_5\mathbf{A}_0^{-1}$ and $\mathbf{A}_3^{-1}\mathbf{D}_0\mathbf{A}_2^{-1}$ can similarly be formed with the expression trees in figure 3.9 and figure 3.10.



$\theta_8 \cdot P_a$ - forms all product terms of the diagonal elements.
$\theta_9 \cdot P_a$ - forms all product terms of the off-diagonal elements.
- interchange of vector elements in processors 1,2,9 and 10.

Figure 3.9 Forming all the product terms of $\mathbf{A}_1^{-1}\mathbf{D}_5\mathbf{A}_0^{-1}$ and $\mathbf{A}_3^{-1}\mathbf{D}_0\mathbf{A}_2^{-1}$.

The elements of $\mathbf{A}_3^{-1}\mathbf{D}_0\mathbf{A}_2^{-1}$ and $\mathbf{A}_1^{-1}\mathbf{D}_5\mathbf{A}_0^{-1}$ are formed by adding the product terms $\dot{w}_4$ and $\dot{w}_5$ together as in figure 3.10 to give F.

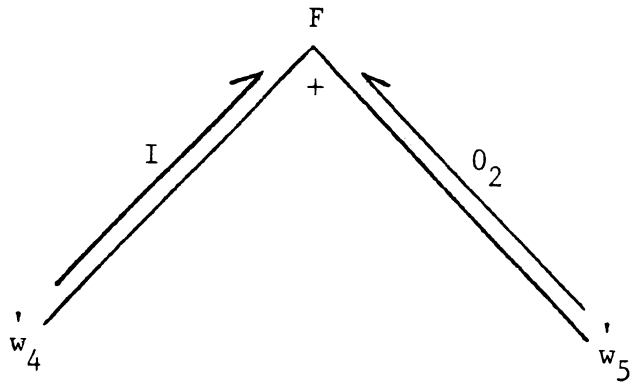
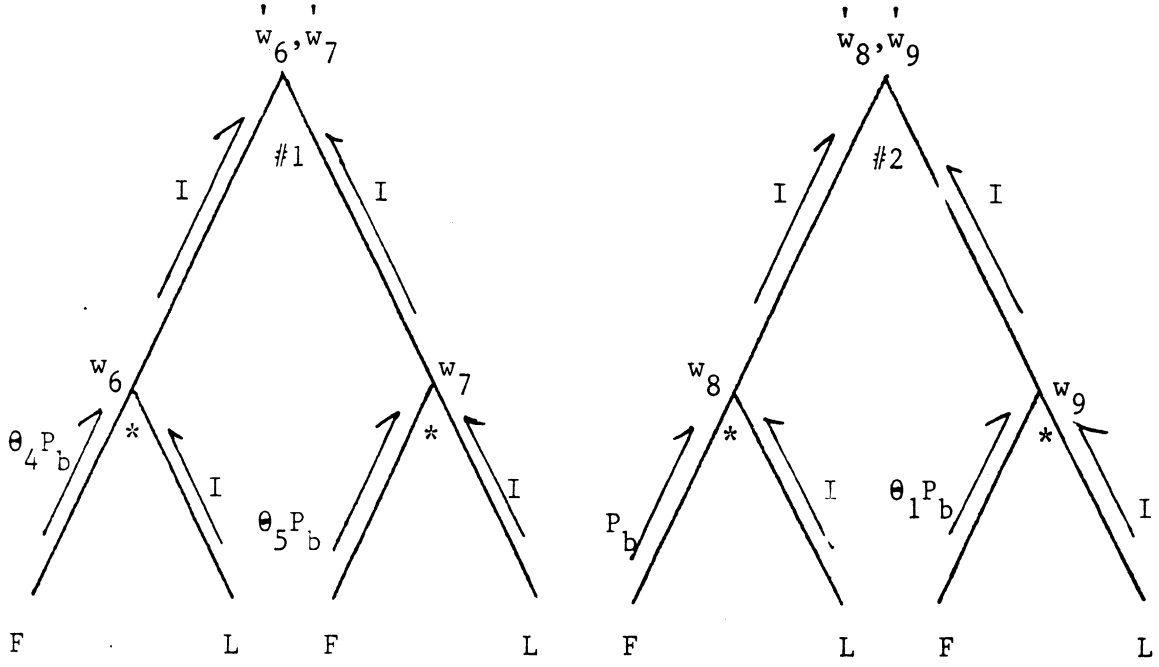


Figure 3.10 Add ng prodi ct terms to form elements of $\mathbf{A}_3^{-1}\mathbf{D}_0\mathbf{A}_2^{-1}$ and $\mathbf{A}_1^{-1}\mathbf{D}_5\mathbf{A}_0^{-1}$.

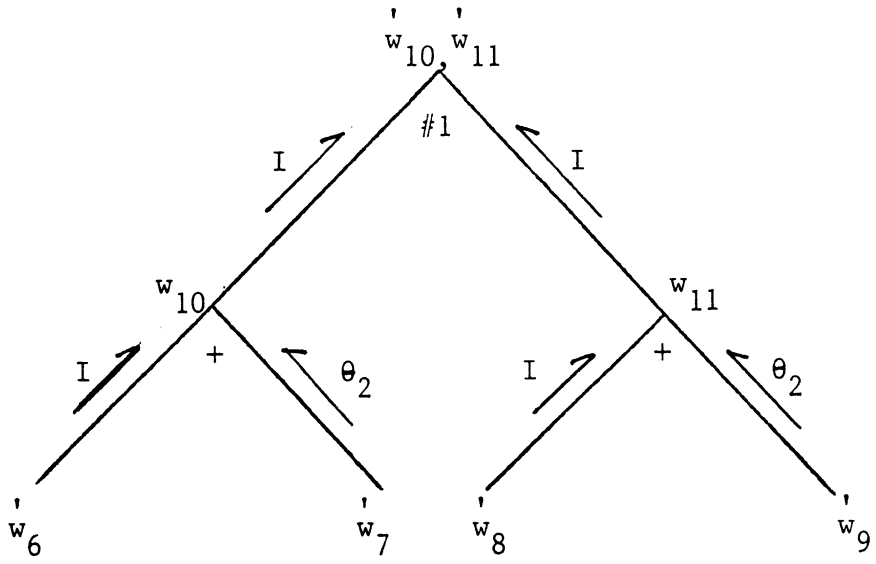
Having V_1^{-1} and V_2^{-1} computed, we can now form $V_2^{-1}V_3V_1^{-1}$. The product terms for all the elements in $U = V_2^{-1}V_3$ are computed according to the two expression trees in figure 3.11.



$\theta_4 \cdot P_b$ - forms all product terms for the diagonal elements of **U**.
$\theta_5 \cdot P_b$ - forms all product terms for $u_{1,0}, u_{0,1}, u_{2,3}$ and $u_{3,2}$ of **U**.
P_b - forms all product terms for $u_{2,0}, u_{0,2}, u_{1,3}$ and $u_{3,1}$ of **U**.
$\theta_1 \cdot P_b$ - forms all product terms for $u_{3,0}, u_{0,3}, u_{2,1}$ and $u_{1,2}$ of **U**.
#1 & #2 - interchange of vector elements in processors 1, 2, 5, 6, 9, 10, 13 and 14.

Figure 3.11 Forming the product terms of $V_2^{-1}V_3$.

We now have at most four product terms for each element in **U**. The expression tree in figure 3.12 adds $\dot{u}_6$ to $\dot{u}_7$ and $\dot{u}_8$ to $\dot{u}_9$ to form two sums of product terms for each element in **U**.



#1 - interchange of vector elements in processors
0, 1, 2, 3, 12, 13, 14 and 15.

Figure 3.12 Forming intermediate sums of product terms.

Each element in **U** is finally formed by adding its own corresponding two intermediate sums of product terms according to the expression tree in figure 3.13.

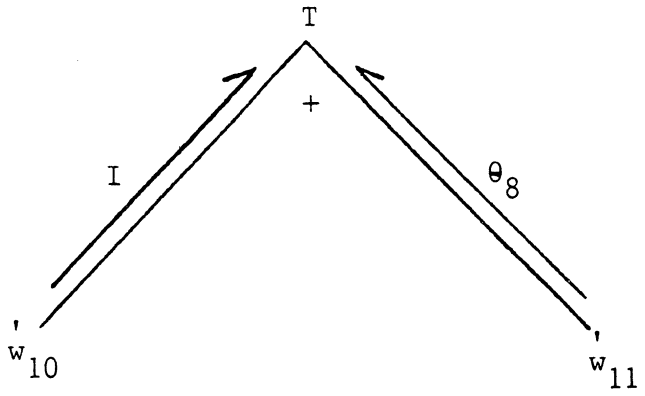
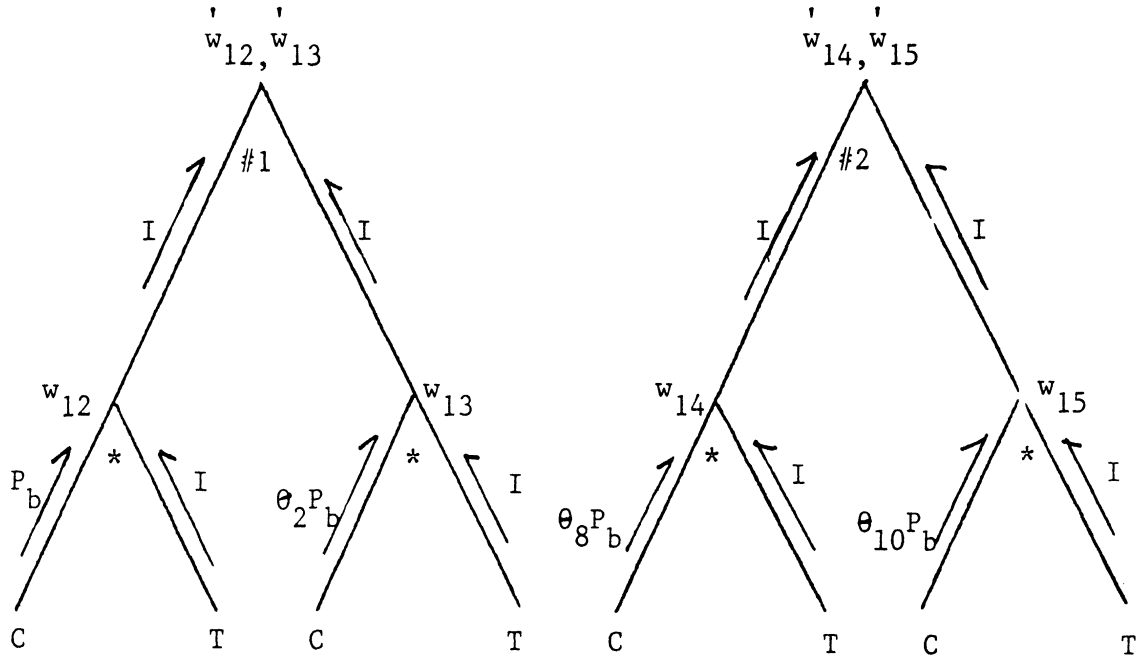


Figure 3.13 Forming the elements of $\mathbf{V}_2^{-1}\mathbf{V}_3$.

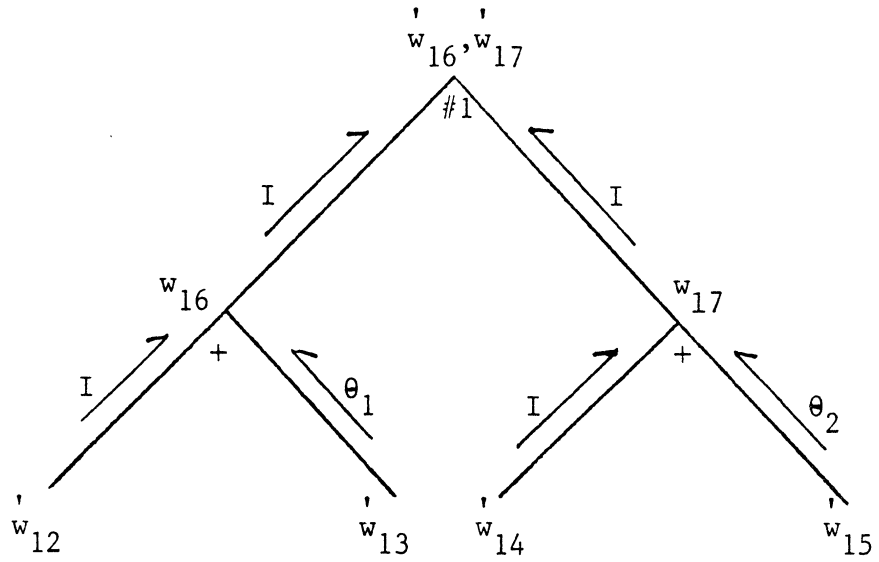
Having $\mathbf{U} = \mathbf{V}_2^{-1}\mathbf{V}_3$ formed, we can now proceed to compute all the product terms of the elements in $\mathbf{H} = \mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1}$ according to the two expression trees in figure 3.14.



P_b - forms all product terms for the diagonal elements of **H**.
$\theta_2 \cdot P_b$ - forms all product terms for $h_{0,1}$, $h_{1,0}$, $h_{2,3}$ and $h_{2,3}$.
$\theta_8 \cdot P_b$ - forms all product terms for $h_{0,2}$, $h_{2,0}$, $h_{1,3}$ and $h_{3,1}$.
$\theta_{10} \cdot P_b$ - forms all product terms for $h_{0,3}$, $h_{3,0}$, $h_{1,2}$ and $h_{2,1}$.
#1 & #2 - interchange of the vector elements in processors 1, 2, 5, 6, 9, 10, 13 and 14.

Figure 3.14 Forming all the product terms of $\mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1}$.

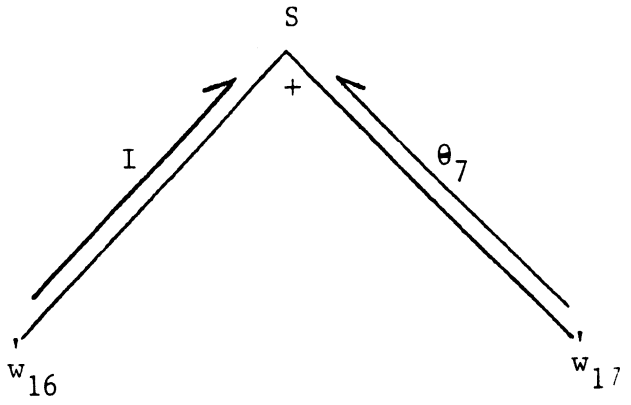
We have now at most four product terms for each element in **H**. The expression tree in figure 3.15 adds $\dot{w}_{12}$ to $\dot{w}_{13}$ and $\dot{w}_{14}$ to $\dot{w}_{15}$ to form two sums of product terms for each element in **H**.



#1 - interchange of vector elements in processors
4, 5, 6, 7, 8, 9, 10 and 11.

Figure 3.15 Forming intermediate sum of product terms.

Each element in **H** is finally formed by adding its own corresponding two intermediate sums of product terms according to the expression tree in figure 3.16.



$$\mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1} = \begin{bmatrix} s_0 & s_1 & s_4 & s_5 \\ s_2 & s_3 & s_6 & s_7 \\ s_8 & s_9 & s_{12} & s_{13} \\ s_{10} & s_{11} & s_{14} & s_{15} \end{bmatrix}.$$

Figure 3.16 Forming the elements of $\mathbf{H} = \mathbf{V}_2^{-1}\mathbf{V}_3\mathbf{V}_1^{-1}$.

A total enumeration algorithm and the heuristic algorithm have been used to determine data mapping functions for the above matrix inversion algorithm. It is found that the heuristic algorithm produced an optimum solution in this case.

In the heuristic algorithm, the ordering of the vector variables into a sequence is not unique. The following is one such sequence,

$$S, w_{17}, w_{16}, w_{15}, w_{14}, w_{13}, w_{12}, T, w_{11}, w_{10}, w_9, w_8, F, L, w_4, w_5, E, w_3, w_2, C, D, A.$$

The total communication cost before applying the heuristic algorithm is 34 routing steps. Applying the heuristic algorithm, the mapping functions for all the variables, except C and F, are determined to be identity. The sequence of communication requirements for F is determined to be $\theta_4 P_b$, $\theta_5 \cdot P_b$, $\theta_0 \cdot P_b$, P_b. The corresponding sequence of optimum data mapping functions for F is found to be P_b, P_b, P_b, P_b. Due to the initial data mapping functions, P_b, for F, the sequence of communication patterns for C is $P_b \cdot \theta_4 \cdot P_a$, $P_b \cdot \theta_5 \cdot P_a$, $P_b \cdot \theta_8 \cdot P_a$, $P_b \cdot \theta_9 \cdot P_a$, P_b, $\theta_2 \cdot P_b$, $\theta_8 \cdot P_b$, $\theta_{10} \cdot P_b$. The corresponding sequence of optimum data mapping

functions for C is determined to be $P_b \cdot \theta_5 \cdot P_a$, $P_b \cdot \theta_5 \cdot P_a$, $P_b \cdot \theta_5 \cdot P_a$, $P_b \cdot \theta_5 \cdot P_a$, P_b, P_b, P_b, P_b. The resulting total communication cost of the inversion algorithm is 21 routing steps. For this particular case, it is also the minimum communication cost possible.

3.4.2. Fast Fourier Transform

The radix-2 version of the Fast Fourier Transform (FFT) that we are going to describe is based on Pease's work[Peas68]. It has a signal flow diagram as shown in figure 3.17 and is dependent on perfect shuffle and bit reversal permutation.

We will first describe the discrete Fourier transform. Let $A_0 [k]$, $k = 0, 1,..., R-1$, be $R = 2^r$ samples of a time function sampled at instants that are spaced equally apart. The discrete Fourier transform of $A_0 [k]$ is then a discrete function $X [j]$, $j = 0, 1,..., R-1$, where

$$X [j] = \sum_{k=0}^{R-1} A_0 [k] W^{jk}, \qquad j = 0, 1,..., R-1$$

$$\text{and } W = e^{\frac{2\Pi i}{R}}.$$

Let $(k_{r-1}, k_{r-2}, \ldots, k_0)$ and $(j_{r-1}, j_{r-1}, \ldots, j_0)$ be the binary representations of $k, j \in \{ 0, 1, 2,..., R-1 \}$. Then

$$X [j_{r-1}, j_{r-2}, \ldots, j_0] = \sum_{k_0} \sum_{k_1} \cdots \sum_{k_{r-1}} A_0 [k_{r-1}, k_{r-2}, \ldots, k_0] W^{jk},$$

where each of the indices k_i are summed over the binary values 0 and 1.

Based on the above relationship, the FFT algorithm can be computed in r stages. At stage i, $1 \leq i \leq n$, a vector A_i will be computed from A_{i-1}, for $i \geq 1$. The vectors are defined as follows.

$$A_1 [k_{r-2}, \ldots, k_0, j_0] = \sum_{k_{r-1}} A_0 [k_{r-1}, k_{r-2}, \ldots, k_0] W^{j_0 k_{r-1} 2^{r-1}},$$

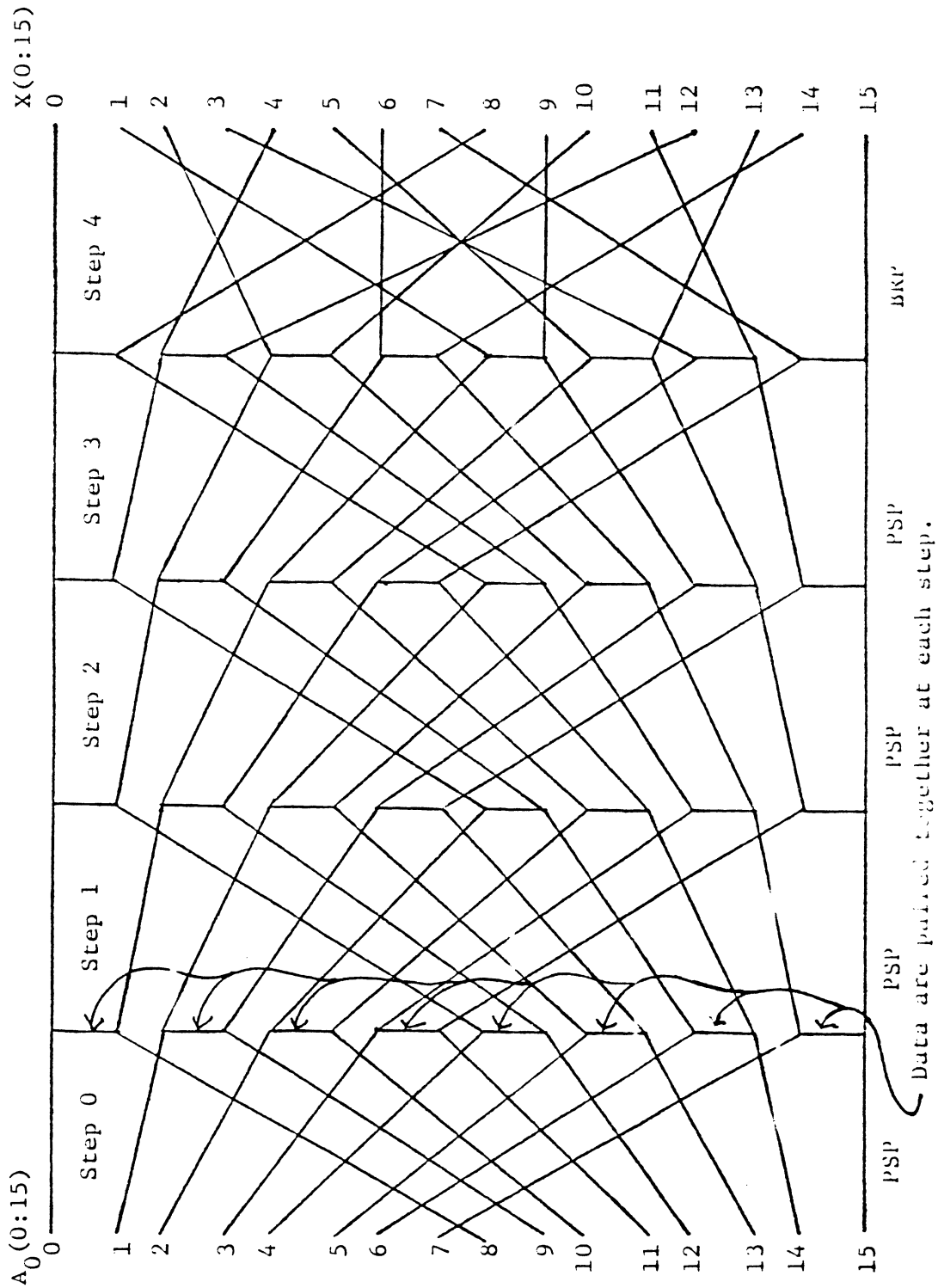


Figure 3.17 Signal flow graph of a 16 points FFT.

$$A_t[\, k_{r-t-1}, \ldots, k_0, j_0, \ldots, j_{t-1}\,] = \sum_{k_{r-t}} A_{t-1}[\, k_{r-t}, \ldots, k_0, j_0, \ldots, j_{t-2}\,] W^{(j_{t-1}2^{t-1} + \cdots + j_0)k_{r-t}2^{r-t}},$$

where $t = 2, 3, \ldots, r$.

The roles of perfect shuffle and bit reversal permutation in the signal flow graph (figure 3.17) become evident when we consider an element of A_i. Here, $A_i[\, j_{r-1}, j_{r-2}, \ldots, 0\,]$ and $A_i[\, j_{r-1}, j_{r-2}, \ldots, 1\,]$ of A_i is the weighted sum of two elements, $A_{i-1}[\, 0, j_{r-2}, \ldots, j_1\,]$ and $A_{i-1}[\, 1, j_{r-2}, \ldots, 1\,]$, of A_{i-1}. A perfect shuffle will pair the two elements together for the computation of $A_i[\, j_{r-1}, j_{r-2}, \ldots, 0\,]$ and $A_i[\, j_{r-1}, j_{r-2}, \ldots, 0\,]$. The last vector, A_r, has elements that are the values of the $X[\, j\,]$, but the elements are scrambled in bit reversal order. That is,

$$A_r[\, j_0, j_1, \ldots, j_{r-2}, j_{r-1}\,] = X[\, j_{r-1}, j_{r-2}, \ldots, j_0\,].$$

A bit reversal permutation will bring the elements of X back into normal order.

Since the computations required are executed on pairs of data, the elements of the vectors can be stored in pairs and a 16 points FFT can be executed on an eight processing elements system. Each vector, $A_i, 0 \leq i \leq 4$, can be divided into two vectors, such that

$$\dot{A}_i[\, 0{:}7\,] = A_i[\, 0,2,4,6,9,11,13,15\,]$$

and

$$\ddot{A}_i[\, 0{:}7\,] = A_i[\, 1,3,5,7,8,10,12,14\,].$$

For the ith computation stage, each element in $\dot{A}_i$ has to be paired with an element in $\ddot{A}_i$. The logical transfers for $\dot{A}_i$ and $\ddot{A}_i$, $0 \leq i \leq 3$, are determined to be

$$\dot{P}: (\, x_2, x_1, x_0\,) \rightarrow (\, x_1, x_0, x_2\,)$$

and

$$\ddot{P}: (\, x_2, x_1, x_0\,) \rightarrow (\, x_1, x_0, \bar{x}_2\,, \text{ respectively.}$$

Finally, since

$$\dot{A}_4 [\, 0{:}7 \,] = X[\, 0,4,2,6,9,13,11,15 \,]$$

and

$$\ddot{A}_4 [\, 0{:}7 \,] = X[\, 8,12,10,14,1,5,3,7 \,],$$

to bring X into normal order such that

$$\dot{X}[\, 0{:}7 \,] = X[\, 0,2,4,6,9,11,13,15 \,]$$

and

$$\ddot{X}[\, 0{:}7 \,] = X[\, 1,3,5,7,8,10,12,14 \,],$$

the logical transfers for $\dot{X}$ and $\ddot{X}$ are determined to be

$$\dot{P}_r : (\, x_2,x_1,x_0 \,) \rightarrow (\, x_2,x_0,x_1 \,)$$

and

$$\ddot{P}_r : (\, x_2,x_1,x_0 \,) \rightarrow (\, \bar{x}_2,x_1,x_0 \,), \text{ respectively.}$$

Let R_i, S_i, T_i & U_i, $0 \leq i \leq 3$, be vectors containing powers of W. For example, $S_0 = [\, 1, 1, 1, 1, W^8, W^8, W^8, W^8 \,]$. Then the expression trees for stage 1,2,3 and 4 are shown in the figures 3.18, 3.19 and 3.20.

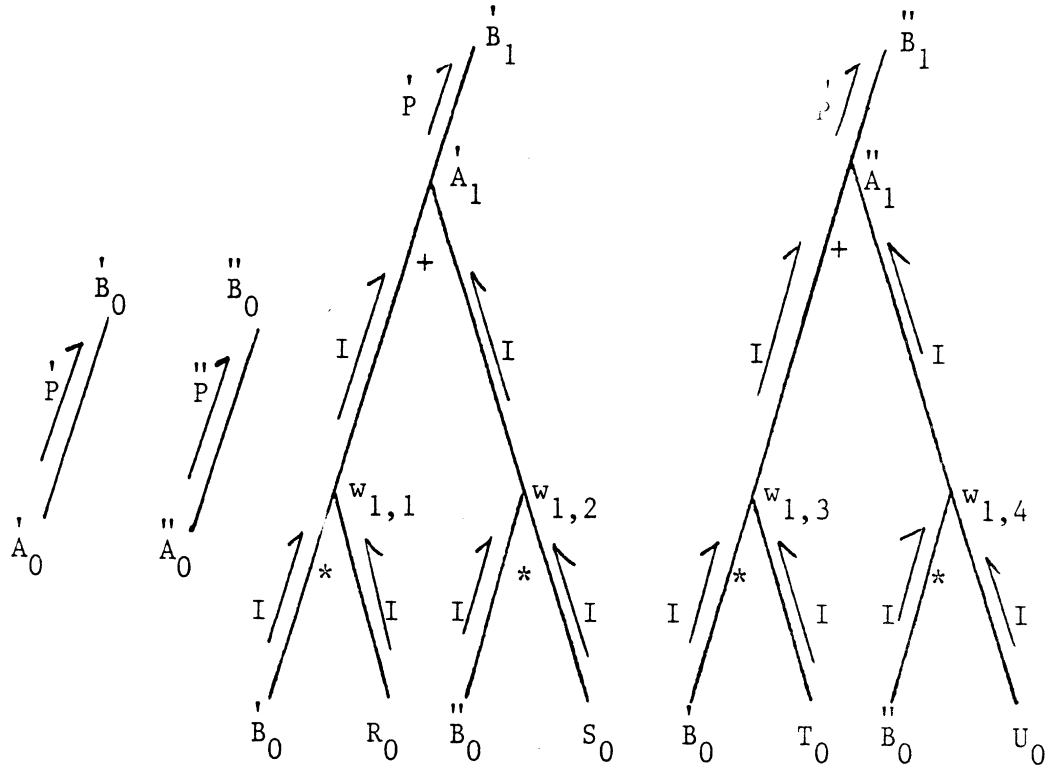


Figure 3.18 Expression trees for computing A_1 (stage 1).

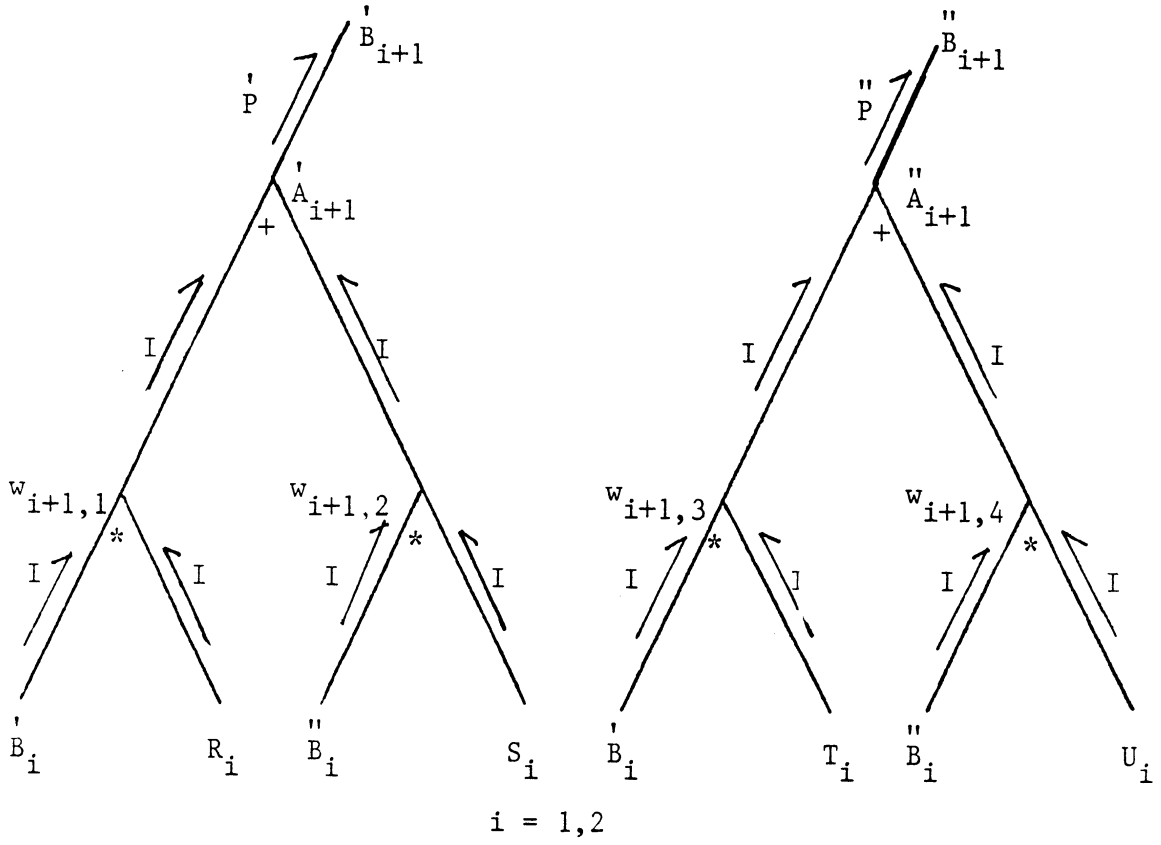


Figure 3.19 Expression trees for computing A_2 and A_3 (stage 2 & 3).

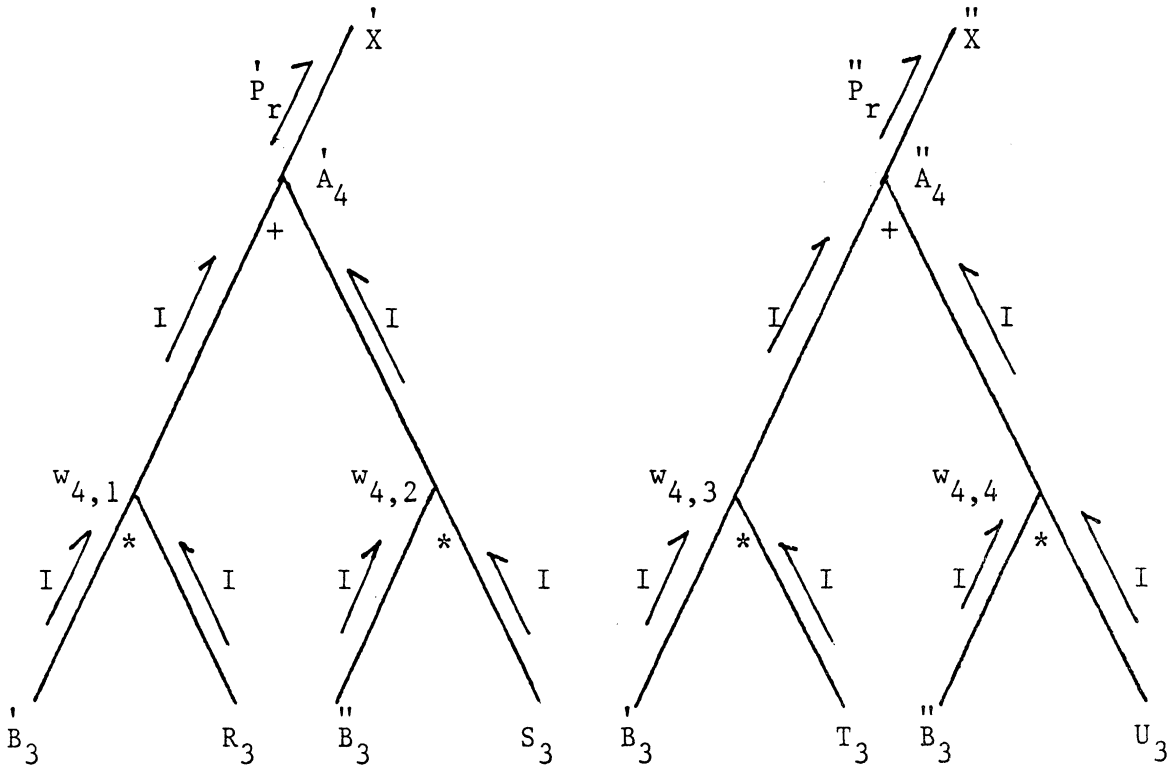


Figure 3.20 Expression trees for computing A_4 (stage 4).

Since the constant vectors, R_i, S_i, T_i & U_i, $0 \leq i \leq 3$, are used only once, if P is a logical transfer for one of the above constant vectors, then the optimum data mapping function for the vector is P. We do not have to consider data mapping functions for these constant vectors in the heuristic algorithm. The ordering of the vector variables into a sequence for the heuristic algorithm is clearly not unique. The following is one such sequence,

$$\dot{B}_3, \ddot{B}_3, \dot{B}_2, \ddot{B}_2, \dot{B}_1, \ddot{B}_1, \dot{B}_0, \ddot{B}_0, \dot{A}_0, \ddot{A}_0.$$

The total communication cost before applying the heuristic algorithm is 20 routing steps. For one of the solution obtained, the sequences of logical transfers specifying the communication requirements of $\dot{B}_i$ and $\ddot{B}_i$, $0 \leq i \leq 3$, are the same. Furthermore, $\dot{B}_i$ and $\ddot{B}_i$ have the same sequence of data mapping functions. Table II shows the sequences of logical transfers and data mapping functions determined for the vector variables. The resulting total

communication cost of the FFT algorithm is 8 routing steps. For this particular case, it is also the minimum communication cost possible.

TABLE II

Logical Transfers and Date Mapping Functions

for 16 Points FFT

Vector Variables	Sequences of Logical Transfers	Sequences of Data Mapping Functions
$\dot{B}_3, \ddot{B}_3$	$\dot{P}_r, \ddot{P}_r$	$\dot{P}_r, \dot{P}_r$
$\dot{B}_2, \ddot{B}_2$	$\dot{P}_r\dot{P}, \ddot{P}_r\ddot{P}$	$\dot{P}_r\dot{P}, \dot{P}_r\dot{P}$
$\dot{B}_1, \ddot{B}_1$	$\dot{P}_r\dot{P}^2, \ddot{P}_r\ddot{P}\ddot{P}$	$\dot{P}_r\dot{P}^2, \dot{P}_r\dot{P}^2$
$\dot{B}_0, \ddot{B}_0$	$\dot{P}_r, \ddot{P}_r\ddot{P}^2\ddot{P}$	$\dot{P}_r, \dot{P}_r$
$\dot{A}_0$	$\dot{P}_r\dot{P}$	$\dot{P}_r\dot{P}$
$\ddot{A}_0$	$\ddot{P}_r\ddot{P}$	$\ddot{P}_r\ddot{P}$

CHAPTER 4

DATA ALIGNMENT

We shall now develop techniques for determining alignment function for each internal node of the expression trees of a given parallel program.

In this section, we will assume that logical transfers for a binary operation on any two vector variables (internal or external node variables), u and v, of a given expression tree in a parallel program are specified as

$$v\,(P_v^{-1}(i)) * u\,(P_u^{-1}(i)),$$

where * stands for any binary operator.

If the alignment function for w, the temporary result vector of the binary operation, is an identity function, then logical transfers for aligning u and v to w are $P_{vw} = P_v$ and $P_{uw} = P_u$. We will assume that data mapping functions for the leaf variables of an expression tree are identity functions. If the data mapping functions are not identity functions, then we will replace each logical transfer and each data mapping function with a new logical transfer and an identity function, respectively. For instance, if the logical transfer and data mapping function for a vector variable, v_i, are $\dot{P}_{v_i}$ and $\dot{F}_{v_i}$, respectively, then the new logical transfer will be $P_{v_i} = \dot{P}_{v_i} \dot{F}_{v_i}^{-1}$ and the new data mapping function will be an identity function. Though logical transfers for a binary operation on any two temporary variables may be changed during the process of determining where the two variables should be aligned, we will assume that logical transfers for all temporary variables are initially identity functions, or else

they will be transformed recursively to identity functions as in chapter 3.

We will also assume that there is no precedence relationship between expression trees of a given parallel program. This assumption enables us to partition a parallel program with i independent expression trees into i independent subproblems. Applying the optimum data alignment algorithm developed for a single expression tree to each of the i independent subproblems wi'l produce an optimum solution for the parallel program. Based on the optimum algorithm, a heuristic algorithm also will be developed for parallel programs where precedence relationship between expression trees exists.

The data alignment algorithm is based closely on the relationship between logical transfers and alignment functions. Hence, we will be studying this relationship in the next two sections.

4.1. Logical Transfers and Alignment Functions

We now characterize the relationship between logical transfers and alignment functions. Consider the following binary operation on two leaf variables, u and v,

$$v(P_v^{-1}(i)) * u(P_u^{-1}(i)).$$

Let w be the temporary result vector of the above expression. If the alignment function for w is I, then $u(P_u^{-1}(i))$ is to be multiplied by $v(P_v^{-1}(i))$ and stored in $w(i)$. The corresponding logical transfers, P_{vw} and P_{uw}, for the binary operation are P_v and P_u, respectively (figure 4.1), and the total communication cost is given by $D(P_u) + D(P_v)$.

However, more often than not, identity alignment function will not give minimum communication cost. We may want to have the variables aligned according to another alignment function, A_w, so that $u(P_u^{-1}(i))$ will be multiplied by $v(P_v^{-1}(i))$ and stored in $w(A_w(i))$, and the corresponding total communication cost

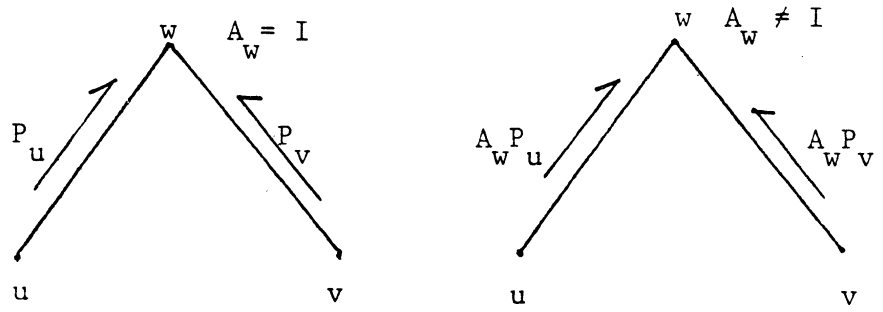


Figure 4.1 Logical transfers and alignment function.

$$D(P_{uw}) + D(P_{vw}),$$

where $P_{vw} = A_w P_v$ and $P_{uw} = A_w P_u$,

is minimized(figure 4.1). Suppose the vector variable, v, is a temporary result vector of another binary operation. Since logical transfers for temporary result vectors are initially identity functions, if the alignment function for v is A_v, then $u(P_u^{-1}(i))$ is to be multiplied by $v(A_v^{-1}(i))$ and stored in $u(A_w(i))$. The corresponding new logical transfer for aligning v to w is then $P_{vw} = A_w A_v^{-1}$, and the total communication cost for the multiplication is $D(P_{vw}) + D(P_{uw})$.

Thus, the data alignment algorithm determines, for each internal node w_i of an expression tree, a corresponding alignment function, A_{w_i} , so that the communication cost of a parallel program given by

$$\sum_{\substack{\text{statements}\\ \text{of the}\\ \text{given}\\ \text{algorithm}}} \sum_{\substack{v_i:\ \text{variable}\\ w_j:\ \text{partial}\\ \text{result}\\ w_j = \text{parent}\\ \text{of } v_i}} D(P_{v_i w_j}) + \sum_{\substack{w_i,\ w_j:\ \text{internal}\\ \text{nodes}\\ w_j = \text{parent}\\ \text{of } w_i}} D(P_{w_i w_j})$$

is minimized. In this chapter, we also will be limiting ourselves to alignment functions that are in Δ. If A_w is an alignment function in Δ, then there is a corresponding $(n+1)\times(n+1)$ nonsingular binary matrix $\mathbf{A}_w$ whose first row is $[\,1\,0\,0\,\cdots\,0\,]$.

The following are some basic theorems.

Theorem 4.1

The set of all alignment functions, $\underline{A}_{all,w}$, for logical transfer P_v, such that $A_a \in \underline{A}_{all,w}$ iff $D(P_{vw}) = D(A_aP_v) \leq 1$, is given by

$$\underline{A}_{all,w} = \{\mathbf{L}_j\mathbf{U}_i\mathbf{P}_v^{-1} \mid \mathbf{L}_j \in \underline{\mathbf{L}} \text{ and } \mathbf{U}_i \in \underline{\mathbf{U}}\}.$$

The cardinality of $\underline{A}_{all,w}$ is 2^{n^2}.

Proof :

Let $\mathbf{A}_h = \mathbf{L}_j\mathbf{U}_i\mathbf{P}_v^{-1}$. Then $\mathbf{A}_h\mathbf{P}_v = (\mathbf{L}_j\mathbf{U}_i\mathbf{P}_v^{-1})\mathbf{P}_v = \mathbf{L}_j\mathbf{U}_i$ is **LU** decomposable. Therefore, if $\mathbf{A}_h = \mathbf{L}_j\mathbf{U}_i\mathbf{P}_v^{-1}$, then $\mathbf{A}_h \in \underline{A}_{all,w}$.

Let $\mathbf{A}_h \in \underline{A}_{all,w}$. Then $\mathbf{A}_h\mathbf{P}_v = \mathbf{L}_a\mathbf{U}_b$ and $\mathbf{A}_h = \mathbf{U}_b^{-1}\mathbf{L}_a^{-1}\mathbf{P}_v^{-1}$. Therefore,

$$\underline{A}_{all,w} = \{\mathbf{L}_j\mathbf{U}_i\mathbf{P}_v^{-1} \mid \mathbf{L}_j \in \underline{\mathbf{L}} \text{ and } \mathbf{U}_i \in \underline{\mathbf{U}}\}.$$

Since there are $2^{\frac{(n^2 - n)}{2}}$ **U**'s and $2^{\frac{(n^2 + n)}{2}}$ **L**'s, $|\underline{A}_{all,w}| = 2^{n^2}$.

□

Theorem 4.2

Let $D(A_wP_v) \leq 1$. If $A_b \in \underline{\mathbf{L}}A_w$, then $D(A_bP_v) \leq 1$.

Proof :

Let $\mathbf{A}_w\mathbf{P}_v = \mathbf{LU}$. Then $(\underline{\mathbf{L}}\mathbf{A}_w)\mathbf{P}_v = \underline{\mathbf{L}}\mathbf{U}$. Therefore, if $\mathbf{A}_b \in \underline{\mathbf{L}}\mathbf{A}_w$, then $\mathbf{A}_b\mathbf{P}_v \in \underline{\mathbf{L}}\mathbf{U}$ and $D(A_bP_v) \leq 1$.

□

Theorem 4.3

Given P_v and P_u, there exists an alignment function A_w such that

$$D(A_wP_v) \leq 1 \text{ and } D(A_wP_u) \leq 1.$$

Proof :

Now every nonsingular binary matrix is **LUL** decomposable[Peas77]. Let

$$\mathbf{P}_v^{-1}\mathbf{P}_u = \mathbf{U}_r\mathbf{L}_g\mathbf{U}_t.$$

Let $\mathbf{A}_w = \mathbf{U}_r^{-1}\mathbf{P}_v^{-1}$. The logical transfers are then $P_{vw} = A_wP_v$ and $P_{uw} = A_wP_u$, where $D(P_{vw}) \leq 1$ and $D(P_{uw}) \leq 1$.

□

Theorem 4.4

Let A_{w0}, A_{w1} and A_{w2} be alignment functions for the three temporary vector variables, $w0$, $w1$ and $w2$, shown in figure 4.2. Let $All(A_{w0},A_{w1})$ be a set of alignment functions such that $A_{w2} \in All(A_{w0},A_{w1})$ iff $D(P_{w0w2}) \leq 1$ and $D(P_{w1w2}) \leq 1$, where $P_{w0w2} = A_{w2}A_{w0}^{-1}$ and $P_{w1w2} = A_{w2}A_{w1}^{-1}$. Then $All(A_{w0},A_{w1})$ can be partitioned into cosets of $\underline{\mathbf{L}}$.

Proof :

Consider $A_{w2} \in All(A_{w0},A_{w1})$, if $D(A_{w2}A_{w0}^{-1}) \leq 1$ and $D(A_{w2}A_{w1}^{-1}) \leq 1$, then $D(L_iA_{w2}A_{w0}^{-1}) \leq 1$ and $D(L_iA_{w2}A_{w1}^{-1}) \leq 1$, $L_i \in \underline{\mathbf{L}}$. Therefore, $\underline{L}A_{w2} \subseteq All(A_{w0}\,A_{w1})$ and $All(A_{w0},A_{w1})$ can be partitioned into cosets of $\underline{\mathbf{L}}$.

□

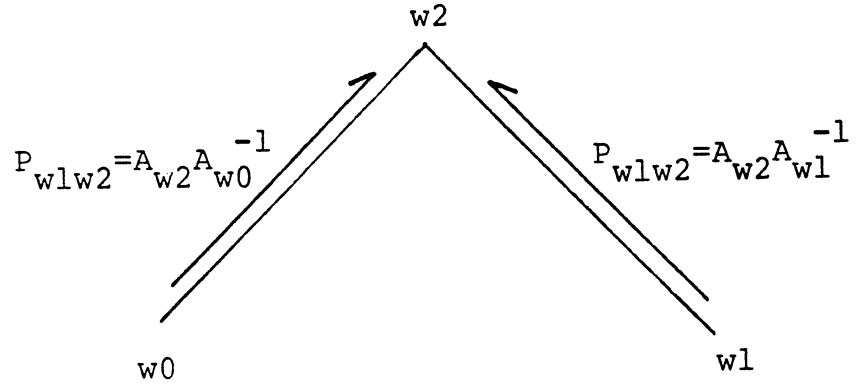


Figure 4.2 Logical transfers and alignment functions for temporary variables.

Theorem 4.5

Let A_{w0}, A_{w1} and A_{w2} be alignment functions for the three temporary vector variables, $w0$, $w1$ and $w2$, shown in figure 4.2. The logical transfer, P_{w1w2}, for $w1$ is then $A_{w2}A_{w1}^{-1}$. If $D(P_{w1w2}) = 2$, then there exists another $\mathbf{A}_b \in \underline{\mathbf{L}}\mathbf{A}_{w1}$, such that for the new $\dot{P}_{w1w2} = A_{w2}A_b^{-1}$, $D(\dot{P}_{w1w2}) \leq 1$.

Proof :

Let $\mathbf{A}_{w2}\mathbf{A}_{w1}^{-1} = \mathbf{L}_r\mathbf{U}_s\mathbf{L}_l$. Then $\mathbf{A}_{w2}\mathbf{A}_{w1}^{-1}\mathbf{L}_l^{-1} = \mathbf{L}_r\mathbf{U}_s$. Let $\mathbf{A}_b = \mathbf{L}_l\mathbf{A}_{w1}$. Then $D(\dot{P}_{w1w2}) \leq 1$, where $\dot{P}_{w1w2} = A_{w2}A_b^{-1}$.

□

While it is difficult to compute the lower and upper limits of the resulting communication cost of an arbitrary expression tree after applying the data alignment algorithm, the lower and upper limits of the resulting communication cost of a full expression tree can be computed and is given by the following theorem.

Theorem 4.6

Consider a full expression tree with $r = 2^n$, $n \geq 2$ leaf variables; if the logical transfers for the leaf variables are all different, then the resulting communication cost after determining the alignment functions is given by

$$r\text{-}1 \leq cost \leq \frac{3}{2} r.$$

Proof :

Clearly, $r\text{-}1 \leq cost$. We have to prove $cost \leq \frac{3}{2} r$ only. Consider the alignment function for $v\,(P_v^{-1}(i))$ * $u\,(P_u^{-1}(i))$ in the earlier example; by theorem 4.3, we always can find an alignment function A_w such that $D(P_{vw}) \leq 1$ and $D(P_{uw}) \leq 1$.

Let v and u be the leaf variables of a full expression tree. With w and W_G, they form a subtree (figure 4.3) of the full expression tree.

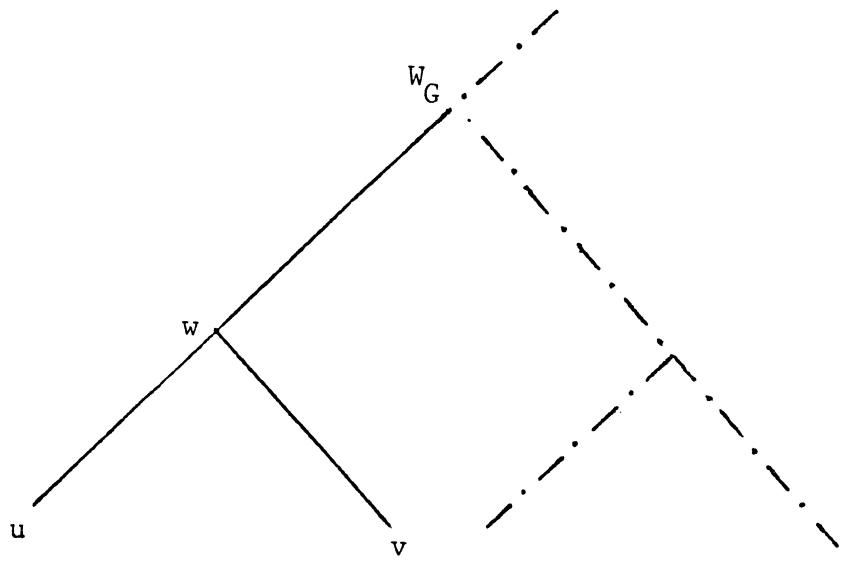


Figure 4.3 Subtree of a full expression tree.

Let the alignment function for w_G be $A_{w_G} = I$. Then by theorem 4.5, we always can find $\dot{\mathbf{A}}_w \in \underline{\mathbf{LA}}_w$ such that the logical transfer, $P_{ww_G} = A_{w_G}\dot{A}_w^{-1}$, can be realized in one pass. Furthermore, since $\dot{\mathbf{A}}_w \in \underline{\mathbf{LA}}_w$, by theorem 4.2, the new logical transfers for v and u, $\dot{P}_{vw} = \dot{A}_w P_v$ and $\dot{P}_{uw} = \dot{A}_w P_u$ can both be realized in one pass.

If we let the alignment functions for the ancestors of the parents of the leaf variables be identity and determine similarly the alignment functions for all of the leaf variables and parent nodes of the leaf variables, the total communication cost associated with the expression tree is less than or equal to $\frac{3}{2}r$.

□

4.2. Characterization of Optimum Sequences

The data alignment algorithm that we will be describing is similar to the data mapping algorithm in the previous chapter. We assume an expression tree with k nodes. The nodes are numbered in postorder fashion, and they form a sequence $\hat{V} = v_1, v_2, \ldots, v_k$. Through partial enumeration of possible partial solutions, the algorithm determines a sequence of alignment functions,

$$\hat{E} = A_1, A_2, \ldots, A_k,$$

where A_i is the alignment function for node v_i, so that the communication cost of executing the corresponding expression tree is minimized. For reasons that will be detailed later, if v_i is an external node, then we let A_i equal $P_{v_i}^{-1}$. During step h of the algorithm, $\underline{A}_h$, a set of alignment functions is determined for node v_h. Subsequences of alignment functions for the nodes of the subtree with root node v_h are formed by concatenating every $A_j \in \underline{A}_h$ with two subsequences of alignment functions, to give

$$\hat{E}_L, \hat{E}_R, A_j,$$

where $\hat{E}_L$ is a subsequence of alignment functions formed earlier for the nodes of the left subtree of v_h,

$\hat{E}_R$ is a subsequence of alignment functions formed earlier for the nodes of the right subtree of v_h.

As in data mapping, the subsequences formed will be called intermediate subsequences. Intermediate subsequences that can lead to an optimum sequence for the expression tree are saved for a future step, while the rest are eliminated. The intermediate subsequences that are saved are called candidate subsequences. A set of candidate subsequences for the subtree with root node v_h is denoted by SEQ_h. In the following, we will describe rules for eliminating some of the intermediate subsequences from consideration. We also will describe A_h, the set of alignment functions for v_h, that must be determined during step h of the algorithm.

Some basic notations and definitions for the algorithm are needed.

Notation 4.1 :

An expression tree with k nodes is designated by *k–tree*. Since the nodes are numbered in postorder fashion, the root node of *k–tree* is v_k.

□

Notation 4.2 :

For $j \leq k$, the *j–stree* is a subtree of *k–tree*. Its root node is v_j and it includes all the descendants of node v_j of *k–tree*. The tree *kj–stree* is a subtree of *k–tree* with *j–stree* removed (figure 4.4). Similarly, for $h < j$, if *h–stree* is a substree of *j–stree*, *jh–stree* is defined as a subtree of *j–stree* with *h–stree* removed.

□

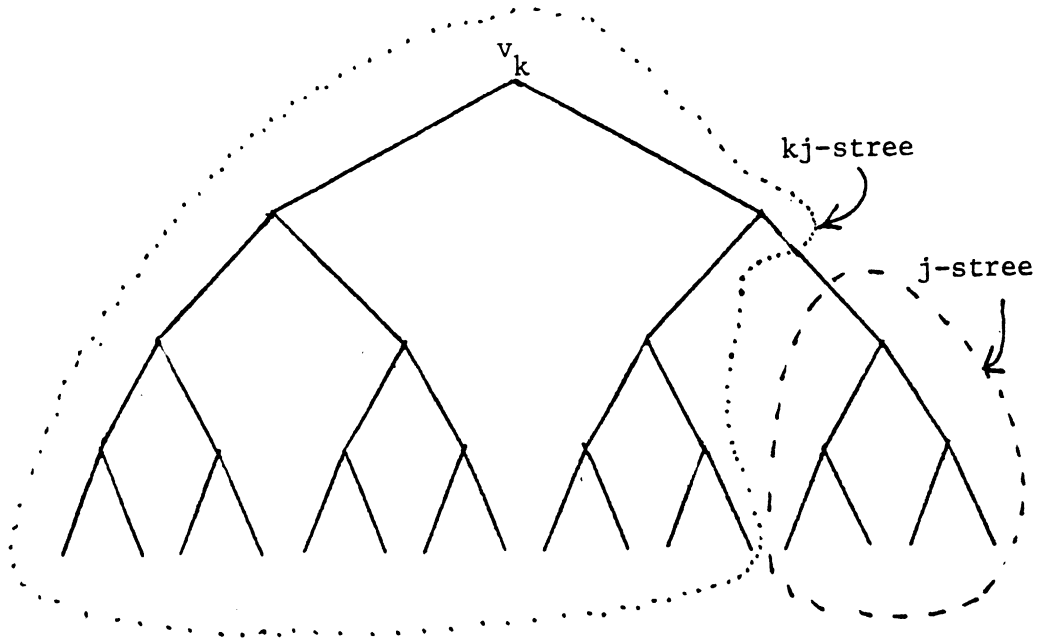


Figure 4.4 Relationship between *k-tree*, *j-stree* and *kj-stree*.

Notation 4.3 :

A sequence, $\hat{E}i_j = A_{a,i}, A_{a+1,i}, \ldots, A_{j,i}$, is a subsequence of alignment functions for subtree *j-stree* iff v_r, $a \leq r \leq j$ are nodes in *j-stree*. Here, $A_{r,i}$ in $\hat{E}i_j$ is an alignment function for node v_r. Note: since the nodes are numbered in postorder fashion, the last alignment function in a subsequence of alignment functions, $\hat{E}i_j$, for subtree *j-stree* is always the alignment function for node v_j.

□

Notation 4.4 :

$\hat{E}i_{jh}$ is a subsequence of alignment functions for subtree *jh-stree* iff every alignment function in the subsequence is for a node in *jh-stree*.

□

Definition 4.1 :

The cost of a sul sequence for a subtree is the communication cost in executing the subtree when alignment functions for all of its internal nodes are specified by the subsequence For ex: mple, the costs of $\hat{Ei}_j$ and $\hat{Ei}_{jh}$ are $d(\hat{Ei}_j)$ and $d(\hat{Ei}_{jh})$, respectively. $d(\hat{Ei}_j)$ can be defined recursively as follows,

$$d(\hat{Ei}_j) = d(\hat{Ei}_h) + d(\hat{Ei}_{jh}) + D(P_{v_h v_t}),$$

where $v_h \neq v_j$ and v_h is an internal node of *j–stree*,

v_t is the parent node of v_h,

$\hat{Ei}_{jh}$ is a subsequence of alignment functions for subtree *jh–stree*.

The cost of a sequence for an expression tree, $\hat{Ei}$, is similarly defined as $d(\hat{Ei})$.

□

Definition 4.2 :

A sequence $\hat{Ei}$ is optimum if there does not exist another sequence $\hat{Ej}$ such that

$$d(\hat{Ei}) > d(\hat{Ej}).$$

□

Notation 4.5 :

Let $\underline{S}_h$ be a set of subsequences of alignment functions for *h–stree*. Then $Min(\underline{S}_h)$ is the cost of the minimum cost for an element in $\underline{S}_h$.

□

Consider an internal node, v_b. Since the nodes are numbered in postorder fashion, the right son node of v_b is v_{b-1}. Let v_l be the left son node of v_b. Let $\hat{Ei}_l = A_a, A_{a+1}, \ldots, A_l$ and $\hat{Ej}_{b-1} = A_{l+1}, A_{l+2}, \ldots, A_{b-1}$ be subsequences of alignment functions for *l–stree* and *(b–1)–stree*, respectively. The only way to form a subsequence of alignment functions for

b–stree from these two subsequences is to concatenate the two subsequences with an alignment function, A_b, for node v_b, so that the new subsequence is

$$\hat{E}r_b = E_{i_l}, E_{j_{b-1}}, A_b = A_a, A_{a+1}, \ldots, A_l, A_{l+1}, A_{l+2}, \ldots, A_{b-1}, A_b.$$

The communication cost in executing the subtree *b–stree* is then

$$d(\hat{E}r_b) = d(\hat{E}i_l) + d(\hat{E}j_{b-1}) + D(A_b A_l^{-1}) + D(A_b A_{b-1}^{-1}).$$

The following theorems determine the conditions under which an intermediate subsequence for *h–stree* is to be eliminated. We will consider two intermediate subsequences $\hat{E}1_h$ and $\hat{E}2_h$ where $\hat{E}1_h \neq \hat{E}2_h$. Given a sequence $\hat{E}1$ having $\hat{E}1_h$ as its subsequence, we will examine conditions under which we can always find $\hat{E}2$ that has $\hat{E}2_h$ as its subsequence such that $d(\hat{E}1) \geq d(\hat{E}2)$. Since we are seeking an optimum solution only, these are conditions under which $\hat{E}1_h$ can be eliminated.

Theorem 4.7

Let $\hat{E}1_h$ be a subsequence of $\hat{E}1$. Given a subsequence $\hat{E}2_h$ such that $d(\hat{E}1_h) - d(\hat{E}2_h) \geq 2$, we can always find $\hat{E}2$ containing $\hat{E}2_h$ such that $d(\hat{E}1) \geq d(\hat{E}2)$.

Proof :

The communication cost associated with $\hat{E}1$ is

$$d(\hat{E}1) = d(\hat{E}1_h) + d(\hat{E}1_{kh}) + D(A_{t,1} A_{h,1}^{-1}),$$

where v_t is the parent node of v_h,

$A_{t,1}, A_{h,1} \in \hat{E}1$ are alignment functions for v_t and v_h, respectively.

Form $\hat{E}2$ by replacing the subsequence $\hat{E}1_h$ in $\hat{E}1$ with $\hat{E}2_h$. Then,

$$d(\hat{E}2) = d(\hat{E}2_h) + d(\hat{E}1_{kh}) + D(A_{t,1} A_{h,2}^{-1}),$$

where $A_{h,2} \in \hat{E}2$ is the alignment function for node v_h.

Now

$$d(\hat{E}1) - d(\hat{E}2) \geq 2 + D(A_{t,1}A_{h,1}^{-1}) - D(A_{t,1}A_{h,2}^{-1}).$$

Since $0 \leq D(A_{t,1}A_{h,1}^{-1}), D(A_{t,1}A_{h,2}^{-1}) \leq 2,$

$$-2 \leq D(A_{t,1}A_{h,1}^{-1}) - D(A_{t,1}A_{h,2}^{-1}) \leq 2.$$

Therefore,

$$d(\hat{E}1) - d(\hat{E}2) \geq 0.$$

Corollary 4.7.1

The candidate set, $\underline{SEQ}_h$, can be partitioned into two disjoints sets, $\underline{SEQ}_h^1$ and $\underline{SEQ}_h^0$, where

$$\hat{Si}_h \in \underline{SEQ}_h^0 \text{ iff } D_{1,h}(\hat{Si}_h) = Min(\underline{SEQ}_h),$$
$$\text{and } \hat{Si}_h \in \underline{SEQ}_h^1 \text{ iff } D_{1,h}(\hat{Si}_h) = Min(\underline{SEQ}_h) + 1.$$

□

Theorem 4.8

Let $\hat{E}1_h$ be a subsequence of $\hat{E}1$. Given a subsequence $\hat{E}2_h$ such that $A_{h,1} = A_{h,2}$ and $d(\hat{E}1_h) \geq d(\hat{E}2_h)$, we can always find $\hat{E}2$ containing $\hat{E}2_h$ such that $d(\hat{E}1) \geq d(\hat{E}2)$.

Proof :

The communication cost associated with $\hat{E}1$ is

$$d(\hat{E}1) = d(\hat{E}1_h) + d(\hat{E}1_{\bar{h}h}) + D(A_{t,1}A_{h,1}^{-1}),$$

where v_t is the parent node of v_h,

$A_{t,1}$, $A_{h,1} \in \hat{E}1$ are alignment functions for v_t and v_h, respectively.

Form $\hat{E}2$ by replacing the subsequence $\hat{E}1_h$ in $\hat{E}1$ with $\hat{E}2_h$. Then,

$$d(\hat{E}2) = d(\hat{E}2_h) + d(\hat{E}1_{kh}) + D(A_{t,1}A_{h,2}^{-1}),$$

where $A_{h,2} \in \hat{E}2$ is the alignment function for node v_h.

Since $A_{h,1} = A_{h,2}$, $d(\hat{E}1) \geq d(\hat{E}2)$.

Corollary 4.8.1

Let $\underline{a_h}$ be the set of last alignment functions in all of the subsequences in $\underline{SEQ_h}$. Then $\underline{a_h}$ can be partitioned into two sets, $\underline{a_h^z}$, $z = 0$ or 1, where $\underline{a_h^z}$ is the set of last alignment functions in all of the subsequences in $\underline{SEQ_h^z}$.

□

Theorem 4.9

Let $\hat{E}1_h$ be a subsequence of $\hat{E}1$. Given a subsequence $\hat{E}2_h$ such that $\mathbf{A}_{h,1}\mathbf{A}_{h,2}^{-1}$ is **LU** decomposable, $\mathbf{A}_{h,1} \neq \mathbf{A}_{h,2}$ and $d(\hat{E}1_h) - d(\hat{E}2_h) \geq 1$, we can always find $\hat{E}2$ containing $\hat{E}2_h$ as its subsequence such that $d(\hat{E}1) \geq d(\hat{E}2)$.

Proof :

Consider $\hat{E}1$, its cost is

$$d(\hat{E}1) = d(\hat{E}1_h) + d(\hat{E}1_{kh}) + D(A_{t,1}A_{h,1}^{-1}),$$

where $v_{t,1}$ is the parent node of v_h,

$A_{t,1}$, $A_{h,1} \in \hat{E}1$ are alignment functions for v_t and v_h, respectively.

Form $\hat{E}2$ by replacing $\hat{E}1_h$ in $\hat{E}1$ with $\hat{E}2_h$. Then,

$$d(\hat{E}2) = d(\hat{E}2_h) + d(\hat{E}2_{kh}) + D(A_{t,1}A_{h,2}^{-1})$$

where $A_{h,2} \in \hat{E}2$ is the alignment function for node v_h.

Therefore,

$$d(\hat{E}1) - d(\hat{E}2) = d(\hat{E}1_h) + d(A_{t,1}A_{h,1}^{-1}) - d(\hat{E}2_h) - d(A_{t,1}A_{h,2}^{-1})$$
$$\geq 1 + d(A_{t,1}A_{h,1}^{-1}) - d(A_{t,1}A_{h,2}^{-1}).$$

If $(A_{t,1} = A_{h,1})$, then $d(A_{t,1}A_{h,1}^{-1}) = 0$ and $d(A_{t,1}A_{h,2}^{-1}) = 1$. Hence, $d(\hat{E}1) - d(\hat{E}2) \geq 0$. If $(A_{t,1} \neq A_{h,1})$ then $1 \leq d(A_{t,1}A_{h,1}^{-1}) \leq 2$, $0 \leq d(A_{t,1}A_{h,2}^{-1}) \leq 2$, and $-1 \leq d(A_{t,1}A_{h,1}^{-1}) - d(A_{t,1}A_{h,2}^{-1}) \leq 3$. Therefore,

$$d(\hat{E}1) - d(\hat{E}2) \geq 0.$$

□

The partial enumeration algorithm will be inefficient if we let $\underline{A_h} = \Delta$, for all internal nodes v_h in the *k-tree*. Based on the following theorems, we will determine the set of alignment functions, $\underline{A_h}$, that is needed to form intermediate subsequences, $\hat{E}i_h$'s, to guarantee an optimum solution.

For convenience, we will limit ourselves to node v_h, its left son v_l, its right son v_r and their corresponding sets of candidate subsequences $\underline{SEQ_h}$, $\underline{SEQ_l}$ and $\underline{SEQ_r}$, respectively. Their corresponding sets of alignment functions are $\underline{A_h}$, $\underline{A_l}$ and $\underline{A_r}$, respectively.

If v_l is a leaf node, we will let $\underline{A_l} = \{P_l^{-1}\}$, where P_l is the logical transfer specified for v_l in the given parallel program. We will also let $\underline{SEQ_l} = \{\hat{E}_l\}$, where $\hat{E}_l = P_l^{-1}$. This can be done here because with an alignment function A_h for v_h, the resulting logical transfer, $P_{lh} = A_h A_l^{-1}$, for v_l, is mathematically equal to the actual logical transfer $A_h P_l$. If v_r is a leaf node, then $\underline{A_r}$ and $\underline{SEQ_r}$ are similarly defined.

We now characterize the set of alignment functions, $\underline{A_h}$ for internal node v_h. Let $\hat{E}j_r \in \underline{SEQ_r}$ and $\hat{E}i_l \in \underline{SEQ_l}$. Then $A_{j,r}$ and $A_{i,l}$ are alignment functions for v_r and v_l in the subsequences $\hat{E}j_r$ and $\hat{E}i_l$, respectively. Let the subset of $\underline{A_h}$ that is used with the alignment functions $A_{j,r}$ and $A_{i,l}$ be $\underline{A_h}^{ij}$. The following two theorems characterize $\underline{A_h}^{ij}$.

Theorem 4.10

If $D(A_{j,r}A_{i,l}^{-1}) \leq 1$ or $D(A_{i,l}A_{j,r}^{-1}) \leq 1$, then the set $\underline{A_h}^{ij}$ contains at most the elements $A_{i,l}$ and $A_{j,r}$.

Proof :

Suppose $D(A_{j,r}A_{i,l}^{-1}) \leq 1$. Form $\hat{E}x_h = \hat{E}i_l$, $\hat{E}j_r$, $A_{x,h}$, where $A_{x,h} \notin \{A_{i,l}, A_{j,r}\}$. The cost of $\hat{E}x_h$ is then

$$d(\hat{E}x_h) = d(\hat{E}i_l) + d(\hat{E}j_r) + D(A_{x,h}A_{i,l}^{-1}) + D(A_{x,h}A_{j,r}^{-1}).$$

We always can form $\hat{E}s_h = \hat{E}i_l$, $\hat{E}j_r$, $A_{j,r}$, such that,

$$d(\hat{E}s_h) = d(\hat{E}i_l) + d(\hat{E}j_r) + D(A_{j,r}A_{i,l}^{-1}) + D(A_{j,r}A_{j,r}^{-1}).$$

The difference, $d(\hat{E}x_h)$ - $d(\hat{E}s_h)$, is then

$$D(A_{x,h}A_{i,l}^{-1}) + D(A_{x,h}A_{j,r}^{-1}) - D(A_{j,r}A_{i,l}^{-1}).$$

Now, $1 \leq D(A_{x,h}A_{i,l}^{-1}), D(A_{x,h}A_{j,r}^{-1}) \leq 2$. Therefore, $1 \leq d(\hat{E}x_h) - d(\hat{E}s_h) \leq 4$. If the difference is greater than or equal to two, then $\hat{E}x_h \notin \underline{SEQ_h}$ (theorem 4.7). The difference is equal to one iff $D(A_{x,h}A_{i,l}^{-1}) = D(A_{x,h}A_{j,r}^{-1})=1$. But by theorem 4.9, $\hat{E}x_h \notin \underline{SEQ_h}$.

Corollary 4.10.1

If $D(A_{j,r}A_{i,l}^{-1}) = 0$, then $A_{j,r} = A_{i,l}$ and $\underline{A_h}^{ij} = \{A_{i,l}\}$.

□

Theorem 4.11

If $D(A_{j,r}A_{i,l}^{-1}) = 2$ and $D(A_{i,l}A_{j,r}^{-1}) = 2$, then it is sufficient that

$$\underline{A_h}^{ij} = \{A_{i,l}, A_{j,r}\} \bigcup All(A_{i,l}, A_{j,r}),$$

where $A_y \in All(A_{j,r}, A_{i,l})$ iff $D(A_yA_{j,r}^{-1}) \leq 1$ and $D(A_yA_{i,l}^{-1}) \leq 1$.

Proof :

Consider any alignment function $A_{x,h} \notin \{A_{j,r}, A_{i,l}\} \bigcup All(A_{j,r}, A_{i,l})$. Let

$$\hat{E}x_h = \hat{E}i_l, \hat{E}j_r, A_{x,h},$$

then,

$$d(\hat{E}x_h) = d(\hat{E}j_r) + d(\hat{E}i_l) + D(A_{x,h}A_{j,r}^{-1}) + D(A_{x,h}A_{i,l}^{-1}).$$

Suppose $D(A_{x,h}A_{i,l}^{-1}) = 2$, form $\hat{E}j_h = \hat{E}i_l, \hat{E}j_r, A_{j,r}$, such that

$$d(\hat{E}j_h) = d(\hat{E}j_r) + d(\hat{E}i_l) + D(A_{j,r}A_{i,l}^{-1}) + D(A_{j,r}A_{j,r}^{-1}).$$

Therefore,

$$\begin{aligned} d(\hat{E}x_h) - d(\hat{E}j_h) &= D(A_{x,h}A_{j,r}^{-1}) + D(A_{x,h}A_{i,l}^{-1}) - D(A_{j,r}A_{i,l}^{-1}) - D(A_{j,r}A_{j,r}^{-1}) \\ &= D(A_{x,h}A_{j,r}^{-1}). \end{aligned}$$

If $D(A_{x,h}A_{j,r}^{-1}) = 2$, then $d(\hat{E}x_h) - d(\hat{E}j_h)=2$ and $\hat{E}x_h \notin \underline{SEQ_h}$ (theorem 4.7). If $D(A_{x,h}A_{j,r}^{-1}) = 1$, then $d(\hat{E}x_h) - d(\hat{E}j_h) = 1$. By theorem 4.9, $\hat{E}x_h \notin \underline{SEQ_h}$. Similarly, if $d(A_{x,h}A_{j,r}^{-1}) = 2$, it can be shown that $\hat{E}x_h$ can be eliminated by forming $\hat{E}i_h = \hat{E}i_l, \hat{E}j_r, A_{i,l}$.

□

The following theorems determine additional conditions under which $All(A_{i,l}, A_{j,r})$ does not need to be included in $\underline{A_h}^{ij}$.

Theorem 4.12

Let $\hat{E}i_l \in \underline{SEQ_l}$ and $\hat{E}j_r \in \underline{SEQ_r^1}$. $A_{i,l}$ and $A_{j,r}$ are the last alignment functions in the subsequences $\hat{E}i_l$ and $\hat{E}j_r$. If $D(A_{i,l}A_{j,r}^{-1}) = 1$, then for an optimal solution it is sufficient that

$$\underline{A_h}^{is} = \{A_{i,l}\ ,\ A_{s,r}\} \text{ for } A_{s,r} \text{ in any } \hat{E}s_r \in \underline{SEQ_r^1}.$$

Proof :

Consider $A_{x,h}$, where $A_{x,h} \neq A_{i,l}$ and $A_{x,h} \neq A_{s,r}$, for any $\hat{E}s_r \in \underline{SEQ_r^1}$. Let

$$\hat{E}x_h = \hat{E}i_l\ ,\ \hat{E}s_r\ ,\ A_{x,h}, \qquad \hat{E}s_r \in \underline{SEQ_r^1}.$$

The communication cost associated with $\hat{E}x_h$ is then

$$d(\hat{E}x_h) = d(\hat{E}i_l) + d(\hat{E}s_r) + D(A_xA_{i,l}^{-1}) + D(A_xA_{s,r}^{-1}),$$

where $A_{s,r}$ is the last alignment function in the sequence $\hat{E}s_r$.

We can also form $\hat{E}i_h = \hat{E}i_l\ ,\ \hat{E}j_r\ ,\ A_{i,l}$. The communication cost associated with $\hat{E}i_h$ is

$$\begin{aligned} d(\hat{E}i_h) &= d(\hat{E}i_l) + d(\hat{E}j_r) + D(A_{i,l}A_{i,l}^{-1}) + D(A_{i,l}A_{j,r}^{-1}) \\ &= d(\hat{E}i_l) + d(\hat{E}j_r) + 1. \end{aligned}$$

Therefore,

$$d(\hat{E}x_h) - d(\hat{E}i_h) = D(A_xA_{i,l}^{-1}) + D(A_xA_{s,r}^{-1}) - 1.$$

Since $A_x \neq A_{s,r}$ and $A_x \neq A_{i,l}$, $1 \leq D(A_xA_{s,r}^{-1}), D(A_xA_{i,l}^{-1}) \leq 2$. Therefore, $2 \leq D(A_xA_{s,r}^{-1}) + D(A_xA_{i,l}^{-1}) \leq 4$ and $1 \leq d(\hat{E}x_h) - d(\hat{E}i_h) \leq 3$. If $d(\hat{E}x_h) - d(\hat{E}i_h) \geq 2$, then $\hat{E}x_h \notin \underline{SEQ_h}$ (theorem 4.7). $d(\hat{E}x_h) - d(\hat{E}i_h)$ is equal to one iff $D(A_xA_{i,l}^{-1}) = D(A_xA_{s,r}^{-1}) = 1$. Since $D(A_xA_{i,1}^{-1}) = 1$, by theorem 4.9, $\hat{E}x_h \notin \underline{SEQ_h}$.

Therefore, for an optimal solution, it is sufficient that $\underline{A_h}^{is} = \{A_{i,l}\ ,\ A_{s,r}\}$ for $A_{s,r}$ in any $\hat{E}s_r \in \underline{SEQ_r^1}$.

□

Theorem 4.13

Let $\hat{Ei}_l \in \underline{SEQ}_l$ and $\hat{Ej}_r \in \underline{SEQ}_r^0$. $A_{i,l}$ and $A_{j,r}$ are the last alignment functions in the subsequences $\hat{Ei}_l$ and $\hat{Ej}_r$. If $D(A_{i,l}A_{j,r}^{-1}) = 1$, then for an optimum solution, it is sufficient that

$$\underline{A_h}^{is} = \{A_{i,l}A_{s,r}\} \text{ for } A_{s,r} \text{ in any } \hat{Es}_r \in \underline{SEQ}_r^0 \bigcup \underline{SEQ}_r^1 = \underline{SEQ}_r$$

Proof :

Consider the subsequence $\hat{Ei}_h = \hat{Ei}_l\ ,\ \hat{Ej}_r\ ,\ A_{i,l}$. The cost of the subsequence, $\hat{Ei}_h$, is given by $d(\hat{Ei}_l) + Min(\underline{SEQ}_r^0) + 1$. Consider a subsequence,

$$\hat{Eu}_h = \hat{Ei}_l\ ,\ \hat{Es}_r\ ,\ A_x,$$

$$\text{where } \hat{Es}_r \in \underline{SEQ}_r^1 \text{ and } A_x \notin \{A_{s,r}\ ,\ A_{i,l}\}.$$

The cost of the subsequence, $\hat{Eu}_h$, is

$$\begin{aligned} d(\hat{Eu}_h) &\geq d(\hat{Ei}_l) + Min(\underline{SEQ}_r^1) + 2 \\ &\geq d(\hat{Ei}_l) + Min(\underline{SEQ}_r^0) + 3 \\ &\geq d(\hat{Ei}_h) + 2. \end{aligned}$$

By theorem 4.7, $\hat{Es}_r \notin \underline{SEQ}_h$. Therefore, it is sufficient for $\underline{A_h}^{is} = \{A_{i,l}\ ,\ A_{s,r}\}$ for any $A_{s,r}$ in $\hat{Es}_r \in \underline{SEQ}_r^1$. By theorem 4.12, it is sufficient for $\underline{A_h}^{is} = \{A_{i,l}\ ,\ A_{s,r}\}$ for any $A_{s,r}$ in $\hat{Es}_r \in \underline{SEQ}_r^0$, too.

□

Now, $\underline{a}_r^z$, $z = 0$ or 1, is the set of last alignment functions in all of the subsequences in $\underline{SEQ}_r^z$ (theorem 4.8). From theorem 4.5, we know that if $\underline{a}_r^z$ contains a coset of $\underline{\mathbf{L}}$, then for any $A_{i,l}$ in $\hat{Ei}_l \in \underline{SEQ}_l$ there exists an $A_{s,r}$ in $\hat{E}_r$ in $\underline{SEQ}_r^z$ such that $D(A_{i,l}A_{s,r}^{-1}) \leq 1$. Determination of whether there exists $A_{s,r}$ in $\hat{Es}_r \in \underline{SEQ}_r^z$ such that given $A_{i,l}$ of $\hat{Ei}_r$ in $\underline{SEQ}_l$, $D(A_{i,l}A_{j,s}^{-1}) = 1$ can be simplified if we can identify the existence of a coset.

The following two theorems characterize the cosets in $\underline{a}_h^i$.

Theorem 4.14

If $\underline{\mathbf{L}}\mathbf{A}_{x,h} \subseteq \underline{a}_h$, then $\underline{\mathbf{L}}\mathbf{A}_{x,h} \subseteq \underline{a}_h^0$ or $\underline{\mathbf{L}}\mathbf{A}_{x,h} \subseteq \underline{a}_h^1$.

Proof :

If the above is not true, then there exists $A_{y,h}$ in $\hat{E}y_h \in \underline{a}_h^0$ and $A_{t,h} = L_t A_{y,h}$ in $\hat{E}t_h \in \underline{a}_h^1$. But $d(\hat{E}t_h)$ - $d(\hat{E}y_h)$=1 and $D(A_{t,h}A_{y,h}^{-1}) = 1$. This contradicts theorem 4.9 that excludes $\hat{E}t_h$ from $\underline{SEQ}_h$.

□

Let $\underline{a}_h$ again be partitioned into two sets. They are $\underline{a}_{h,s}$ and $\underline{a}_{h,b}$, where $A_j \in \underline{a}_{h,b}$ iff $\underline{\mathbf{L}}A_j \subseteq \underline{a}_{h,b}$. The following theorem shows that if an alignment function, A_x, is in $\underline{a}_{h,s}$, then A_x is in $\underline{a}_{r,s}$ or $\underline{a}_{l,s}$.

Theorem 4.15

If $A_{x,h} \in \underline{a}_{h,s}$, then $A_{x,h} \in \underline{a}_{l,s} \bigcup \underline{a}_{r,s}$.

Proof :

Let $\hat{E}x_h \in \underline{SEQ}_h$ be the candidate subsequence containing $A_{x,h}$. Its cost can be broken down as followed:

$$d(\hat{E}x_h) = d(\hat{E}x_l) + d(\hat{E}x_r) + D(A_{x,h}A_{x,l}^{-1}) + D(A_{x,h}A_{x,r}^{-1}).$$

Suppose the theorem is not true, then since

$$\underline{A}_h = \bigcup_{\substack{A_{i,l} \in \underline{a}_l \\ A_{j,r} \in \underline{a}_r}} \underline{A}_h^{ij},$$

where $\underline{A}_h^{ij} = \{ A_{i,l}, A_{j,r} \}$ or $\underline{A}_h^{ij} = \{ A_{i,l}, A_{j,r} \} \bigcup All(A_{i,b} A_{j,r})$, we have to consider the following two cases only,

1. $A_{x,h} \in All(A_{x,l}, A_{x,r})$ for some $A_{x,l} \in \underline{a}_l$ and $A_{x,r} \in \underline{a}_r$, where $A_{x,h} \neq A_{x,l}$ and $A_{x,h} \neq A_{x,r}$.

or

2. $A_{x,h} \in \underline{a}_{l,b}$ or $A_{x,h} \in \underline{a}_{r,b}$.

Case 1

If $A_{x,h} \in All(A_{x,l}, A_{x,r})$ for some $A_{x,l} \in \underline{a}_l$ and $A_{x,r} \in \underline{a}_r$, then $D(A_{x,h}A_{x,r}^{-1}) = 1$ and $D(A_{x,h}A_{x,l}^{-1}) = 1$. Since $\underline{\mathbf{L}}A_{x,h} \subseteq All(A_{x,l}, A_{x,r})$, the following intermediate subsequence, $\hat{E}2_h = \hat{E}x_l, \hat{E}x_r, L_1A_{x,h}$, where L_1 is any lower triangular matrix in $\underline{\mathbf{L}}$, and

$$d(\hat{E}2_h) = d(\hat{E}x_l) + d(\hat{E}x_r) + D(L_1A_{x,h}A_{x,l}^{-1}) + D(L_1A_{x,h}A_{x,r}^{-1})$$

will be formed by the data alignment algorithm. Since

$$D(A_{x,h}A_{x,l}^{-1}) = D(L_1A_{x,h}A_{x,l}^{-1}) \text{ and } D(A_{x,h}A_{x,r}^{-1}) = D(L_1A_{x,h}A_{x,r}^{-1}),$$
$$d(\hat{E}x_h) = d(\hat{E}2_h).$$

Now, if $\hat{E}2_h$ is in $\underline{SEQ}_h$, then $\underline{\mathbf{L}}A_{x,h}$ should be a subset of $\underline{a}_{h,b}$. But this contradicts the fact that $A_{x,h} \in \underline{a}_{h,s}$. Therefore, $\hat{E}2_h$ should be eliminated by theorem 4.7 or 4.8 or 4.9. If $\hat{E}2_h$ is eliminated by theorem 4.7 or theorem 4.9, then $\hat{E}x_h$ should be eliminated, too. But this contradicts $\hat{E}x_h \in \underline{SEQ}_h$. Therefore, $\hat{E}2_h$ can be eliminated by some subsequence $\hat{E}y_h$ where $A_{h,y} = A_{h,2} = L_1A_{x,h}$ and $d(\hat{E}2_h) = d(\hat{E}y_h)$ (theorem 4.8). In that case, $L_1A_{x,h}$ is still in $\underline{a}_h$. Furthermore, $\underline{\mathbf{L}}A_{x,h} \subseteq \underline{a}_{h,b}$, and it contradicts $A_{x,h} \in \underline{a}_{h,s}$.

Case 2

Now, if $A_{x,h} = A_{x,l} \in \underline{a}_{l,b}$ or $A_{x,h} = A_{x,r} \in \underline{a}_{r,b}$, then $D(A_{x,h}A_{x,l}^{-1}) = 0$ or $D(A_{x,h}A_{x,r}^{-1}) = 0$. Suppose $A_{x,h} = A_{x,l} \in \underline{a}_{l,b}$ and $A_{x,h} \neq A_{x,r} \in \underline{A}_{r,b}$. The data alignment algorithm will form

the following intermediate subsequence

$$\hat{E}y_h = \hat{E}y_l\,,\ \hat{E}x_r\,,\ L_1A_{x,l}\,,$$

$$\text{where } A_{y,l} = L_1A_{x,l} \in \underline{a}_{l,b},$$

such that

$$d(\hat{E}y_h) = d(\hat{E}y_l) \ + \ d(\hat{E}x_r) \ + \ D(L_1A_{x,l}A_{x,r}^{-1}) \ + \ D(L_1A_{x,l}A_{x,l}^{-1}L_1^{-1}).$$

Since $d(\hat{E}\underline{y}_l) = d(\hat{E}x_l)$, using the same arguments as in case 1, it can be shown that $\underline{\mathbf{L}}\mathbf{A}_0 \subseteq a_{h,b}$, and this contradicts $A_{x,h} \in A_{h,s}$.

If $A_{x,l} = A_{x,r} = A_{x,h}$, it is obvious that $A_{x,h} \in \underline{a}_{h,s}$ iff $A_{x,l} \in \underline{a}_{l,s}$ or $A_{x,r} \in \underline{a}_{r,s}$. Otherwise, $A_{x,h} \in \underline{a}_{h,b}$ and $\underline{L}A_{x,h} \in \underline{a}_{h,b}$. Therefore, if $A_{x,h} \in \underline{a}_{h,s}$, then $A_{x,h} \in \underline{a}_{l,s}$ or $A_{x,h} \in \underline{a}_{r,s}$.

Corollary 4.15.1

Let $A_{x,h} \in \underline{a}_h$ and $UAll = \bigcup_{\substack{\underline{A}_h^{ij} \neq \{A_{i,l},\ A_{j,r}\} \\ A_{i,l} \in \underline{a}_l \\ A_{j,r} \in \underline{a}_r}} All(A_{i,l},\ A_{j,r})$.

If $A_{x,h} \in (\underline{a}_{l,b} \bigcup \underline{a}_{l,b} \bigcup UAll) \bigcap (\overline{\underline{a}_{l,s} \bigcup \underline{a}_{r,s}})$, then $A_{x,h} \in \underline{a}_{h,b}$.

Proof :

Since $A_{x,h} \in \underline{a}_h$, if $A_{x,h} \notin \underline{a}_{h,s}$, then $A_{x,h} \in \underline{a}_{h,b}$. The contrapositive of the theorem is if $A_{x,h} \notin \underline{a}_{r,s} \bigcup \underline{a}_{h,s}$, then $A_{x,h} \in \underline{a}_{h,b}$.

Now, $\underline{A}_h = \bigcup_{\substack{A_{i,l} \in \underline{a}_l \\ A_{j,r} \in \underline{a}_r}} \underline{A}_h^{ij}$

$$= \underline{a}_{r,s} \bigcup \underline{a}_{r,b} \bigcup \underline{a}_{l,s} \bigcup \underline{a}_{l,b} \bigcup UAll.$$

Therefore, if $A_{x,h} \in (\underline{a}_{l,b} \bigcup \underline{a}_{l,b} \bigcup UAll) \bigcap (\overline{\underline{a}_{l,s} \bigcup \underline{a}_{r,s}})$, then $A_{x,h} \in \underline{a}_{h,b}$.

Corollary 4.15.2

If $A_{x,h} \in \underline{a}_{h,s}$ then $A_{x,h} = P_c^{-1}$, where P_c is the logical transfer for a leaf node, v_c, in sub-tree *h-stree*.

Proof :

Clearly, applying the theorem recursively, it can be shown that if $A_{x,h} \in \underline{a}_{h,s}$ then $A_{x,h} \in \underline{a}_{c,s}$, where v_c is a leaf node of *h-stree*. But $\underline{a}_c = \underline{A}_c = \{P_c^{-1}\}$.

□

Based on theorem 4.14 and corollary 4.15.1, it can be concluded that we can treat each coset as a unit and need not consider every alignment functions in the coset. For the case, where $(\underline{a}_{l,b} \bigcup \underline{a}_{r,b} \bigcup Uall) \bigcap (\underline{a}_{l,s} \bigcup \underline{a}_{r,s}) \neq \emptyset$, if $A_{x,h} \in (\underline{a}_{l,b} \bigcup \underline{a}_{r,b} \bigcup Uall) \bigcap (\underline{a}_{l,s} \bigcup \underline{a}_{r,s})$ and $A_{x,h} \in \underline{a}_h$, then $A_{x,h} \in \underline{a}_{h,s}$ or $\underline{L}A_{x,h} \in \underline{a}_{h,b}$. In fact, except for the rare case, $\underline{L}A_x \subseteq \underline{a}_{r,s} \bigcup \underline{a}_{l,s}$ for some A_x, if $\underline{L}A_x \subseteq \underline{a}_{h,b}$, then $\underline{L}A_x \subseteq \underline{a}_{r,b}$ or $\underline{L}A_x \subseteq \underline{a}_{l,b}$ or $\underline{L}A_x \subseteq All\,(A_{i,l}, A_{j,r})$, $A_{i,l} \in \underline{a}_r$ & $A_{j,r} \in \underline{a}_l$. Instead of forming intermediate subsequences with every alignment functions in $All\,(A_{i,b}\, A_{j,r})$, only one intermediate subsequence has to be formed for each coset in $All\,(A_{i,l}, A_{j,r})$.

4.3. Data Alignment Algorithms

Based on theorems in section 4.1 and 4.2, the following algorithm could be proved to produce optimum solutions.

Algorithm III

```
MAIN PROGRAM.
BEGIN
 FIND_OPTIMUM_SEQ ( root_node , SEQ^0 , SEQ^1 );
 / ** Any sequence ∈ SEQ^0 is an optimum sequence **/
END.
```

```
SUBROUTINE FIND_OPTIMUM_SEQ ( v , SEQ^0 , SEQ^1 )
BEGIN
/* Subroutine to determine candidates subsequences for v ***/
 IF ( v_h has no children ) THEN
 BEGIN
  /** v is leaf node **/
  SEQ^0 = {P_v^-1};
  SEQ^1 = ∅;
 END
 ELSE
 BEGIN
  best = large number;
  SEQ = SEQ^0 = SEQ^1 = ∅;
  Let l be the left son of v_h;

  / ** Form SEQ_l and SEQ_r **/
  FIND_OPTIMUM_SEQ ( l, SEQ_l^0 , SEQ_l^1 );
  Let r be the right son of v_h;
  FIND_OPTIMUM_SEQ ( r, SEQ_r^0 , SEQ_r^1 );

  /** Form SEQ - the set of intermediate sequences **/
  FORM_CANDIDATE ( SEQ , SEQ_l^0 , SEQ_r^0 , best );
  FORM_CANDIDATE ( SEQ , SEQ_r^0 , SEQ_l^0 , best );
  FORM_CANDIDATE ( SEQ , SEQ_l^0 , SEQ_r^1 , best );
  FORM_CANDIDATE ( SEQ , SEQ_r^0 , SEQ_l^1 , best );

  /** Check if more intermediate subsequences needed **/
  IF ( Min(SEQ_r^1) + Min(SEQ_l^1) - best < 2 )    ( Theorem 4.6 )
  THEN
  BEGIN
   FORM_CANDIDATE ( SEQ , SEQ_r^1 , SEQ_l^1 , best );
   FORM_CANDIDATE ( SEQ , SEQ_l^1 , SEQ_r^1 , best );
  END;
  IF (Min(SEQ_l^0) + Min(SEQ_r^0) + 2 - best < 2 )    ( Theorem 4.5 )
  THEN FORM_ALL ( 0 , 0 , best , SEQ );

  IF (Min(SEQ_l^1) + Min(SEQ_r^0) + 2 - best < 2 )    ( Theorem 4.5 )
  THEN
  BEGIN
```

```
    FORM_ALL ( 0, 1, best, SEQ );
    FORM_ALL ( 1, 0, best, SEQ );
   END;

   /** Form set of candidate subsequences **/
   REDUCE SEQ according to theorem 4.5 & 4.8.;
   PARTITION SEQ into SEQ^0 and SEQ^1,
    SUCH THAT Ex̂_h ∈ SEQ^0 IFF d(Ex̂_h) = Min(SEQ);
  END;
END.

SUBROUTINE FORM_CANDIDATE ( SEQ , SEQ_l , SEQ_r , best )
BEGIN
  /** Form candidate subsequences to be included in SEQ **/
  FOR EVERY Ex̂_l ∈ SEQ_l DO
  BEGIN
   locbest = large number;
   done = false;
   WHILE ( Min(SEQ_l) + Min(SEQ_r) < locbest) AND not done DO     ( Theorem 4.6 )
   BEGIN
    /** Form candidate subsequences to be included in SEQ **/
    If A_{y,r} ∈ Ey̌_r ∈ SEQ_r represent a coset in a_r
    THEN BEGIN
        Let A_{x,l} A^{-1}_{y,r} = L_a U_b L_c
        Form Ez̃_r from Ey̌_r by replacing every A_{y,r} in Ey̌_r with L_c^{-1} A_{y,r}.
        Form Ey̌_h = Ex̂_l , E_{z,r} , L_a A_{y,r}.
        END
    ELSE BEGIN
        FORM Ey̌_h = Ex̂_l, Ey̌_r, A_{x,l}
          where A_{x,l} is an alignment function for l in Ex̂_l, Ey_r ∈ SEQ_r;
        END;

    /** Is it a possible candidate subsequence ? **/
    IF d(Ey̌_h) < locbest
    THEN
    BEGIN
     candidatesequence = Ey̌_h;
     locbest = d(Ey̌_h);
    END;
```

```
    /** DONE ? **/
    IF EVERY $\hat{E}y_r \in \underline{SEQ}_r$ has been used THEN done = true;
   END;
   INCLUDE candidate subsequence into SEQ;
  END;
END;

SUBROUTINE FORM_ALL ( i, j, best, SEQ )
BEGIN
  /** Form All($A_{z,l}$ , $A_{y,r}$) if necessary **/
  /** Form all subsequences due to All($A_{z,l}$ , $A_{y,r}$) **/
  /** ALL the subsequences form are of the same cost **/
  FOR EVERY $A_{z,r} \in \hat{E}x_r \in \underline{SEQ}_r^i$ and $A_{y,l} \in \hat{E}y_l \in \underline{SEQ}_l^j$ DO
  BEGIN
    IF NECESSARY TO FORM All($A_{z,r}$ , $A_{y,l}$) DO
    BEGIN
      /** Form candidate subsequences to be included in SEQ **/
      FOR EVERY coset in All($A_{z,r}$ , $A_{y,l}$) DO
      BEGIN
        Let an arbitrary alignment function $A_{z,h}$ in the coset to represent the coset.
        FORM $\hat{E}z_h = \hat{E}y_l$ , $\hat{E}x_l$ , $A_{z,h}$
        $\underline{SEQ} = \underline{SEQ} \bigcup \hat{E}z_h$
      END;
    END;
  END;

  IF ( best > $d(\hat{E}z_h)$ ) THEN
     best = $d(\hat{E}z_h)$;
END;
```

□

The complexity of the algorithm is clearly dependent on the number of intermediate subsequences formed throughout the algorithm. For an estimate, we will assume reduction according to theorems 4.7 and 4.9 is not made. This is necessary because it is difficult to assess the number of intermediate subsequences eliminated by theorems 4.7 and 4.9. Furthermore, we will assume that the expression tree is a full binary tree with 2^n leaf nodes. The

tree has a height of n, and the levels are labeled from 0,...,n–1 as shown in figure 4.5.

From theorems 4.10, 4.11, 4.12 and 4.13, it can be concluded that for the maximum number of intermediate subsequences, there should not be any cosets in $a^0_{l,b}$ and $a^0_{r,b}$. Furthermore, $\underline{a}^0_{l,s} = \underline{a}^0_{r,s} = 2^i$, where v_h is at level i of the tree, for each node.

The maximum number of intermediate subsequences formed by concatenating every subsequence in $\underline{SEQ}_l$ and every subsequence in $\underline{SEQ}_r$ with one of the last alignment functions in the two subsequences is $2|\underline{a}_r|\times|\underline{a}_l|$. The maximum number of cosets that we have to include in $\underline{A}_h$ is $|\underline{a}^0_{r,s}|\times|\underline{a}^0_{l,s}|t$, where t is the maximum number of cosets in $All(A_{x,l}, A_{u,r})$, $A_{x,l} \in \underline{a}^0_{l,s}$ and $A_{x,r} \in \underline{a}^0_{r,s}$. Therefore, the maximum number of intermediate subsequences formed with an alignment function from one of the cosets is $|\underline{a}^0_{r,s}|\times|\underline{a}^0_{l,s}|t$. Hence, for the worst case,

$$\text{the number of intermediate subsequences} = 2|\underline{a}_r|\times|\underline{a}_l| + t|\underline{a}^0_{r,s}|\times|\underline{a}^0_{l,s}|,$$

where t is the maximum number of cosets in $ALL(A_{x,l}, A_{y,r})$, $A_{x,l} \in \underline{a}^0_{l,s}$ and $A_{y,r} \in \underline{a}^0_{r,s}$.

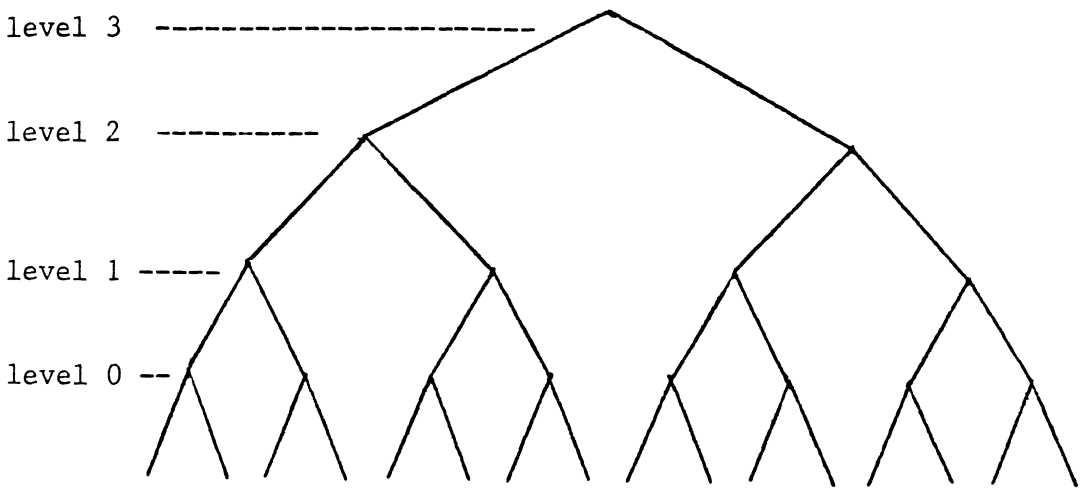


Figure 4.5 A full expression tree.

With no subsequence elimination according to theorems 4.7 and 4.9, the number of subsequences in $\underline{SET}_h = |\underline{A}_h|$. Since only $|\underline{a}_r| + |\underline{a}_l|$ candidate subsequences will be produced from the $2|\underline{a}_r| \times |\underline{a}_l|$ intermediate subsequences (Theorem 4.8), $|\underline{A}_h| = |\underline{a}_r| + |\underline{a}_l| + |\underline{a}^0_{r,s}| \times |\underline{a}^0_{l,s}| t$. The following theorem finds maximum $|\underline{A}_h|$.

Theorem 4.16

Maximum $|\underline{A}_h|$ formed for node, v_h , which is at level i is

$$2^i(2^{i+1} - 1)t + 2^{i+1},$$

where t = the maximum of the number of cosets in $ALL(A_x, A_y)$ for all A_x, A_y and $A_x \neq A_y$.

Proof :

Mathematic induction will be used to prove this theorem. Consider a node at level 0. $|\underline{a}_r| = |\underline{a}_l| = |\underline{a}^0_{r,s}| = |\underline{a}^0_{l,s}| = 1$. Therefore,

$$|\underline{A}_0| = |\underline{a}_r| + |\underline{a}_l| + |\underline{a}^0_{r,s}| \times |\underline{a}^0_{l,s}| t ,$$

$$= 2^0(2^1 - 1)t + 2^1,$$

$$= 2^0(t) + 2^1 = t+1.$$

Now, for a node v_x at level i, maximum $|\underline{a}^0_x| = 2^i$ (theorem 4.15). Suppose the theorem is true for node up to level i-1. Consider node v at level i.

$$|\underline{a}_r| = |\underline{a}_l| = 2^{i-1}(1 + 2 + \cdots 2^{i-1})t + 2^i.$$

$$|\underline{a}^0_{r,s}| = |\underline{a}^1_{l,s}|.$$

Therefore,

$$|\underline{A}_h| = |\underline{a}_l| + |\underline{a}_r| + |\underline{a}^0_{l,s}| \times |\underline{a}^0_{r,s}| t$$

$$= 2(2^{i-1}(1 + 2 + \cdots + 2^{i-1})t + 2^i + 2^i \times 2^i t$$

$$= 2^i(1 + 2 + \cdots + 2^{i-1})t + 2^{i+1} + 2^{2i}t$$

$$= 2^i(1 + 2 + \cdots 2^{i-1} + 2^i)t + 2^{i+1}$$

$$= 2^i(2^{i+1} - 1)t + 2^{i+1}.$$

□

Theorem 4.17 :

The complexity of the algorithm is $O(k^4)$, where k is the total number of internal and external nodes.

Proof :

Now, the number of intermediate subsequences formed for node v_h at level i is

$$2|\underline{a}_r| \times |\underline{a}_l| + t(2^{2i}) = 2|\underline{A}_r| \times |\underline{A}_l| + t(2^{2i})$$

$$= 2(2^{i-1}(2^i-1)t + 2^i)^2 + 2^{2i}t.$$

Since we are concerned with complexity only, we only need be concerned with the factor $2^{4i-2}t$. The number of nodes at level h is given by $\frac{2^n}{2^{h+1}}$. The total number of intermediate subsequences at level h is, hence, of the the order of $2^{n+3h-3}t$. The total number of intermediate subsequences generated throughout the algorithm is then of the order of

$$\sum_{i=1}^{n} 2^{n+3i-3}t = \frac{2^n(2^{3n}-1)}{7}\,t.$$

Since there are $\frac{k+1}{2} = 2^n$ leaf nodes in the *k-tree* (a full expression tree), the complexity of the algorithm is $O(k^4)$.

□

For a large *k-tree*, $\underline{A_h}$ in algorithm III may still grow larger and larger, with v_h getting closer and closer to the root node, until it equals Δ. Thus, given a parallel program with large expression trees, we may have to partition each large expression tree into smaller expression trees where precedence relationship exists, and use the following heuristic algorithm to solve for a near-optimal solution.

A Heuristic Algorithm

Consider the parallel algorithm shown in figure 3.2. Here, expression tree #2 is dependent on expression tree #1. Since the optimum alignment function obtained by algorithm III for v_A in expression tree #1 may not be identity, we must appropriately determine the logical transfers for A in expression tree #2 before solving for an optimum sequence of alignment functions for expression tree #2. More often than not, algorithm III will produce a set of optimum alignment functions for v_A. Thus, the problem of determining which of them will lead to an optimum solution, if it ever will, is complex. Hence, we have developed a heuristic algorithm for finding a near-optimum solution when precedence relationship between expression trees exists.

Algorithm IV

```
1. FOR j = 1 TO z = "no. of expression trees" DO
   BEGIN
     Determine optimum sequence of alignment functions for expression tree j.
     Pick a no. of optimum alignment functions for the result vector of expression

        tree j.
     For each of them determine optimum sequences for other expression
        trees as follows
     FOR i=j+1 TO z DO
     BEGIN
       Determine all the logical transfers for the expression tree i
       Determine optimum sequences of alignment functions for expression i
       Arbitrarily pick an optimum sequence of alignment functions for expression tree i
```

END;
Keep the optimum sequence of alignment functions (for expression tree j) that gives lowest total cost to the complete problem.
If $j < z$ then determine all the logical transfers for the expression tree j+1.
END;

□

The above heuristic algorithm has been tested on a number of example problems, its performance has been compared with that of a total enumeration algorithm and it has been found that in most cases, the heuristic algorithm generates optimal or near-optimal solutions.

The results of five experiments on the heuristic algorithm are shown in table III. In each experiment, the parallel program contains three full binary trees, each with four leaf nodes (figure 4.6). Each tree has the same vector variable for all its leaf nodes and the logical transfers for the leaf nodes are different. v_z, $2 \leq z \leq 3$ is the result vector of tree #(z-1). Thus, there is precedence among the three trees. The five parallel programs differ in their communication requirements for their vector variables.

Table III

Experiments on the Heuristic Algorithm

Parallel program	Optimum communication cost	Resulting communication cost (heuristic algorithm)
1	15	15
2	14	14
3	14	14
4	13	13
5	13	14

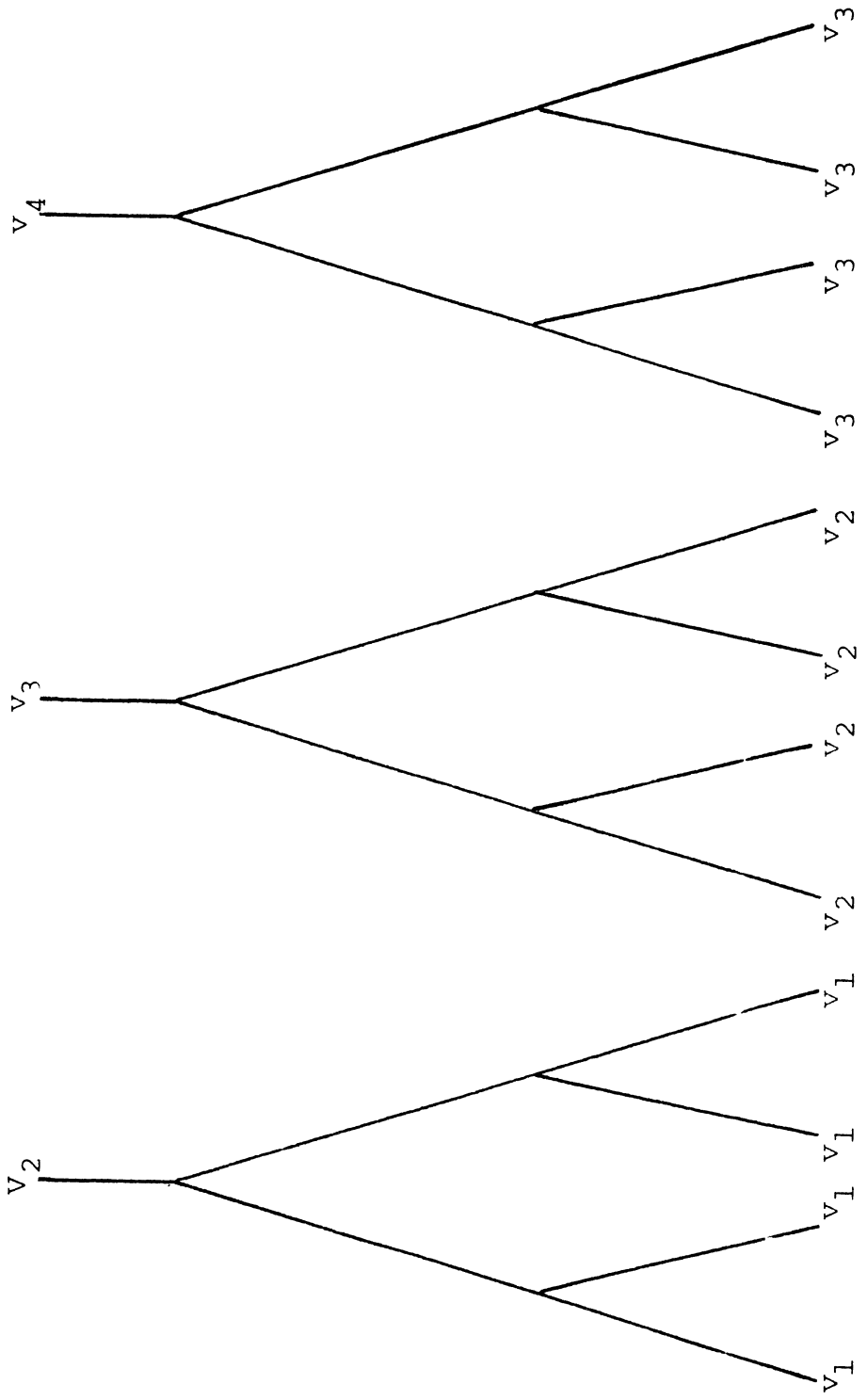


Figure 4.6 Expression trees of a parallel program for data alignment.

4.4. Examples

The heuristic algorithm developed in this chapter was used to find alignment functions for the lower triangular matrix inversion algorithm and the FFT algorithm described in chapter 3.

For the lower triangular matrix inversion algorithm, alignment functions for w_2, $\dot{w}_2$, w_3, $\dot{w}_3$, E, w_4, $\dot{w}_4$, w_5, $\dot{w}_5$, F are determined to be $\theta_5 \cdot P_a$. Alignment functions for w_6, $\dot{w}_6$, w_7, $\dot{w}_7$, w_8, $\dot{w}_8$, w_9, $\dot{w}_9$, w_{10}, $\dot{w}_{10}$, w_{11}, $\dot{w}_{11}$, T, w_{12}, $\dot{w}_{12}$, w_{13}, $\dot{w}_{13}$, w_{14}, $\dot{w}_{14}$, w_{15}, $\dot{w}_{15}$, w_{16}, $\dot{w}_{16}$, w_{17}, $\dot{w}_{17}$ and S are determined to be P_a. Alignment functions for other variables, e.g., C, are determined to be identity functions. The resulting communication cost of executing the inversion algorithm is 21 routing steps. As in the data mapping case, the above solution is determined to be optimum.

For the FFT algorithm described in chapter 3, the alignment functions for $\dot{B}_0$ and $\ddot{B}_0$ are determined to be $\dot{P}$ and $\ddot{P}$, respectively. For $\dot{A}_i$, $\dot{B}_i$, $w_{i,1}$, $w_{i,2}$, $1 \leq i \leq 3$, the alignment functions are $(\dot{P}^{-1})^{i+1}$. For $\dot{A}_i$, $\dot{B}_i$, $w_{i,1}$, $w_{i,2}$, $1 \leq i \leq 3$, the alignment functions are $(\ddot{P}^{-1})^{i+1}$. The alignment functions for $\dot{A}_4$, $\dot{X}$, $w_{3,1}$ and $w_{3,2}$ are $\dot{P}^{-1}\dot{P}_r^{-1}$. The alignment functions for $\ddot{A}_4$, $\ddot{X}$, $w_{3,3}$ and $w_{3,4}$ are $\ddot{P}^{-1}\ddot{P}_r^{-1}$. The resulting communication cost is 8 routing steps, which is the minimum communication cost possible.

CHAPTER 5

APPLICATIONS FOR OTHER NETWORKS

In this chapter, we will show how the algorithms developed in chapters 3 and 4 are applicable to Γ, a class of multistage interconnection networks that is functionally equivalent to the omega network. Two networks, the N_1 network and the N_2 network, are defined to be functionally equivalent [WuFe79] iff

$$\underline{Q}_1 = R_1 \underline{Q}_2 R_2,$$

where $\underline{Q}_1$ is the set of admissible network permutations for the N_1 network,

$\underline{Q}_2$ is the set of admissible network permutations for the N_2 network,

R_1 and R_2 are permutations.

Any admissible network permutation of the N_1 network can be decomposed into three functions, a permutation R_1, an admissible network permutation for N_2 and a permutation R_2. For Γ, the class of multistage interconnection networks with which we are concerned, $R_1 = R_2 = R_1^{-1}$ and R_1 is in Δ. As an example, if $\underline{Q}_2$ is equal to Ω, the set of admissible network permutations for the omega network, and $\underline{Q}_1$ is the set of admissible network permutations for the indirect binary n-cube[Peas77] shown in figure 5.1, then $R_1 = R_2 = R_1^{-1}$ is a bit reversal permutation.

In the following two sections, we will be concerned with a network in Γ, the N_1 network. It is functionally related to the omega network as follows.

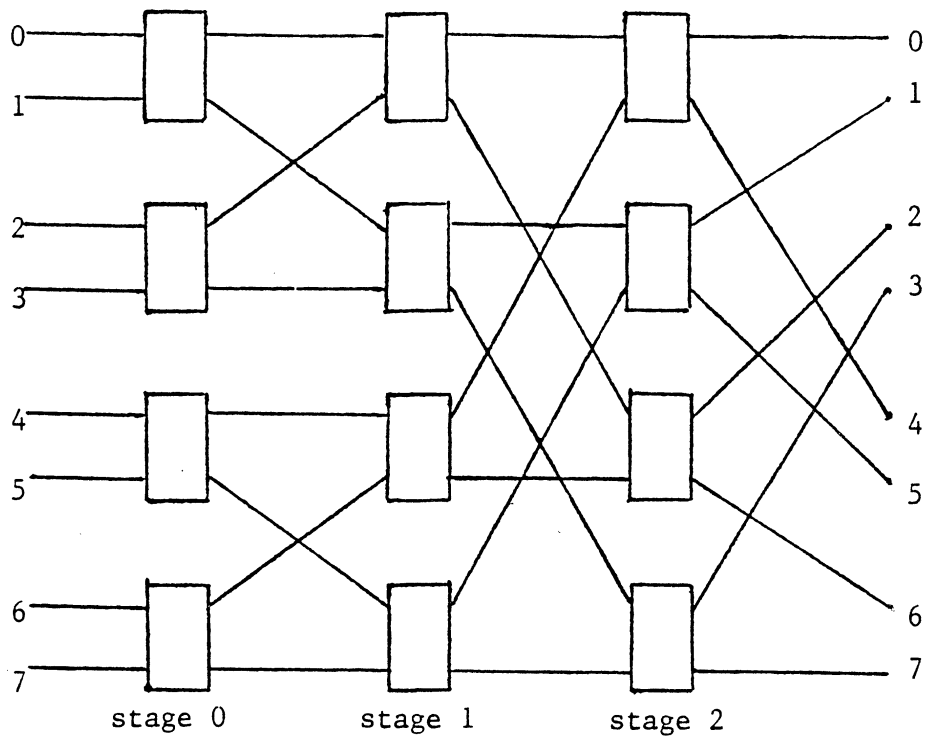


Figure 5.1 A $2^3 \times 2^3$ indirect binary n-cube.

$$\underline{Q}_1 = R\,\Omega\,R,$$

where $\underline{Q}_1$ is the set of admissible network permutations for the N_1 network,

R is a permutation in Δ and $R = R^{-1}$.

In the following, we will let S_1 denote the SIMD multicomputer system with a N_1 network. We will let S_Ω denote the SIMD multicomputer system with an Ω network. We also will let D_Ω and D_N be the distance functions associated with the omega network and the N_1 network, respectively. The following lemma relates the two distance functions, D_Ω and D_N.

Lemma 5.1

$$D_\Omega(\mathbf{P}) = D_N(\mathbf{RPR}), \text{ for } \mathbf{P} \in \Delta.$$

Proof :

Since the omega network and the N_1 network are functionally equivalent, $\mathbf{P} \in \Omega$ iff $\mathbf{RPR} \in \underline{Q}_1$. Therefore, $D_\Omega(\mathbf{P}) = 1$ iff $D_N(\mathbf{RPR}) = 1$.

Now, $D_N(\mathbf{P}) = 0$ iff $\mathbf{P} = \mathbf{I}$. Similarly, $D_N(\mathbf{RPR}) = 0$ iff $\mathbf{RPR} = \mathbf{I}$. Clearly, $\mathbf{P} = \mathbf{I}$, iff $\mathbf{RPR} = \mathbf{I}$. Therefore, $D_\Omega(\mathbf{P}) = 0$ iff $D_N(\mathbf{RPR}) = 0$.

If $D_\Omega(\mathbf{P}) = 2$, then $\mathbf{P}$ is $\mathbf{LUL}$ decomposable. Let $\mathbf{P} = \mathbf{L}_a\mathbf{U}_b\mathbf{L}_c$. $\mathbf{P}$, therefore, can be realized by the following two admissible physical transfers through the omega network,

$$\mathbf{L}_c \text{ and then } \mathbf{L}_a\mathbf{U}_b.$$

Now, $\mathbf{RL}_c\mathbf{R}$ and $\mathbf{RL}_a\mathbf{U}_b\mathbf{R}$ both can be realized in one pass through the N_1 network. Therefore, $\mathbf{RL}_a\mathbf{U}_b\mathbf{RRL}_c\mathbf{R} = \mathbf{RL}_a\mathbf{U}_b\mathbf{L}_c\mathbf{R} = \mathbf{RPR}$ can be realized in two passes through the N_1 network.

Now, let $\mathbf{RPR}$ be a permutation that cannot be realized in one pass through the N_1 network. Clearly, $\mathbf{P}$ can be decomposed into $\mathbf{L}_d\mathbf{U}_e\mathbf{L}_f$, where $\mathbf{L}_d \neq \mathbf{I}$, $\mathbf{U}_e \neq \mathbf{I}$ and $\mathbf{L}_f \neq \mathbf{I}$. Then $\mathbf{RPR} = \mathbf{RL}_d\mathbf{U}_e\mathbf{L}_f\mathbf{R} = \mathbf{RL}_d\mathbf{U}_e\mathbf{RRL}_f\mathbf{R}$. $\mathbf{RPR}$, therefore, can be realized by the following two admissible physical transfers through the N_1 network,

$$\mathbf{RL}_f\mathbf{R} \text{ and then } \mathbf{RL}_d\mathbf{U}_e\mathbf{R}.$$

Therefore, every permutation in Δ can be realized in less than or equal to two passes through the N_1 network, and $D_\Omega(\mathbf{P}) = 2$ iff $D_N(\mathbf{RPR}) = 2$.

Clearly, it follows that $D_\Omega(\mathbf{P}) = D_N(\mathbf{RPR})$, for $\mathbf{P} \in \Delta$.

□

In section 5.1, we will show how data mapping algorithms developed in chapter 3 can be used to map a parallel program onto S_1. In section 5.2, we will show how data alignment algorithms developed in chapter 4 can be used for S_1.

5.1. Data Mapping Algorithms

In this section, we will show how data mapping algorithms developed in chapter 3 can be used to map optimally or near-optimally parallel programs onto S_1.

We will first describe a data mapping algorithm for parallel programs whose expression trees are independent. Algorithm V below optimally will map such a parallel program onto S_1.

Algorithm V

1. Transform the given parallel program by multiplying **R** by every logical transfer, $\mathbf{P}_i$, in the parallel program to give a corresponding new logical transfer $\mathbf{RP}_i$.

2. Apply Algorithm I to map optimally each subproblem of the transformed parallel program onto S_Ω.

3. Multiply each data mapping function, $\mathbf{F}_i$, thus obtained for $\mathbf{RP}_i$ by **R** to give a new data mapping function $\mathbf{RF}_i$. This data mapping function $\mathbf{RF}_i$ is used in the new system S_1, for the corresponding logical transfer, $\mathbf{P}_i$, in the original parallel program.

□

The set of data mapping functions for S_1 thus obtained can be shown to map optimally the given parallel program onto S_1. In the following, we will first show that the total communication cost of executing the transformed parallel program mapped onto S_Ω is equal to the communication cost of executing the original parallel program correspondingly mapped onto S_1 with the N_1 network. Then, we will show that the data mapping algorithm does indeed produce an optimum solution.

Theorem 5.1

The total communication cost of executing the transformed parallel program mapped onto S_Ω is equal to the total communication cost of executing the original parallel program correspondingly mapped onto S_1.

Proof :

The communication cost of executing a parallel program mapped onto a multicomputer system is composed of the communication cost in realizing all of the logical transfers in the parallel program and the communication cost in remapping.

Based on algorithm V, if $\mathbf{P}_i$ is the logical transfer in the original parallel program, then $\mathbf{RP}_i$ is the corresponding logical transfer in the transformed parallel program. If $\mathbf{F}_i$ is the data mapping function for $\mathbf{RP}_i$ in the transformed parallel program, then $\mathbf{RF}_i$ is the data mapping function for $\mathbf{P}_i$ in the original parallel program for S_1.

Now, the communication cost of realizing logical transfer, $\mathbf{P}_i$, through the N_1 network is $D_N(\mathbf{P}_i\mathbf{F}_i^{-1}\mathbf{R})$. The communication cost of realizing the corresponding logical transfer, $\mathbf{RP}_i$ through the omega network is $D_\Omega(\mathbf{RP}_i\mathbf{F}_i^{-1})$. Based on lemma 5.1, $D_N(\mathbf{P}_i\mathbf{F}_i^{-1}\mathbf{R}) = D_\Omega(\mathbf{RP}_i\mathbf{F}_i^{-1})$.

Let $\mathbf{F}_i$ and $\mathbf{F}_{i+1}$ be the data mapping functions for the ith and i+1th logical transfers for a vector variable, v, in the transformed parallel program, respectively. Then, $\mathbf{RF}_i$ and $\mathbf{RF}_{i+1}$ are the corresponding data mapping functions for the ith and i+1th logical transfers for v in the original parallel program, respectively.

Now, the communication cost of realizing the remapping, $\mathbf{RF}_{i+1}\mathbf{F}_i^{-1}\mathbf{R}$, through the N_1 network is $D_N(\mathbf{RF}_{i+1}\mathbf{F}_i^{-1}\mathbf{R})$. The communication cost of realizing the corresponding remapping, $\mathbf{F}_{i+1}\mathbf{F}_i^{-1}$ through the omega network is $D_\Omega(\mathbf{F}_{i+1}\mathbf{F}_i^{-1})$. Based on lemma 5.1, $D_N(\mathbf{RF}_{i+1}\mathbf{F}_i^{-1}\mathbf{R}) = D_\Omega(\mathbf{F}_{i+1}\mathbf{F}_i^{-1})$.

Since the remapping cost and the cost of realizing the logical transfers are equal in both cases, the total communication cost of executing the transformed parallel program mapped onto S_Ω is equal to the total communication cost of executing the original parallel program correspondingly mapped onto S_1.

□

Theorem 5.2

Algorithm V will optimally map a given parallel program onto an SIMD multicomputer system with a N_1 network.

Proof :

Based on theorem 5.1, the communication cost of executing the transformed parallel program mapped onto S_Ω is equal to the communication cost of executing the original parallel program correspondingly mapped onto S_1.

If the data mapping solution for the parallel program on the N_1 network is not optimum, then there exists another data mapping solution with lower total communication

cost. This means that there exists a better data mapping solution for the execution of the transformed parallel program on S_Ω. This contradicts the fact that algorithm I in chapter 3 is optimum. Therefore, algorithm V is an optimum algorithm.

□

For parallel programs where precedence relationship between expression trees exists, the heuristic algorithm in chapter 3 can be modified similarly. Since logical transfers for a vector variable, v, may depend on the data mapping functions of other vector variables, the logical transfers for v must not be transformed until all data mapping functions for those vector variables and the logical transfers for v are determined. The following heuristic algorithm is a modified version of algorithm II in chapter 3.

Algorithm VI

1. Determine the precedence relationships between the vector variables and put them in a sequence, $v_1, v_2, \ldots, v_z$, such that logical transfers for v_j is not dependent on any variables v_i , $i > j$.

2. For j=1 to z DO

   BEGIN

   Transform all the logical transfers, $\mathbf{P}_{f,j}$, for v_j to $\mathbf{RP}_{f,j}$.

   Apply algorithm I to determine optimum sequences of data mapping functions for v_j.

   Pick *limit* number of optimum sequences for v_j and for each of them determine optimum sequences for other variables as followed :

   For i=j+1 TO z DO

   BEGIN

   Determine logical transfers for v_i.

   Transform all the logical transfers,$\mathbf{P}_{f,i}$, for v_i to $\mathbf{RP}_{f,i}$.

Apply algorithm I to determine optimum sequences of data mapping functions for v_i.

Arbitrarily pick an optimum sequence for v_i.

END;

Keep the optimum sequence (for v_j) that leads to the lowest total cost

for all the variables.

If $j \neq z$, determine logical transfers for v_{j+1}.

END;

□

5.2. Data Alignment Algorithms

A similar approach can be taken in adapting data alignment algorithms in chapter 4 for S_1. For parallel programs where no precedence relationship between its expression trees exists, algorithm VII below provides an optimal solution.

Algorithm VII

1. Transform the given parallel program by multiplying **R** by logical transfers that are for the leaf variables of the expression trees in the program. If **P** is a logical transfer for a leaf variable, then **PR** is the corresponding new logical transfer.

2. Apply algorithm III to each expression tree of the transformed parallel program to obtain an optimum sequence of alignment functions.

3. Each alignment function, $\mathbf{A}_i$, thus obtained for an internal node (variable v_i) is then to be multiplied by **R** to give a new data alignment function $\mathbf{RA}_i$ for the

corresponding node v_i in the original parallel program.

□

The set of alignment functions thus obtained can be shown to be optimum for the execution of the given parallel program on S_1.

Notation 5.1 :

Let $\underline{O}$ be the set of optimum sequences of alignment functions obtained in step 3 of algorithm VII, each sequence being for an unique expression tree in the given parallel program. Let $\underline{T}$ be the corresponding set of optimum sequences of alignment functions obtained in step 2 of algorithm VII, each sequence being for an unique expression tree in the transformed parallel program.

□

As we did for data mapping, we will show that with $\underline{O}$ and $\underline{T}$, the total communication cost in executing the transformed parallel program on S_Ω is equal to the total communication cost in executing the corresponding parallel program on S_1. We will then show that algorithm VII does indeed produce an optimum solution.

Theorem 5.3

With $\underline{O}$ and $\underline{T}$ determined for a given parallel program, the total communication cost in executing the transformed parallel program on S_Ω is equal to the total communication cost in executing the original parallel program on S_1.

Proof :

Based on algorithm VII, if $\mathbf{P}_i$ is a logical transfer for a leaf variable in the original parallel program, then $\mathbf{P}_i\mathbf{R}$ is the corresponding logical transfer in the transformed parallel program. If $\mathbf{A}_i$ is the data alignment function for an internal node (variable v_i) in the transformed parallel program, then $\mathbf{RA}_i$ is the corresponding data alignment function for v_i in the original parallel program.

Now, the communication cost of realizing logical transfer, $\mathbf{P}_i$, through the N_1 network is $D_N(\mathbf{RA}_i\mathbf{P}_i)$, where node i (variable v_i)is the parent node of the leaf node that has $\mathbf{P}_i$ as its logical transfer. The communication cost of realizing the corresponding logical transfer, $\mathbf{RP}_i$ through the omega network is $D_\Omega(\mathbf{A}_i\mathbf{P}_i\mathbf{R})$. Based on lemma 5.1, $D_N(\mathbf{RA}_i\mathbf{P}_i) = D_\Omega(\mathbf{A}_i\mathbf{P}_i\mathbf{R})$.

Let node j (variable v_j) be the parent node of node i. Let $\mathbf{A}_j$ be the data alignment function for v_j in the transformed parallel program. Then $\mathbf{RA}_i$ and $\mathbf{RA}_j$ are the corresponding data alignment functions for v_i and v_j in the original parallel program, respectively. The logical transfers for aligning v_i to v_j in the transformed parallel program and the original parallel program are then $\mathbf{A}_j\mathbf{A}_i^{-1}$ and $\mathbf{RA}_j\mathbf{A}_i^{-1}\mathbf{R}$, respectively.

Now, the communication cost of realizing the logical transfer of $\mathbf{RA}_j\mathbf{A}_i^{-1}\mathbf{R}$, through the N_1 network is $D_N(\mathbf{RA}_j\mathbf{F}_i^{-1}\mathbf{R})$. The communication cost of realizing the corresponding logical transfer, $\mathbf{A}_i\mathbf{A}_j^{-1}$ through the omega network is $D_\Omega(\mathbf{A}_j\mathbf{A}_i^{-1})$. Based on lemma 5.1, $D_N(\mathbf{RA}_j\mathbf{A}_i^{-1}\mathbf{R}) = D_\Omega(\mathbf{A}_j\mathbf{A}_i^{-1})$.

Since the communication costs of realizing the logical transfers are equal in both cases, given $\underline{O}$ and $\underline{T}$, the total communication cost of executing the transformed parallel program on S_Ω is equal to the total communication cost of executing the corresponding original parallel program on S_1.

□

Theorem 5.4

The data alignment algorithm described for S_1 is an optimum algorithm that will minimize the total communication cost of a given parallel program.

Proof :

Based on theorem 5.3, with $\underline{O}$ and $\underline{T}$, the communication cost of executing the transformed parallel program on S_Ω is equal to the communication cost of executing the corresponding original parallel program on S_1.

If the set of data alignment functions for the parallel program on the N_1 network is not optimum, then there exists another set of data alignment functions that gives lower total communication cost. This means that there exists a better set of data alignment functions for the execution of the transformed parallel program on S_Ω. This contradicts the fact that algorithm III in chapter 4 is optimum. Therefore, algorithm VII is optimum.

□

For a parallel program where precedence relationship exists, the heuristic algorithm in chapter 4 can be modified. Since logical transfers for the leaf variables of expression tree j may depend on expression tree i, where $i < j$, they are not to be transformed until alignment functions for expression tree i , where $i < j$, are determined. The following heuristic algorithm is a modified version of algorithm VI in chapter 4.

Algorithm VIII

1. For $j = 1$ to $z =$ *no. of expression trees* DO

BEGIN

Transform all logical transfers for the leaf variables of expression tree j.

Apply algorithm III to determine optimum sequences of alignment functions for expression tree *j*.

Pick *limit* number of optimum sequences for expression tree *j* and for each of them determine optimum sequences for other variables as followed :

For *i*=*j*+1 TO *z* DO

BEGIN

Determine logical transfers for the leaf variables of expression tree *i*.

Transform all the logical transfers of the leaf variables.

Apply algorithm III to determine optimum sequences of alignment functions for expression tree *i*.

Arbitrarily pick an optimum sequence for expression tree *i*.

END;

Keep the optimum sequence (expression tree *j*) that leads to the lowest total cost to the complete program.

If $j \neq z$, determine logical transfers for expression tree *j*.

END;

□

CHAPTER 6

SUMMARY AND CONCLUSIONS

The total execution time of a parallel algorithm on a multicomputer system can be broken down into the actual computation time and the time of interprocessor communication. On an SIMD multicomputer system, the computation time usually is dependent only on the capability of the processing elements, but the communication time is dependent on a number of factors, such as the interconnection network and the data storage scheme. In this thesis, the problem of minimizing the total communication time in parallel computations has been studied for a class of parallel algorithms and a class of interconnection networks.

In chapter 1, the entire SIMD system, including the relevant components from both hardware and software, is characterized. Based on the characterization, the communication cost minimization problem is formulated.

In chapter 2, a representation scheme for Δ, a class of important permutations with which this thesis is concerned, is developed. Based on the representation scheme, criterion for a permutation in Δ to be realizable in one pass through the omega network is determined. A method for static mapping of a single vector variable onto an SIMD multicomputer system with an omega network is developed. The method is useful for a large class of networks that are functionally equivalent to the omega network. However, it requires solving a system of nonlinear equations.

In chapter 3, an SIMD multicomputer system with an omega network is assumed. Given a parallel program where precedence relationship between its expression trees does not exist, the program is partitioned into independent subproblems, each specifying all the communication requirements for a different vector variable in the parallel program. An optimum nonstatic data mapping algorithm for such subproblems is developed. Applying the algorithm to each subproblem yields data mapping functions that map the program optimally onto the SIMD system. For parallel programs where precedence relationship between expression trees exists, a heuristic algorithm that is based on the optimum data mapping algorithm is developed.

In chapter 4, an SIMD multicomputer system with an omega network is also assumed. Given a parallel program where precedence relationship between its expression trees does not exist, the program is partitioned into independent subproblems, each containing a different expression tree in the parallel program. A data alignment algorithm that will minimize the communication cost associated with an expression tree is developed. Applying the algorithm to each subproblem again yields an optimum solution to the communication cost minimization problem. For parallel programs where precedence relationship between expression trees exists, a heuristic algorithm that is based on the optimum alignment algorithm is also developed.

The two heuristic algorithms developed in chapters 3 and 4 are found to generate optimal or near-optimal solutions for most of the example problems. In fact, optimal solutions are obtained for the matrix inversion program and the FFT program presented in chapter 3.

In chapter 5, it is proved that the algorithms developed in chapters 3 and 4 are also adaptable for any SIMD multicomputer system interconnected with a network from Γ.

In conclusion, the problem of minimizing the interprocessor data communication in parallel computation for a class of parallel algorithms and a class of interconnection networks has been fully investigated. Useful and practical algorithms have been designed for solving the problem. The methodologies developed are widely applicable to many parallel algorithms and many useful interconnection networks. Instead of using two approaches simultaneously for a parallel program and then choosing the one that gives lower total communication cost, one may choose to use only one of the approaches.

Suggestions for Further Research

The next logical step to this research is to develop a representation scheme for a larger class of permutations. With the new representation scheme, one may then develop techniques to minimize the communication costs of a more general class of parallel algorithms and possibly a more general class of interconnection networks. Based on the current representation scheme, we also can develop techniques for other networks that are functionally equivalent to the omega network.

The question of whether given a set of permutations, there exists a data mapping function such that each of the permutations in the set can be realized in one or zero pass is still open. Some necessary conditions have been established and are detailed in the appendix. However, sufficient conditions have not yet been established.

APPENDIX

APPENDIX

DATA MAPPING FUNCTION FOR A GIVEN SET OF PERMUTATIONS

The question of whether given a set of permutations, $\underline{P}$, for a vector variable v, there exists a data mapping function for v such that each of the permutations in $\underline{P}$ can be realized in one or zero pass through an omega network is still open. However, in the following theorems, some necessary conditions are established.

Theorem A.1

With the mapping function F, the set of all permutations $\underline{P_F}$, that can be realized in one or zero pass through an omega network is given by

$$\underline{P_F} = \{ \mathbf{L}_a\mathbf{U}_b\mathbf{F} \mid \mathbf{L}_a \in \underline{\mathbf{L}} \ \& \ \mathbf{U}_b \in \underline{\mathbf{U}} \}.$$

The cardinality of $\underline{P_F}$ is 2^{n^2}.

Proof :

Clearly, $\mathbf{P}_i\mathbf{F}^{-1}$ is in $\underline{P_F}$ iff $\mathbf{P}_i\mathbf{F}^{-1}$ is **LU** decomposable. If $\mathbf{P}_i = \mathbf{LUF}^{-1}$, then $\mathbf{P}_i\mathbf{F}^{-1}$ is **LU** decomposable. Let $\mathbf{P}_i\mathbf{F}^{-1} = \mathbf{L}_c\mathbf{U}_d$. Then $\mathbf{P}_i = \mathbf{L}_c\mathbf{U}_d\mathbf{F}$. Therefore,

$$\underline{P_F} = \{ \mathbf{L}_a\mathbf{U}_b\mathbf{F} \mid \mathbf{L}_a \in \underline{\mathbf{L}} \ \& \ \mathbf{U}_b \in \underline{\mathbf{U}} \}.$$

Since there are $2^{\frac{(n^2-n)}{2}}$ **U**'s and $2^{\frac{(n^2+n)}{2}}$ **L**'s, $| \underline{P_F} | = 2^{n^2}$.

Corollary A.1.1

Given $\underline{P}$, a set of permutations for v, if $|\underline{P}|$ is greater than 2^{n^2}, then there does not exist a data mapping function for v such that every permutations in $\underline{P}$ can be realized in one or zero pass through an omega network.

□

The following two basic lemmas can be obtained easily from linear algebra. Hence, we merely state them here.

Lemma A.1

The set of all lower triangular unit matrices, $\underline{\mathbf{L}}$, is a subgroup of Δ.

□

Lemma A.2

The left coset relation, R_L, defined by $R_L = \{ < \mathbf{P}_1, \mathbf{P}_2 > \mid P_1 \in \underline{\mathbf{L}}P_2 \}$, is an equivalence relation on Δ.[1]

□

R_L partitions Δ into equivalence classes. It is also clear that if $[P_i]$ denotes the equivalence class containing $P_i \in \Delta$, then $[P_i]$ in Δ equals the coset $\underline{\mathbf{L}}P_i$.

[1] Based on lemma 3.2, Δ is a group under matrix multiplication (modoulo 2).

Theorem A.2

The equivalence relation R_L partitions $\underline{P}_F$ into $2^{\frac{(n^2-n)}{2}}$ equivalence classes of $2^{\frac{(n^2+n)}{2}}$ permutations.

Proof :

Consider $P_i \in \underline{P}_F$. Let $\mathbf{P}_i\mathbf{F}^{-1} = \mathbf{LU}$. Then $\underline{\mathbf{LP}}_i\mathbf{F}^{-1} = \underline{\mathbf{LU}}$ and $\underline{\mathbf{LP}}_i = [P_i] \subseteq \underline{P}_F$. Since $|\mathbf{L}| = 2^{\frac{(n^2+n)}{2}}$, $\underline{\mathbf{LP}}_i = [P_i]$ is a partition of $\underline{P}_F$ containing $2^{\frac{(n^2+n)}{2}}$ permutations. Every permutation in P_F belongs to a partition of size $2^{\frac{(n^2+n)}{2}}$ in $\underline{P}_F$. Therefore, R_L partitions $\underline{P}_F$ into $2^{\frac{(n^2-n)}{2}}$ equivalence classes of size $2^{\frac{(n^2+n)}{2}}$.

Corollary A.2.1

Given $\underline{P}$, a set of permutations for v, if R_L partitions $\underline{P}$ into more than $2^{\frac{(n^2-n)}{2}}$ equivalence classes, then there does not exist a data mapping function for v such that every permutations in $\underline{P}$ can be realized in one or zero pass through an omega network.

□

BIBLIOGRAPHY

BIBLIOGRAHPY

[AgLi78] T. Agerwala & B. Lint, "Communication in parallel algorithms for boolean matrix multiplication," IEEE Parallel Processing Conference 78, 1978, pp. 146-153.

[Ager77] T. Agerwala, "Communication, computation, and computer architecture," 1977 Int'l Communication Conference Rec., Chicago, June 1977.

[Bokh81] S. H. Bokhari, "On the mapping problem," IEEE Trans. Computers, C-30, pp. 207-214, March 1981.

[BoMu75] A. Borodin & I. Munro, "The computational complexity of algebraic and numeric problems," American Elsevier, New York, 1975.

[Budn71] P. Budnik & D. J. Kuck, "The organization and use of parallel memories," IEEE Trans. on Computers, December 1971, pp1566-1569.

[Chen81] K.-W Chen, "Minimization of interprocessor communication in parallel computation," Ph.D. dissertation, U. of Michigan, Ann Arbor, 1981.

[Cool65] J. W. Cooley & J. W. Tukey, "An algorithm for the Machine Calculation of complex Fourier series," Mathematics of Computation, Vol. 19, 1965.

[Flyn72] M. Flynn, "Some computer organizations and their effectiveness," IEEE Trans. Computers, C-21 , pp. 948-960, Sept. 1972.

[Gent78] W. M. Gentlemen, "Some complexity results for matrix computations on parallel processors," Journal of the ACM, pp. 112-115, January, 1978.

[Hers64] I. N. Herstein, "Topics in algebra," Lexington, MA, Xerox College, 1964.

[Hell78] D. Heller, "A survey of parallel algorithms in numerical linear algebra," SIAM Review, Vol.20, No. 4 , October 1978, pp740-777.

[IrCh82] K. B. Irani & K.-W. Chen, "Minimization of interprocessors communication for parallel computation," IEEE Trans. Comp., Vol. c-31, #11, November 1982.

[IrCh80] K. B. Irani & K.-W. Chen, "A Jacobi algorithm and its implementation on parallel computers," Proc. of 18th Annual Allerton Conference on Comm., Control, Computing, Oct. 1980.

[IrWu82] K. B. Irani & W. S-F. Wu,"A data mapping methodology for enhancing the capability of a class of multistage interconnection networks", Proceedings of the 1982 Real Time Systems Symposium, Los Angeles, CA, pp. 101-109.

[Kuck78] D. J. Kuck, "The structure of computers and computations," Vol. 1, John Wiley & Sons, 1978.

[KuSt77] H. T. Kung & D. Stevenson, "A software technique for reducing the routing time on a parallel computer with a fixed interconnection network," High Speed Computer and Algorithm Organization, (D.J. Kuck et al. editors), Academic Press, N.Y., pp. 423-433, 1977.

[Kuhn79] R. H. Kuhn, "Efficient mapping of algorithms to single-stage interconnections," 7th Int'l Symp. on Computer Architecture, 1979.

[Kuhn80] R. H. Kuhn, "Transforming algorithms for single-stage and VLSI architectures," 1980 Interconnection Network Workshop, p11-17.

[Lawr75] D. H. Lawrie, "Access and alignment of data in an array processor," IEEE Trans. on Comp., Vol. c-24, #12, December 1975, pp. 1145-1155.

[Lenf78] J. Lenfant, "Parallel permutations of data : A Benes network control algorithm for frequently used permutations," IEEE Trans. Comp., Vol. C-27, #7, July 1978, pp. 637-647.

[LuBa80] S. F. Lundstrom & G. H. Barnes, "A controllable MIMD architecture," IEEE Parallel Processing 1980, pp. 19-27.

[NaSa80] D. Nassimi & Sartaj Sahni, "Parallel algorithms to set-up the Benes permutation network," 1980 Interconnection Network Workshop.

[NaSa81A] D. Nassimi & Sartaj Sahni, "Data broadcasting in SIMD computers", IEEE Trans. Computers, Vol. 2, 1981.

[NaSa81B] D. Nassimi & Sartaj Sahni, "Benes network and parallel permutation algorithms," IEEE Trans. Computers, Vol. 5, 1981.

[Park80] D. S. Parker, "Notes on shuffle/exchange type networks," IEEE Trans. Comp. Vol. C-26, pp 458-473,May 1977.

[Peas68] M. C. Pease, "An adaptation of the fast Fourier transform for parallel processing," Journal of the ACM, Vol. 15, pp. 252-264, Apr. 1968.

[Peas77] M. C. Pease, "The indirect binary n-cube microprocessor Array," IEEE Trans. Comp., c-26, #5, May 1977, pp. 458.

[Sieg78] H. J. Siegel & S. D. Smith, "Study of multistage SIMD interconnection networks," 5th Annual Symp. Computer Archecture, Apr. 1978, pp223-229.

[Smit78] S. D. Smith & H. J. Siegel, "Recirculating, pipelined, and multistage SIMD interconnection Networks," 1978 Parallel Processing Conference.

[Sieg79] H. J. Siegel, "Partitioning permutation networks : the underlying theory," IEEE Parallel Processing, 1979, pp. 175-184.

[Schw80] J. T. Schwartz, "Ultracomputers", ACM Transactions on Programming Language and Systems, Vol. 2, No. 4, October 1980.

[Ston71] H. S. Stone, "Parallel processing with the perfect shuffle," IEEE Trans. Comp., c-20 #2, Feb. 1971, pp. 153-161.

[Swan74] R. C. Swanson, "Interconnections for parallel memories to unscramble p-Ordered vectors," IEEE Trans. on Comp. Nov. 1974.

[ThKu77] C. D. Thompson & H. T. Kung, "Sorting on a mesh-connected parallel computer," Communications of the ACM, April 1977, pp. 263-271.

[WuFe78] C. L. Wu & T. Y. Feng, "Routing techniques for a class of multistage interconnection networks," 1978 Parallel Processin Conference.

[WuFe79] C.L. Wu & T.Y. Feng, "Routing techniques for a class of multistage interconneciton networks," 1979 Parallel Processing Conference, pp 197-205.