A New Network Topology with Multiple Meshes

Debasish Das, Mallika De, and
Bhabani P. Sinha, *Senior Member, IEEE*

Abstract—This paper introduces a new network topology, called **Multi-Mesh (MM)**, which uses multiple meshes as the basic building blocks interconnected in a suitable manner. The proposed network consists of n^4 processors and is 4-regular with a diameter of $2n$. The network also contains a Hamiltonian cycle. Simple routing algorithms for point-to-point communication, one-to-all broadcast, and multicast have been described for this network. It is shown that a simple $n^2 \times n^2$ mesh can also be emulated on this network in $O(1)$ time. Several application examples have been discussed for which this network is found to be more efficient with regard to computational time than the corresponding mesh with the same number of processors. As examples, $O(n)$ time algorithms for finding the sum, average, minimum, and maximum of n^4 data values, located at n^4 different processors have been discussed. Time-efficient implementations of algorithms for solving nontrivial problems, e.g., Lagrange's interpolation, matrix transposition, matrix multiplication, and Discrete Fourier Transform (DFT) computation have also been discussed. The time complexity of Lagrange's interpolation on this network is $O(n)$ for n^2 data points compared to $O(n^2)$ time on mesh of the same size. Matrix transpose requires $O(n^{0.5})$ time for an $n \times n$ matrix. The time for multiplying two $n \times n$ matrices is $O(n^{0.6})$ with an AT-cost of $O(n^3)$. DFT of n sample points can be computed in $O(n^{0.6})$ time on this network. Papers [6], [7] show that n^4 data elements can be sorted on this network in $O(n)$ time.

Index Terms—Mesh, multimesh, diameter, Hamiltonian cycle, point-to-point communication, one-to-all broadcast, multicast, fault-diameter, Lagrange's interpolation, matrix transpose, matrix multiplication, DFT.

1 Introduction

With the advances in VLSI technology, it is now possible to use several thousand processors for constructing a parallel processing system. The processors may communicate, in general, with each other through either a shared memory or an interconnection network. Among the static interconnection networks used for SIMD computers with an array of processors, one of the oldest and very popular architectures is a *two-dimensional-mesh* [11]. Mesh is a simple network with a very regular structure. Also, the fact that the interconnecting wires occupy only a fixed fraction of the area independent of the size of the mesh makes it very attractive for VLSI implementation. These features led to the manufacture of machines, like ILIAC III and IV, SOLOMON, CLIP4, MPP, etc., built around mesh interconnection [11]. Many important algorithms for solving various problems, e.g., matrix operations, simultaneous linear equations, graph-theoretic, and image processing problems, etc. [2], [9], [14], [15], [16], [18] have been efficiently mapped onto this mesh architecture. In a two-dimensional mesh with n^2 processors the degree of $(n-2)^2$ processors is 4, and that of $4(n-2)$ processors is 3, while four corner processors have degree 2. Identifying each processor by two coordinates x and y with respect to some chosen origin, let the processor at the position (x, y) be denoted by $P(x, y)$. The processor $P(x, y)$ is connected to $P(x \pm 1, y \pm 1)$, if they exist, where $1 \leq x, y \leq n$. The diameter of this simple mesh is $2(n-1)$. The interconnection scheme in ILLIAC IV is a little more complicated, with some additional wrap-around and end-around connections, bringing down the diameter to $n-1$. Since the diameter of the mesh is $\Theta(N^{1/2})$, where N is the total number of processors, a lower bound on the time to solve nontrivial problems that involve manipulation of data residing in processors farthest apart in a mesh of size $N(=n^2)$ is $\Omega(N^{1/2})$. In search of an architecture capable of providing faster solutions to such problems, yet retaining most of the attractive properties of a mesh, researchers have studied related interconnection schemes like pyramid [8], [12], mesh-of-trees [13], meshes with broadcast buses [17], [3], etc.

In this paper, we propose a new **Multi-Mesh (MM)** network topology using n^4 processors which is built around n^2 meshes of size $n \times n$ each. The degree of each processor in this MM network is 4, and the diameter of the network is $2n$. The diameter under single node failure is $2n+6$. A Hamiltonian cycle also exists in the network. Algorithms for point-to-point communication, single node broadcast, and multicast have also been developed on this network. Point-to-point communication needs $2n$ communication steps, while one-to-all broadcast and multicast for n^4 processors can be effected in $2n+8$ steps and $n^4+n^3+n^2+n-1$ steps of data communication, respectively. The wormhole routing for complete exchange in the Multi-Mesh network has been done in [5].

We show that an $n^2 \times n^2$ mesh can be emulated on the proposed network in $O(1)$ time. Thus, any algorithm that runs in $O(f(n))$ time in an $n^2 \times n^2$ mesh can always be solved in less than or equal to $O(f(n))$ time. However, this result of emulation merely gives an upper bound on the running time of an algorithm on the MM network. In practice, many real life problems can be solved on the proposed network more efficiently than on the corresponding mesh with the same number of processors. Specifically, the problems whose time complexities are governed by the diameter of the network, i.e., when communications among the farthest processors are necessary for the completion of the algorithms, the MM network offers a distinct advantage over the mesh. As examples of real life applications, simple problems like those of finding the sum, average, minimum, maximum of n^4 data values with $O(n)$ time on the MM network having n^4 processors have been discussed. Note that each of these problems would require $O(n^2)$ time on a simple $n^2 \times n^2$ mesh. Among nontrivial problems, algorithms and their implementations for Lagrange's interpolation, matrix transpose, matrix multiplication, and discrete Fourier transform (DFT) computation have been discussed. These algorithms would cover the basic operations in a large class of numerical problems. The time complexity of Lagrange's interpolation on this network is $O(n)$ for n^2 data points compared to $O(n^2)$ time on mesh. The algorithm for transposing an $n^2 \times n^2$ matrix on the network requires $O(n)$ time. The time complexity for multiplying two $p \times p$ matrices on an MM network with n^4 processors, where $n = p^{0.6}$, is $O(p^{0.6})$, giving an AT-cost of $O(p^3)$. We may note that this problem could be solved on an $p \times p$ mesh in $\Omega(p)$ time, assuming that only the boundary processors can handle the data input/output operations. The DFT of p sample points can also be computed in $O(p^{0.6})$ time on this network with $p^{2.4}$ processors. Also, an algorithm for sorting n^4 elements in $O(n)$ time on this network has been proposed in [6], [7].

The paper is organized as follows. In Section 2, we describe the proposed interconnection scheme. In Section 3, we describe a few topological properties of the network. Section 4 shows how the point to point routing algorithm can be implemented. Sections 5 and 6 present algorithms for one-to-all broadcast and multicast respectively. In Section 7, we show that a simple $n^2 \times n^2$ mesh can be emulated on the MM network in constant time. Section 8 deals with the implementations of different algorithms on the proposed network. Section 9 discusses about the generalized MM network which can be defined with m^2n^2 processors with a diameter of $(m+n)$, where m, n are integers each greater than 2.

- *D. Das and B.P. Sinha are with the Electronics Unit, Indian Statistical Institute, Calcutta 700035, India. E-mail: bhabani@isical.ernet.in.*
- *M. De is with USIC, University of Kalyani, Kalyani 741235, India.*

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 108926.

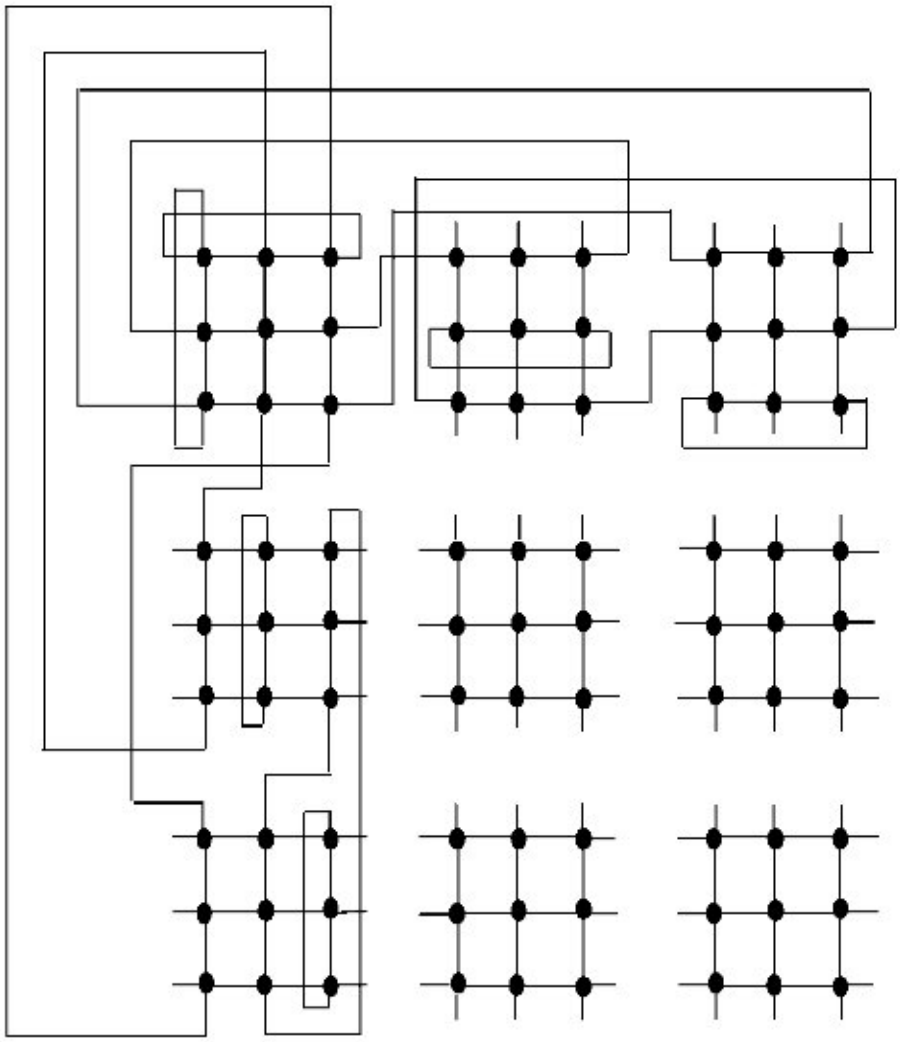

Fig. 1. An example of a multimesh network with n = 3 (all interblock links are not shown).

2 The Multi-Mesh (MM) Network

The proposed Multi-Mesh network is an extension over the simple mesh. In an $n \times n$ mesh, the processors are arranged in n rows and n columns. We use such a mesh as the basic building block of our interconnection scheme. The key idea is to use n^2 such meshes which themselves are again arranged in the form of an $n \times n$ matrix. Each constituent $n \times n$ mesh in this matrix is termed as a *block*. Within each block there are $4(n-2)$ processors on the four outer boundaries each of which has three neighbors within that block. These will be referred to as the *boundary processors*. Also, in each block, there are four corner processors which have only two neighbors within that block. These processors will henceforth be referred to as the *corner processors*. The rest of the $(n-1)^2$ processors in every block, each having 4-neighbors in that block, will be termed as the *internal processors*. We will interconnect different blocks by inserting suitable links among the boundary processors so that each processor will uniformly have four links in the final topology. To describe the interblock connections we need to identify each of the n^4 processors in the MM network uniquely as follows.

A processor inside a given block can be uniquely identified using two coordinates. As the blocks are in turn organized as a matrix, each block can also be uniquely identified using two coordinates α and β as B(α, β). Thus, each of the n^4 processors can be uniquely identified using a 4-tuple of the coordinate values. The first two coordinates are used to describe the block in which the processor lies and the other two coordinates are used to signify the position of the processor inside that specific block. For example, P(α, β, x, y) is a processor lying at the xth row and the yth column of the block B(α, β). Each of these four coordinates may assume a value between 1 and n (both inclusive).

A special symbol $*$ will be used for any one of these four coordinates to denote the set of all processors with all possible values of the respective coordinates. For example, P($*$, $*$, 1, 1) signifies the set of the top left corner processors of all the n^2 blocks.

If the processors P(α, β_1, x_1, y_1) are connected to P(α, β_2, x_2, y_2) for all values of α, $1 \leq \alpha \leq n$, we denote these sets of links by an interconnection between the sets P($*$, β_1, x_1, y_1) and P($*$, β_2, x_2, y_2). In a similar manner, a set of blocks can be represented by putting a $*$ in one or both of the two coordinates of B(α, β).

Interblock connections among the boundary processors are given by the following rules:

1. $\forall \beta, 1 \leq \beta \leq n$, P($\alpha$, β, 1, y) are connected to P(y, β, n, α), where $1 \leq y, \alpha \leq n$, and
2. $\forall \alpha, 1 \leq \alpha \leq n$, P($\alpha$, β, x, 1) are connected to P(α, x, β, n), where $1 \leq x, \beta \leq n$.

All these links are two-way connections. Hence, in the multi-mesh network, all processors have a uniform degree of 4. These interblock connections among the boundary processors will henceforth be referred to as the *interblock links*. Rule 1 interconnects

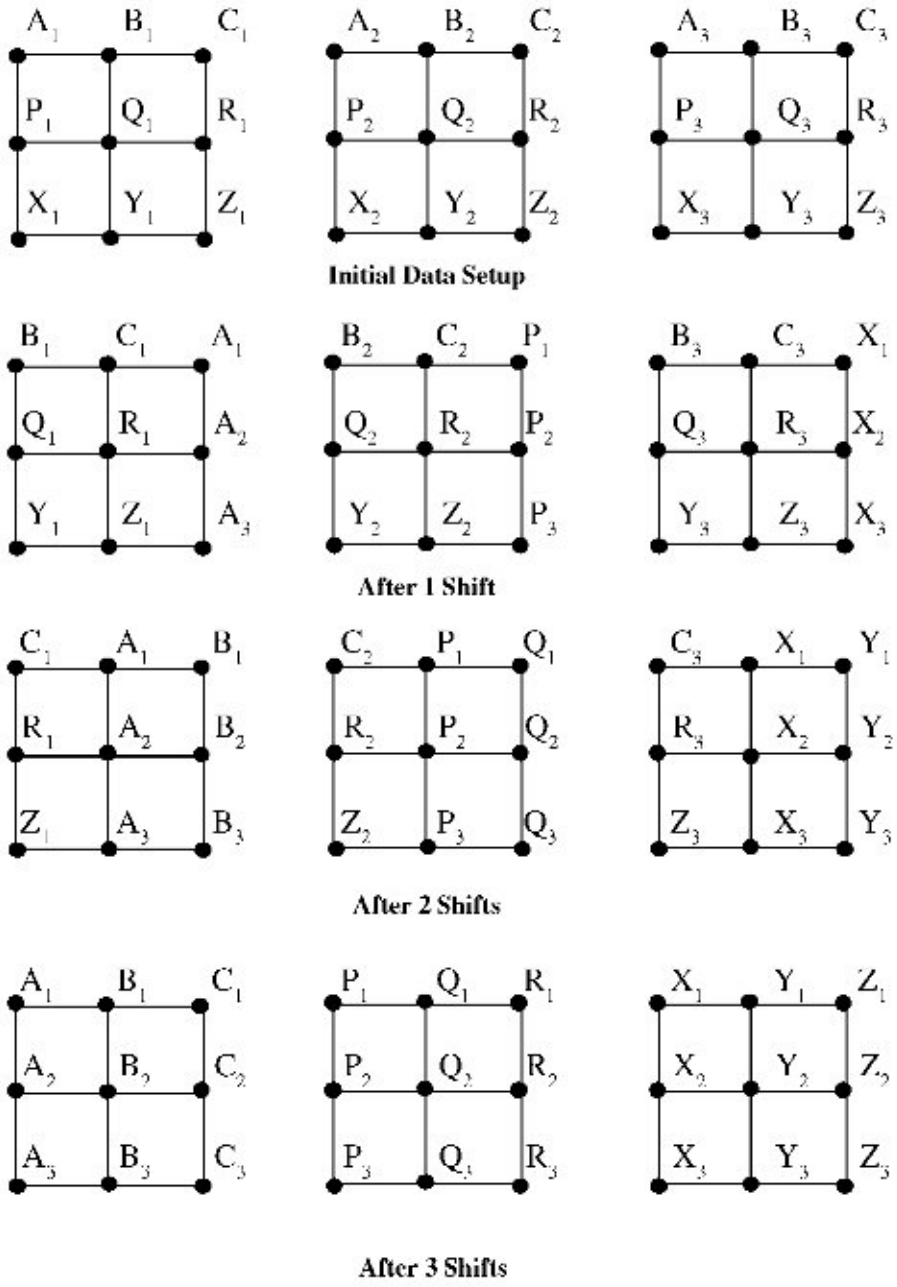

Fig. 2. Data movements along the horizontal cycles.

two blocks in the vertical direction and, so, the corresponding links are called vertical interblock links. Similarly, rule 2 defines the horizontal interblock links.

We note that the processors P(α, $*$, 1, α) are connected to P(α, $*$, n , α) which lie in the same block by means of interblock links. Similarly, the processors P($*$, β, β, 1) are connected to P($*$, β, β, n), again in the same block by interblock links. Thus, we see that four of the boundary processors in each block are connected to some other boundary processors in the same block. It is also to be noted that one of these two connections is in the horizontal direction, while the other is in the vertical direction, which we will specially refer to as the *horizontal wrap-around* connection and the *vertical wrap-around* connection, respectively. An example of a multimesh network for $n = 3$, is given in Fig. 1, where all the interblock links are not shown.

3 Topological Properties

In this section we explore several topological properties of the proposed MM network.

3.1 Cycle Structures

The interblock links induce different cycles in the MM network.

3.1.1 Cycles of Length n

Every wrap-around link in a block induces a cycle of length n. There are two such cycles in each block, one in the horizontal direction and the other in the vertical direction. There are $2n^2$ such cycles of length n in the whole network.

3.1.2 Cycles of Length $2n$

Due to the horizontal interblock links of the MM network, there is a cycle of length $2n$ between the kth row of the block B(i, j) and the

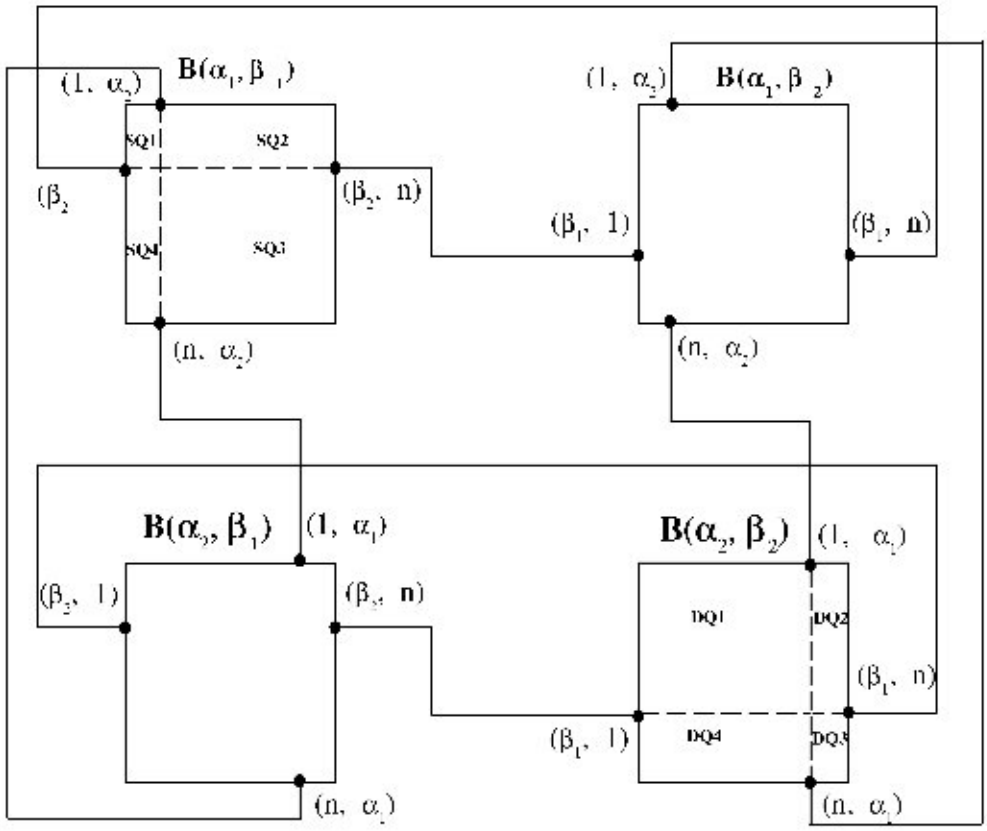

Fig. 3. Possible paths between source and destination blocks.

jth row of the block $B(i,k)$ for $j \neq k, 1 \leq i \leq n$. Thus, there are $n(n-1)/2$ such *horizontal* cycles of length $2n$ in a particular row of processor blocks. Similarly, due to the vertical interblock links of the MM network, there is a cycle of length $2n$ between the kth column of the block $B(i,j)$ and the ith column of the block $B(k,j)$ for $k \neq i, 1 \leq j \leq n$. Also, there are $n(n-1)/2$ such *vertical* cycles in a particular column of processor blocks. In total, there are $n^2(n-1)$ cycles of length $2n$ in the MM network.

The presence of these cycles readily leads to the following results:

Lemma 1. *For a given α, if we shift the data elements in $B(\alpha, *)$ through n positions along the horizontal cycles, then the ith row elements of $B(\alpha, j)$ will move to the jth row of $B(\alpha, i)$.*

Lemma 2. *For a given β, if we shift the data elements in $B(*, \beta)$ through n positions along the vertical cycles, then the ith column elements of $B(j, \beta)$ will move to the jth column of $B(i, \beta)$.*

Example. An example of data movements along the horizontal cycles in a single row of processor blocks is shown in Fig. 2 for $n = 3$.

3.1.3 Hamiltonian Cycle

It has been shown in [4] that there exists a Hamiltonian cycle in the MM network.

3.2 Diameter

Treating Fig. 1 as a representative topology of the proposed MM network, we can see that there exist more than one path between any two processors in the network. We will first show that there exists a path of length less than or equal to $2n$ between any two processors in the network. Next, we will show that there exists a pair of processors such that the shortest distance between them is $2n$.

Let the source processor be designated as $P(\alpha_1, \beta_1, x_1, y_1)$, and the destination processor as $P(\alpha_2, \beta_2, x_2, y_2)$. The block $B(\alpha_1, \beta_1)$, which the source processor lies in, will be referred to as the *source block*. Similarly the block $B(\alpha_2, \beta_2)$ containing the destination processor will be referred to as the *destination block*. From the discussions in Section 2, it is clear that each of the blocks represented by $B(\alpha_1, *)$ and $B(*, \beta_1)$ is directly connected to the source block by one interblock link. Similarly, each of the blocks represented by $B(\alpha_2, *)$ and $B(*, \beta_2)$ is directly connected to the destination block by one interblock link. Hence, we claim that if $\alpha_1 \neq \alpha_2$ and $\beta_1 \neq \beta_2$, we must traverse through at least one intermediate block to reach the destination processor from the source processor. For example, we can use either $B(\alpha_2, \beta_1)$ or $B(\alpha_1, \beta_2)$ as an intermediate block to reach $B(\alpha_2, \beta_2)$ from $B(\alpha_1, \beta_1)$. The situation is clearly explained in Fig. 3.

Theorem 1. *There always exists a path of length less than or equal to $2n$ from any processor $P(\alpha_1, \beta_1, x_1, y_1)$ to another processor $P(\alpha_2, \beta_2, x_2, y_2)$ in the Multi-Mesh network.*

Proof. We join the processors $P(\alpha_1, \beta_1, 1, \alpha_2)$ and $P(\alpha_1, \beta_1, n, \alpha_2)$ by an imaginary vertical line and the processors $P(\alpha_1, \beta_1, \beta_2, 1)$ and $P(\alpha_1, \beta_1, \beta_2, n)$ by an imaginary horizontal line in the source block. These two lines divide the source block into four quadrants, which we name as SQ1, SQ2, SQ3, and SQ4, as shown in Fig. 3 (dotted lines). These four boundary processors in the source block will be referred to as the *source block exits*. Similarly, imaginary lines are drawn in the destination block also, as shown in Fig. 3. Those lines divide the destination block into four quadrants DQ1, DQ2, DQ3, and DQ4. The four boundary processors through which these imaginary lines are drawn will be termed as the *destination block entries*. From Fig. 3, it is apparent that there are four processors in an intermediate block, out of which two are connected to the source block and

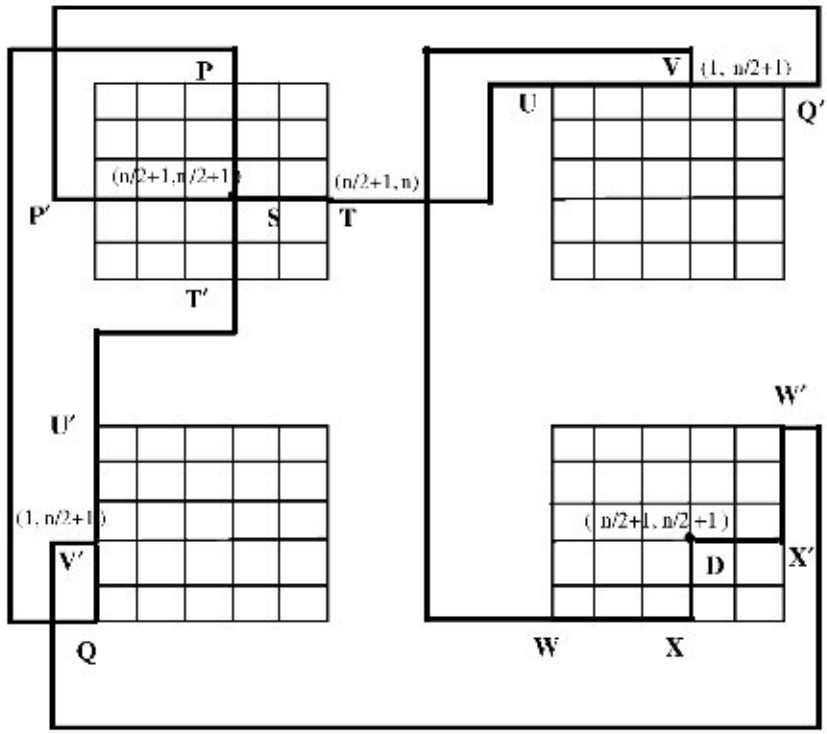

Fig. 4. Four possible paths of length 2n from a P(1, 1, n/2 + 1, n/2 + 1) to P(n/2 + 1, n/2 + 1, n/2 + 1, n/2 + 1) for n = 6.

termed as the *intermediate block entries*, while the other two, connected to the destination block, are termed as the *intermediate block exits*.

Depending on the position of the source and the destination processors in some specific quadrants, we may have 16 different possibilities. Let us first consider one such case, namely, where the source processor lies in the first quadrant SQ1, i.e., $1 \leq x_1 \leq \beta_2$ and $1 \leq y_1 \leq \alpha_2$ and the destination processor lies in DQ1, i.e., $1 \leq x_2 \leq \beta_1$ and $1 \leq y_2 \leq \alpha_1$. The two possible paths PT_1 and PT_2 are considered as follows:

1. $PT_1 : P(\alpha_1, \beta_1, x_1, y_1) \rightarrow P(\alpha_1, \beta_1, \beta_2, 1) \rightarrow P(\alpha_1, \beta_2, \beta_1, n) \rightarrow P(\alpha_1, \beta_2, n, \alpha_2) \rightarrow P(\alpha_2, \beta_2, 1, \alpha_1) \rightarrow P(\alpha_2, \beta_2, x_2, y_2)$.
2. $PT_2 : P(\alpha_1, \beta_1, x_1, y_1) \rightarrow P(\alpha_1, \beta_1, 1, \alpha_2) \rightarrow P(\alpha_2, \beta_1, n, \alpha_1) \rightarrow P(\alpha_2, \beta_1, \beta_2, n) \rightarrow P(\alpha_2, \beta_2, \beta_1, 1) \rightarrow P(\alpha_2, \beta_2, x_2, y_2)$.

The path lengths d_1 and d_2 of PT1 and PT2, respectively, are computed as follows:

$$d_1 = (y_1 - 1) + (\beta_2 - x_1) + 1 + (n - \beta_1) + (n - \alpha_2) + 1 + (x_2 - 1) + (\alpha_1 - y_2).$$

$$d_2 = (x_1 - 1) + (\alpha_2 - y_1) + 1 + (n - \alpha_1) + (n - \beta_2) + 1 + (\beta_1 - x_2) + (y_2 - 1).$$

Since $(d + d_2)/2 = 2n$, it follows that there exists a path of length less than or equal to $2n$.

For the other 15 possible cases of source and destination processor locations in various quadrants, we can also check that [4] there always exists a path of length less than or equal to $2n$.

Next, we show that there exists at least one source - destination pair in the network whose minimum distance is $2n$. For that, let us consider $P(1, 1, \frac{n}{2}+1, \frac{n}{2}+1)$ as the source processor S and $P(\frac{n}{2}+1, \frac{n}{2}+1, \frac{n}{2}+1, \frac{n}{2}+1)$ as the destination processor D as shown in Fig. 4, where n is even. The four possible paths of length $2n$ from the source to the destination are as follows:

Path 1: S T U V W X D
Path 2: S T′ U′ V′ W′ X′ D
Path 3: S P′ Q′ V W X D
Path 4: S P Q V′ W′ X′ D

It has been shown in [4] that, for the other exits from the block B(1, 1), the path length from the source to the destination will not be smaller than $2n$.

For odd n, by taking $P(1, 1, \frac{n+1}{2}, \frac{n+1}{2})$ and $P(\frac{n+1}{2}, \frac{n+1}{2}, \frac{n+1}{2}, \frac{n+1}{2})$ as the source and destination processors, respectively, we can show as above that the minimum path length between them is $2n$. Hence, the theorem. □

3.3 Fault-Diameter

We consider the fault-diameter of the MM network in presence of a single node failure. If the faulty processor is an internal processor of any block, then it can be bypassed by traversing two extra links. If the faulty processor is a boundary processor, then the corresponding interblock link will be affected. For the latter case, we have the following result.

Lemma 3. *If the faulty processor lies on the boundary of some block $B(\alpha, \beta)$, it may increase the path length between any source-destination pair by at most 6.*

Proof. Without loss of generality, let $P(\alpha, \beta, x, n)$ be the faulty processor in $B(\alpha, \beta)$. If the interblock link from the processor $P(\alpha, \beta, x, n)$ to $P(\alpha, x, \beta, 1)$ is included in the path between any source-destination pair, then we can always bypass this link to reach $P(\alpha, x, \beta, 1)$ by detouring in the following way:

$$P(\alpha, \beta, x, n-1) \rightarrow P(\alpha, \beta, x \pm 1, n-1) \rightarrow P(\alpha, \beta, x \pm 1, n) \rightarrow P(\alpha, x \pm 1, \beta, 1) \rightarrow P(\alpha, x \pm 1, \beta \pm 1, 1) \rightarrow P(\alpha, \beta \pm 1, x \pm 1, n) \rightarrow P(\alpha, \beta \pm 1, x, n) \rightarrow P(\alpha, x, \beta \pm 1, 1) \rightarrow P(\alpha, x, \beta, 1).$$

Thus, instead of just two links from $P(\alpha, \beta, x, n-1)$, we need eight links to reach $P(\alpha, x, \beta, 1)$. Hence, the proof. □

Theorem 2. *The diameter of the MM network in the presence of a single node failure is $2n + 6$.*

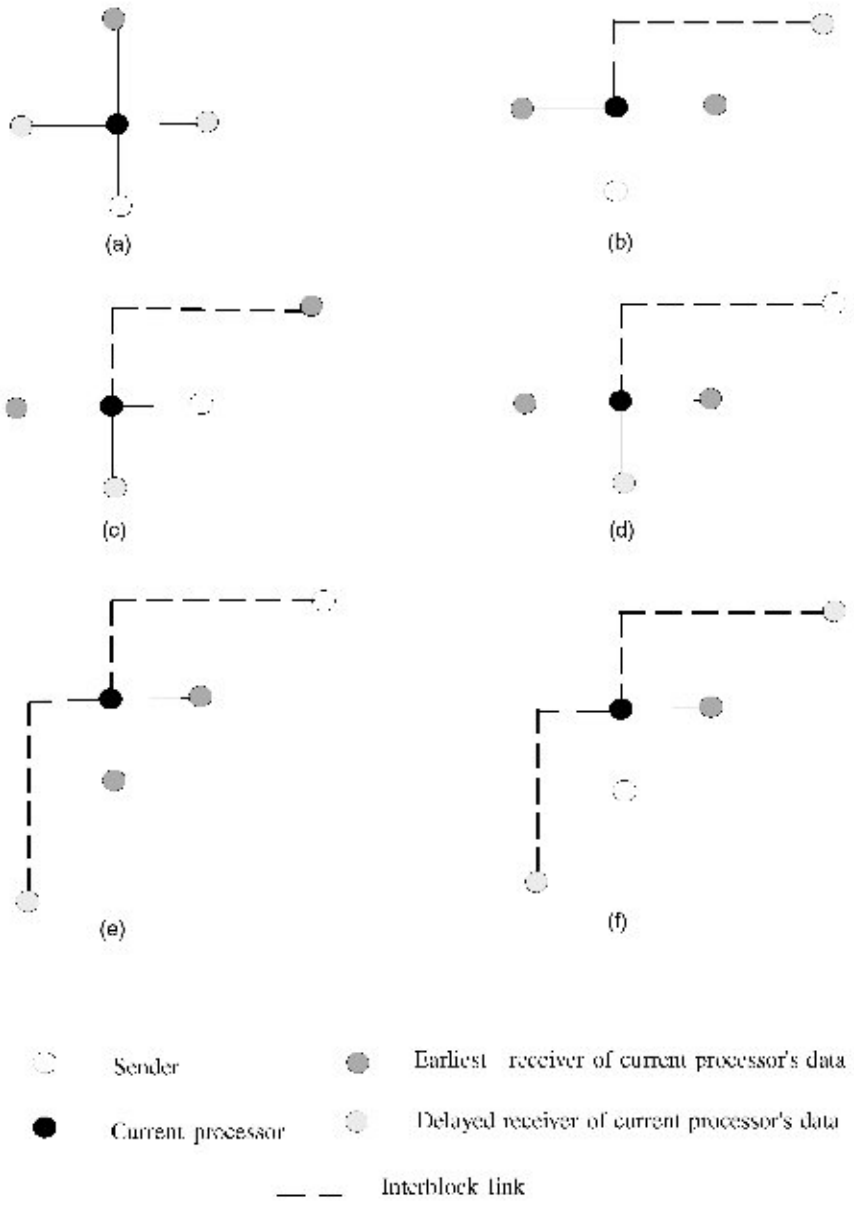

Fig. 5. Priorities of sending data to different processors. (a) Current processor is an internal processor. (b) Current processor is a boundary processor and sender is an internal processor. (c) Current processor is a boundary processor and sender is a boundary processor of the same block. (d) Current processor is a boundary processor and sender is a boundary processor of other block. (e) Current processor is a corner processor and sender is a boundary processor of some other block. (f) Current processor is a corner processor and sender is a boundary processor of the same block.

Proof. Follows from Lemma 3 and Theorem 1. □

Since every processor in the network is of degree 4, any internal processor in a block will be disconnected from the remaining nonfaulty processors if all of its four neighbors are faulty. However, the interconnection between any two blocks of the network is preserved even in presence of $4n-9$ faulty processors. To show this interblock connectivity in the presence of multiple faults, we assume that all the faulty processors appear only on the boundary of the blocks. The total number of boundary processors through which a block is connected to other blocks is $4n-8$ (excluding two wrap-around connections).

If the network has only $4n-9$ faulty nodes and all of them are located at the boundary of a single block, say $B(\alpha,\beta)$, then there must be at least one boundary processor of $B(\alpha,\beta)$ from which an interblock link can be used to go to some other block. Hence, the interblock connectivity is preserved.

If, however, all the $4n-9$ faulty processors are not located at the boundary of the block $B(\alpha,\beta)$, even then each block will be reachable from every other block because the possible number of paths between any two blocks is increased if the $4n-9$ faulty processors are distributed more evenly among different blocks.

4 Point-to-Point Communication Algorithm

The key idea of the point-to-point communication is based on routing the message from the source processor to the destination processor along the restricted path as discussed in the previous section. We describe below the detailed steps for implementing the idea. We will use a NULL identifier in the algorithm all four of whose coordinate values are set to "0." Actually, NULL indicates an invalid processor identifier.

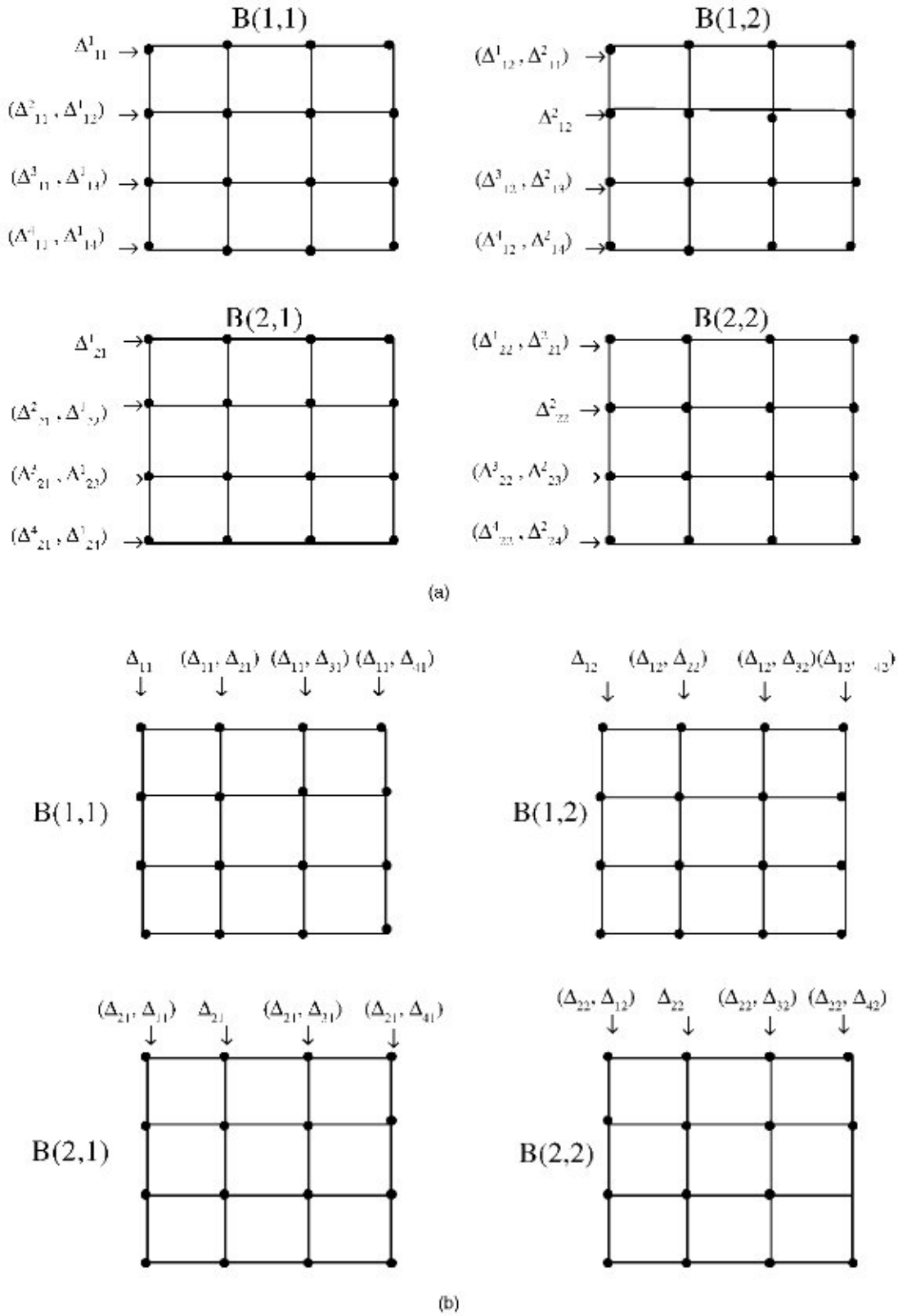

Fig. 6. (a) Contents of different processors after Step 1. (b) Contents of different processors after Step 2.

4.1 Algorithm PP

Step 1: The source processor first determines the quadrant of the source block it lies in. It also finds the quadrant of the destination block in which the destination processor lies. Next, the two possible paths from the source to the destination processor are determined according to the method discussed in the previous section. The two paths are enumerated and the path with the shorter length is chosen. By virtue of Theorem 1, one of these paths must be of length less than or equal to $2n$. The source processor

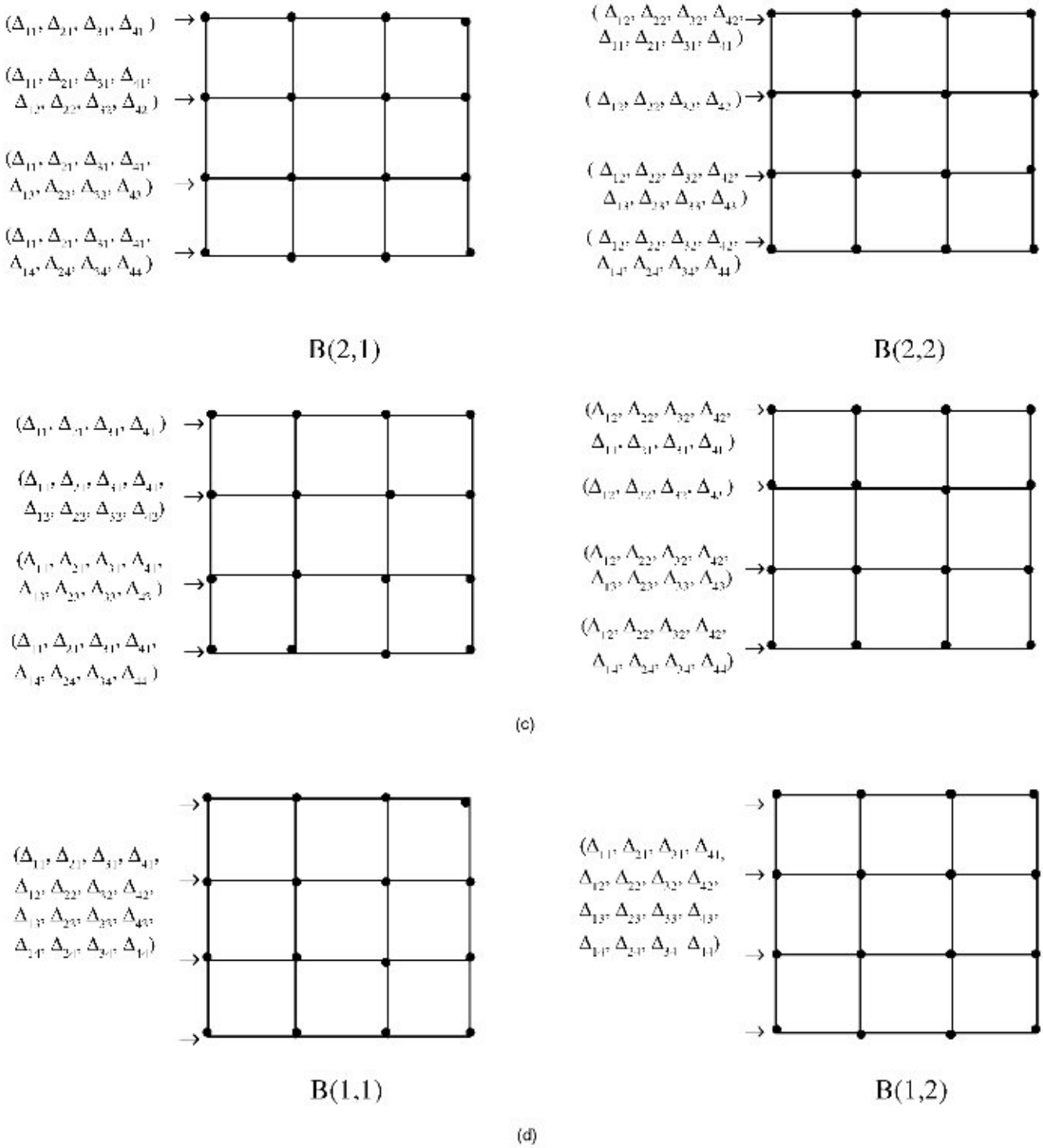

Fig. 6 (continued). (c) Contents of different processors after Step 3. (d) Contents of different processors after Step. 4.

appends three fields, each of four coordinate values, with the data packet to be sent from the source to the destination processor. These appended parts are:

1. Field 1: The identifier coordinates of the source block exit processor.
2. Field 2: The identifier coordinates of the intermediate block exit processor.
3. Field 3: The identifier coordinates of the destination processor.

The augmented data packet is then transmitted along the intrablock links to the source block exit processor.

Step 2: The current processor reads the first field of the augmented part of the recieved data packet. If this identifier is NULL, then go to Step 3. If it is not NULL and not identical with that of the current processor, then the data packet is transmitted along the chosen path toward the processor whose identifier is stored in the first field of the data packet received via one of the intrablock/interblock links of the current processor. Otherwise, the

current processor changes the augmented part of the data packet as follows:

1. field 1 ← field 2;
2. field 2 ← field 3; and
3. field 3 ← NULL.

(Setting any field to NULL will be referred to as *nullification* in our later discussions.)
Repeat step 2.

Step 3: Stop.

It is clear that Step 2 of the algorithm PP will be executed $2n$ times at most.

Example 1. Referring to Fig. 3, let the source processor be located in SQ1 and the destination processor in DQ3. Let the chosen path be $P(\alpha_1, \beta_1, x_1, y_1) \rightarrow P(\alpha_1, \beta_1, 1, \alpha_2) \rightarrow P(\alpha_2, \beta_1, n, \alpha_1) \rightarrow P(\alpha_2, \beta_1, \beta_2, 1) \rightarrow P(\alpha_2, \beta_2, \beta_1, n) \rightarrow P(\alpha_2, \beta_2, x_2, y_2)$. The data packet in the source processor is augmented with the three 4-tuple coordinates as follows

1. $(\alpha_1, \beta_1, 1, \alpha_2)$ as field 1,
2. $(\alpha_2, \beta_1, \beta_2, 1)$ as field 2, and
3. $(\alpha_2, \beta_2, x_2, y_2)$ as field 3.

The first match is obtained at the processor $P(\alpha_1, \beta_1, 1, \alpha_2)$ and, from there, the packet is transmitted along the interblock link to $P(\alpha_2, \beta_1, n, \alpha_1)$ after the first nullification operation. Thus, the first 4-tuple of the augmented part of the data packet, after this step, would contain the identifier of the intermediate block exit processor. The nullification operation is again performed by the processor at $P(\alpha_2, \beta_1, \beta_2, 1)$, where the next match is obtained. After the second nullification operation is executed, the first 4-tuple would contain the identifier of the destination processor. Finally, the last match occurs at the destination processor, which leads to the termination of the algorithm.

5 One-to-All Broadcast

Each of the four neighbors of a processor is connected via its four links which will be referred to as 1) left-link, 2) right-link, 3) up-link, and 4) down-link, respectively. If $P\langle\alpha, \beta, x, y\rangle$ is an internal processor, then it is connected to $P\langle\alpha, \beta, x, y-1\rangle$, $P\langle\alpha, \beta, x, y+1\rangle$, $P\langle\alpha, \beta, x-1, y\rangle$, and $P\langle\alpha, \beta, x+1, y\rangle$ by its left, right, up, and down links, respectively. However, if $P\langle\alpha, \beta, x, y\rangle$ is a boundary processor or a corner processor, then one or two of these links will be interblock links. As an example, the left-link of the boundary processor $P\langle\alpha, \beta, x, 1\rangle$ connects it to $P\langle\alpha, x, \beta, n\rangle$.

We assume here a single-port model for communication, i.e., every processor sends its received data item through only one of its links at a time. The essence of the algorithm is explained as follows:

The four boundary/corner processors situated directly along the up, down, left, and right directions of the source processor $P\langle\alpha, \beta, x, y\rangle$ are $P\langle\alpha, \beta, 1, y\rangle$, $P\langle\alpha, \beta, n, y\rangle$, $P\langle\alpha, \beta, x, 1\rangle$, and $P\langle\alpha, \beta, x, n\rangle$, respectively. Broadcast is started by sending the data from the source processor along its four links successively in the order of the nonincreasing distances from these four boundary/corner processors. Each processor other than the source processor is activated when it receives the data from one of its four neighbors. When a processor receives the data for the first time, it takes the following action:

(A) If the current processor is an internal processor, then its received data is forwarded in the same direction first.
(B) If the current processor is a boundary processor, then data is sent to all the exits of the block at the earliest opportunity. There may be three different cases as given below.

Case 1: Sender is an internal processor.
Case 2: Sender is a boundary processor of the same block.
Case 3: Sender is a boundary processor of some other block.

(C) If the current processor is a corner processor, then there may be two different cases:

Case 1: Sender is a boundary processor of the same block.
Case 2: Sender is a boundary processor of some other block.

The priorities of sending data through different links in all these cases are shown in Figs. 5a through 5f.

The detailed steps of the broadcast algorithm are given in [4] and are omitted here due to brevity.

It can be checked that sending of data in a specific direction can at most be delayed by three units in the source processor. At the boundary of the source block, the data may again be delayed by two units of time after it is received. In the intermediate block, the data may be delayed by at most one time unit to forward it to another block. The delay in the destination block is two time units. Hence, the total delay is no more than eight units of time, i.e., the broadcast can be completed in $2n + 8$ time units starting from the instant of sending the data from the source processor.

6 Multicast

We now describe the algorithm for all-to-all broadcast in which we assume that at a particular instant of time all the processors are transmitting data in the same direction and to only one of the four neighbors.

6.1 Algorithm Multicast

Initially, let the data elements $D(\alpha, \beta, x, y)$ reside in the processors $P(\alpha, \beta, x, y)$ for all values of $\alpha, \beta, x,$ and y.

- **Step 1:** Transmit every data element horizontally along the right links (intrablock or interblock) of the processors successively through $2n - 1$ links (however, the elements $D(*, \beta, \beta, *)$ need to be transmitted only through $n - 1$ successive links).

  Fig. 6a shows the situation for $n = 4$, where Δ_{ij}^k denotes the set of all the n data elements in the k^{th} row of the block $B(i, j)$.
- **Step 2:** For $\beta \neq x$, transmit vertically the set of n data elements $D(\alpha, \beta, x, *)$ currently residing in $P(\alpha, \beta, x, y)$ along the upward links of the processors successively through $2n - 1$ links (however, the data elements $D(\alpha, *, *, \alpha)$ need to be transmitted only through $n - 1$ links).

  Fig. 6b shows only the contents $D(\alpha, \beta, *, *)$ and $D(y, \beta, *, *)$ in each processor after this step, where Δ_{ij} denotes the set of initial data elements in all the processors of the block $B(i, j)$.
- **Step 3:**: The set of n^2 data elements $D(y, \beta, *, *)$ residing in $P(\alpha, \beta, x, y)$ for $\alpha \neq y$ is now transmitted horizontally along the right links successively through the links of the respective horizontal cycles of length n or $2n$.

  Fig. 6c shows the contents of each processor at this point of time.
- **Step 4:** The set of n^3 data elements $D(*, x, *, *)$ received by $P(\alpha, \beta, x, y)$ is now transmitted vertically through $n - 1$ successive links.

  Fig. 6d shows the situation for $n = 4$.

6.2 Time Complexity

Let t_d be the time required for sending a single data element through a link. Then, the total time required for multicast is $(n^4 + n^3 + n^2 + n - 1)t_d$.

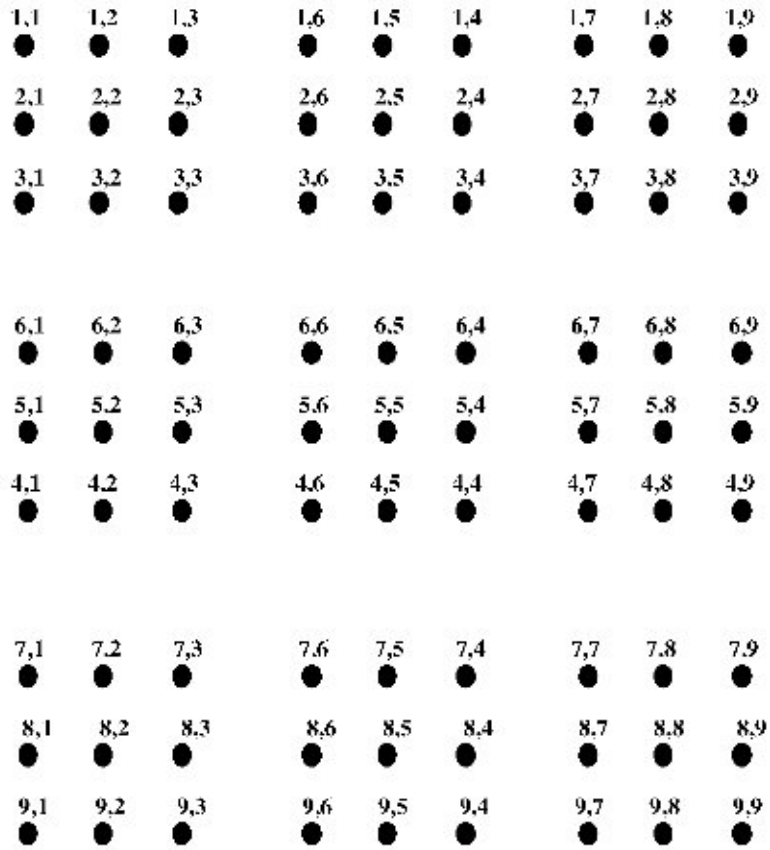

Fig. 7. Initial distribution of elements of a 9×9 matrix on the MM network for emulating a 9×9 mesh.

7 Emulating an $n^2 \times n^2$ Mesh on the MM Network

Many image processing algorithms are mapped onto the mesh architecture because of one useful property of the mesh. In a simple $n^2 \times n^2$ mesh, a processor designated as $P(x, y)$ can know the data contained in four of its neighbors namely, $P(x \pm 1, y \pm 1)$, $1 \leq x, y \leq n^2$, in constant time, which is a primitive requirement for most of the image processing algorithms. Henceforth, we will refer to this property as the 4-neighbor property of a mesh. It may appear at first sight that this 4-neighbor property is destroyed in the MM network because of the boundary processors of each block. An example will help in understanding the problem more clearly. In the MM network, the processors $P(*, *, x, n)$ are directly connected to the processors $P(*, *, x \pm 1, n)$ by two intrablock links in the vertical direction, for $x \neq 1$ or n. They are also connected to the processors designated as $P(*, *, x, n-1)$ by an intrablock link in the horizontal direction. But, the boundary/corner processors of a block are not adjacent to the corresponding processors of the adjacent block. This restriction could be a negative point for using the MM network for implementation of image processing algorithms. However, in this section, we will show that, for odd β, the processors $P(*, \beta, x, n)$ and $P(*, \beta+1, x, n)$, $\beta < n$, can exchange this data in three steps, i.e., $O(1)$ time. Thus, for odd β, $P(*, \beta, x, n)$ and $P(*, \beta+1, x, n)$, $1 \leq \beta < n$, are, in a sense, neighbors of each other. During the same time, for even β, the processors $P(*, \beta, x, 1)$ and $P(*, \beta-1, x, 1)$, $1 < \beta \leq n$, can simultaneously exchange their corresponding data. Similarly, the boundary/corner processors in the vertical direction can also exchange their data in the same amount of time. Thus, the 4-neighbor property of the mesh interconnection is emulated by the MM network in constant time. An example showing the initial distribution of data to preserve desired adjacency among the different data elements on an MM network with $n = 3$ is shown in Fig. 7. The algorithm for such interblock communication in the horizontal direction in $O(1)$ time is presented below:

7.1 Algorithm E

Step 1: The processors identified as $P(*, *, x, n)$ send their data elements to $P(*, x, *, 1)$ using the interblock links in the horizontal direction. At the same time, $P(*, *, x, 1)$ send their data to the processors $P(*, x, *, n)$, $\forall \beta$, $1 \leq \beta \leq$ n.

Step 2: The processors $P(*, x, \beta, 1)$ exchange their data with the processors $P(*, x, \beta+1, 1)$, for $\beta = 1, 3, 5, \cdots$. This can be done because each of the corresponding two processors is directly connected by an intrablock link. Similarly, the processors $P(*, x, \beta, n)$ exchange their data with the processors $P(*, x, \beta+1, n)$, for $\beta = 2, 4, 6, \cdots$.

Step 3: The processors $P(*, x, *, 1)$ transmit their data to $P(*, *, x, n)$ using the interblock links in the horizontal direction and then stop. Similarly, the processors $P(*, x, *, n)$ transmit their data to $P(*, *, x, 1)$ and then stop.

This result is significant in the sense that any algorithm that runs in $O(f(n))$ time in a simple $n^2 \times n^2$ mesh will also run in order $O(g(n))$ $(\leq O(f(n)))$ time in the MM network.

8 Application Examples

It should be noted that the result of the emulation merely gives us an upper bound on the order of the running time of the algorithm. However, algorithms for many real-life problems can be suitably restructured and then mapped on the MM network, resulting in a time complexity which will be of much lesser order than that when implemented on an $n^2 \times n^2$ mesh. This restructuring should

exploit the advantage offered by the interblock links which result in the reduced diameter of the MM network. We illustrate this point by giving below a few examples of typical real-life applications.

Some of the algorithms implemented on mesh require data communication among the diametrically opposite processors at least once. In such situations, the diameter of the network will play the major role in determining the lower bound on the running time. Such algorithms can be easily implemented on the MM network, giving much smaller lower bounds on the order of the running time. For example, finding the sum, average, minimum, maximum, etc., of n^4 data points, using an $n^2 \times n^2$ mesh, cannot be accomplished in less than $O(n^2)$ time units. However, using the MM network, they can be executed in $O(n)$ time units only. Lagrange's interpolation is an example of one nontrivial problem which also falls in this category. Matrix transpose is another problem of this type.

On the other hand, there exist some algorithms for the mesh where the most distantly located processors may not need to communicate with each other or the total number of computations needing to be done overrides this communication time. Matrix multiplication is a representative example of such algorithms. Even in such cases, it is possible to achieve an improvement on the time complexity by using the MM network, by suitably modifying the corresponding algorithm for the mesh. As an example, we will show, in this section, how two $p \times p$ matrices can be multiplied by using the MM network of a suitable size, in $O(p^{0.6})$ time. Using the idea of matrix multiplication, discrete Fourier transform of p data points can also be computed in $O(p^{0.6})$ time.

8.1 Summation /Average /Minimum /Maximum Problem

We first discuss the problem of summing up n^4 data values in n^4 processors. The problem of finding the average, minimum and maximum of n^4 elements can also be similarly dealt with.

We assume that each processor will have two registers, referred to as the H and V registers, for data communication in the horizontal and vertical directions, respectively. For temporary storage of the data received along one of these links, it uses two more registers, denoted by T_h (for the horizontal direction) and T_v (for the vertical direction). Each processor will be assumed to have two other temporary registers T1 and T2. It may be noted that all these registers are not needed for the implementation of this algorithm.

The algorithm is presented below:

Algorithm S
Step 1:
$\forall \alpha, \beta$ and y, $1 \leq \alpha, \beta, y \leq n$ **do in parallel**
  for $i = n-1$ downto 1 do
    $V(\alpha, \beta, i, y) \leftarrow V(\alpha, \beta, i+1, y) + V(\alpha, \beta, i, y)$;
  /* $P(\alpha, \beta, 1, y)$ contains the partial sum of n values */
  $H(\alpha, \beta, 1, y) \leftarrow V(\alpha, \beta, 1, y)$;
  for $j = n-1$ downto 1 do
    $H(\alpha, \beta, 1, j) \leftarrow H(\alpha, \beta, 1, j+1) + H(\alpha, \beta, 1, j)$;
/* Summing along the first row in each block, the sum of the n^2 data values of the block is finally brought to the processor P(α, β, 1, 1) */
  $V(\alpha, \beta, 1, 1) \leftarrow H(\alpha, \beta, 1, 1)$;
  $V(1, \beta, n, \alpha) \leftarrow V(\alpha, \beta, 1, 1)$;
/* Using the vertical interblock links the partial sums of the blocks B($*, \beta$) are transmitted to the nth of the block B($1, \beta$) */
Step 2:
$\forall \beta$, $1 \leq \beta \leq n$ **do in parallel**
  $H(1, \beta, n, \alpha) \leftarrow V(1, \beta, n, \alpha)$;
  for $j = n-1$ downto 1 do
    $H(1, \beta, n, j) \leftarrow H(1, \beta, n, j+1) + H(1, \beta, n, j)$;
/* $P(1, \beta, n, 1)$ now contains the partial sum of the n^3 elements which resided in the n^3 processors of n blocks making a column of the block matrix */
  $V(1, \beta, n, 1) \leftarrow H(1, \beta, n, 1)$;
  $V(1, \beta, 1, 1) \leftarrow V(1, \beta, n, 1)$; / * using the vertical wrap-around connection * /
  $H(1, \beta, 1, 1) \leftarrow V(1, \beta, 1, 1)$;
  $H(1, 1, \beta, n) \leftarrow H(1, \beta, 1, 1)$;
/* The partial sums are brought to the nth column of the block B(1, 1) */
Step 3:
  $V(1, 1, \beta, n) \leftarrow H(1, 1, \beta, n)$;
  for $i = n-1$ downto 1 do
    $V(1, 1, i, n) \leftarrow V(1, 1, i+1, n) + V(1, 1, i, n)$;
/* The final result is computed in P(1, 1, 1, n) */
  $H(1, 1, 1, n) \leftarrow V(1, 1, 1, n)$;
/* Using a horizontal wrap-around connection, the final result is brought to P(1, 1, 1, 1) */
  $H(1, 1, 1, 1) \leftarrow H(1, 1, 1, n)$;

The timing analysis of the algorithm presented above can readily be done. Let each assignment (communication) operation take t_c time units and one addition operation require t_a time units. However, each "+" operation shown in the algorithm consists of one communication and one addition suboperation, making the total time required to be (t_c + t_a). Hence, Step 1 takes $(2n+1)t_c + 2(n-1)t_a$ time units. Steps 2 and 3 take $(n+4)t_c + (n-1)t_a$ and $(n+2)t_c + (n-1)t_a$ time units, respectively. Hence, the total time taken is no more than $(4n+7)t_c + 4(n-1)t_a$ time units. That is, the algorithm runs in $O(n)$ time.

If we want to add kn^4 elements, then, at the very beginning, k elements are to be supplied and added in each processor requiring $kt_c + (k-1)t_a$ time units. The other steps of the algorithm remain the same.

8.2 Lagrange's Interpolation

Let $v_1, v_2, \cdots, v_N$ be the given values of $F(u)$ at $u_1, u_2, \cdots, u_N$ respectively, i.e., $v_i = F(u_i)$. Then, $F(u)$ at the value $\overline{u}$ can be interpolated using the N-point Lagrange's interpolation formula [10] as follows:

$$F(\overline{u}) = \pi(\overline{u}) \sum_i [v_i / \{(\overline{u} - u_i)\pi'(u_i)\}],$$

where

$$v_i = F(u_i), \pi(\overline{u}) = (\overline{u} - u_1)(\overline{u} - u_2)(\overline{u} - u_3) \cdots (\overline{u} - u_N),$$

and

$$\pi'(u_i) = (u_i - u_1)(u_i - u_2) \cdots (u_i - u_{i-1})(u_i - u_{i+1}) \cdots (u_i - u_N).$$

8.2.1 *Parallel Implementation Using the MM Network*

We would use an MM network of n^4 processors for $N(= n^2)$-point Lagrange's interpolation. The basic idea of our proposed algorithm is as follows:

Initially, the data elements $u_{(\beta-1)n+\alpha}$ and $u_{(\alpha-1)n+\beta}$ are fed to the processors $P(\alpha, \beta, n, 1)$ and $P(\alpha, \beta, 1, n)$, respectively, $\forall \alpha, \beta$, $1 \leq \alpha, \beta \leq n$. Also, $\overline{u}$ is fed to the processor P(1, 1, 1, 1). $v_{(\alpha-1)n+\beta}$ are fed to the processors $P(\alpha, 1, \beta, 1)$, $\forall \alpha, \beta$, $1 \leq \alpha, \beta \leq n$.

The differences $(\overline{u} - u_i)$s are computed at the diagonal processors of the diagonal blocks, which are partially multiplied in each block. These are then brought to a single block by using the interblock links, the product term $\pi(\overline{u})$ is computed there, and stored in $P(1, 1, 1, 1)$.

Similarly, each of the differences $(\overline{u} - u_i), (u_i - u_1)$, etc., required for evaluating $(\overline{u} - u_i)\pi'(u_i)$ for every i, $i = 1, 2, \cdots, n$, is computed in a separate processor. These differences are then partially multiplied in each block and then brought to a single block for final multiplication by using the interblock links. $\forall i, v_i$ is then divided by this product $(\overline{u} - u_i)\pi'(u_i)$. By using the interblock links again, these results are brought to a single block, summed up there, and then multiplied by $\pi(\overline{u})$ to give the final interpolated value. The detailed steps of the algorithm, given in [4], run in $O(n)$ time.

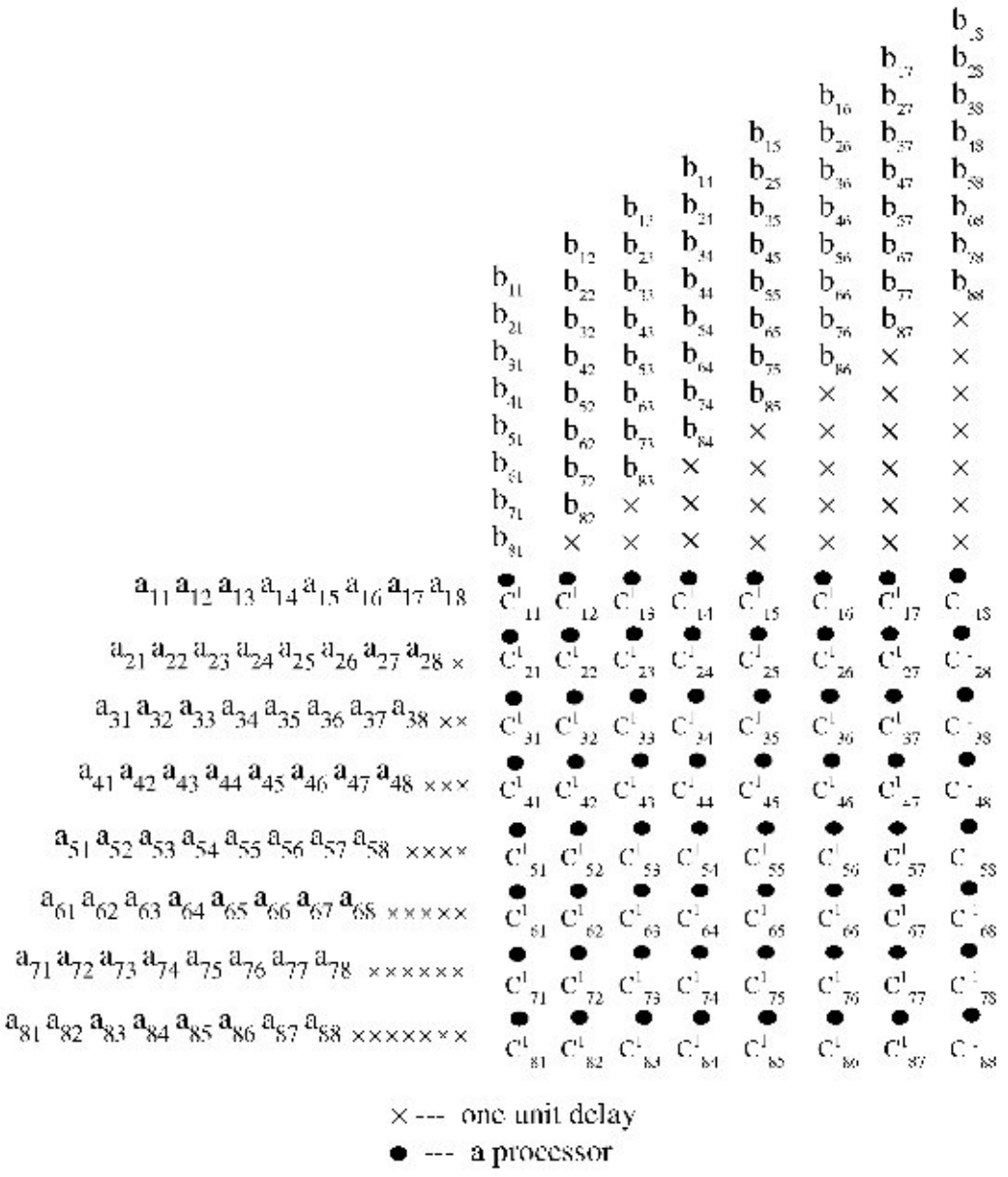

Fig. 8. Computation of various partial products in the block B(1, 1).

If the number of given points is increased by p times, i.e., the values of $F(u)$ are supplied at $pN(=pn^2)$ different points, then the algorithm can also be suitably mapped on the MM network to run in $O(p^2n)$ time [4].

As the number of points is increased from N to pN, the number of terms to be multiplied in computing $\pi(\overline{u})$ and $\pi'(u_i)$ will also be increased by p times. Moreover, the number of terms to be added to evaluate $F(\overline{u})$ will also be increased from N to pN. But, all the terms have to be accommodated in the n^4 processors of the MM network. This can be done in the following way: The pN input values are grouped in p sets:

$$\{u_1, u_2, \cdots, u_N\}, \{u_{N+1}, u_{N+2}, \cdots, u_{2N}\}, \cdots \{u_{(p-1)N+1}, \cdots, u_{pN}\}.$$

For a given input set to the N rows, the columns are successively fed with possible input sets and, each time, the required product terms are evaluated. Then, the input sets to the rows are successively changed and the above procedure is repeated to generate the other partial product components and, finally, the interpolated value is found at the processor P(1, 1, 1, 1). The algorithm for N-point Lagrange's interpolation can be suitably modified for these extra computations by using some more extra registers. Complexity of the modified algorithm will be $O(p^2n)$.

8.3 Matrix Transpose

An $n \times n$ matrix can be transposed on a 2D mesh in $2(n-1)$ communication steps [1]. We show below that an $n^2 \times n^2$ matrix can be transposed on the MM network containing n^4 elements in $8n-4$ communication steps. To describe the algorithm for transposing a matrix on the MM network, let us denote the column j of the given matrix to be transposed by:

$$\begin{matrix} C_j^1 \\ C_j^2 \\ \cdots \\ \cdots \\ \cdots \\ C_j^n \end{matrix}$$

where $C_j^k, k = 1, 2, \cdots$ is again a column of n elements

$a_{(k-1)n+1,j}, a_{(k-1)n+2,j}, \cdots a_{(k-1)n+n,j}$. Similarly, the ith row of the matrix can be denoted by:

$$R_i{}^1\ R_i{}^2 \cdots R_i{}^n,$$

where $R_i{}^k, k = 1, 2, \cdots$ is again a row of n elements

$$a_{i,(k-1)n+1}, a_{i,(k-1)n+2}, \cdots a_{i,(k-1)n+n}.$$

For example, the $4^2 \times 4^2$ matrix in terms of $C_j{}^k$ can be viewed as

$$\begin{array}{cccc|cccc}
C_1{}^1 & C_2{}^1 & C_3{}^1 & C_4{}^1 & C_5{}^1 & C_6{}^1 & C_7{}^1 & C_8{}^1 \\
C_1{}^2 & C_2{}^2 & C_3{}^2 & C_4{}^2 & C_5{}^2 & C_6{}^2 & C_7{}^2 & C_8{}^2 \\
C_1{}^3 & C_2{}^3 & C_3{}^3 & C_4{}^3 & C_5{}^3 & C_6{}^3 & C_7{}^3 & C_8{}^3 \\
C_1{}^4 & C_2{}^4 & C_3{}^4 & C_4{}^4 & C_5{}^4 & C_6{}^4 & C_7{}^4 & C_8{}^4 \\
\hline
C_9{}^1 & C_{10}{}^1 & C_{11}{}^1 & C_{12}{}^1 & C_{13}{}^1 & C_{14}{}^1 & C_{15}{}^1 & C_{16}{}^1 \\
C_9{}^2 & C_{10}{}^2 & C_{11}{}^2 & C_{12}{}^2 & C_{13}{}^2 & C_{14}{}^2 & C_{15}{}^2 & C_{16}{}^2 \\
C_9{}^3 & C_{10}{}^3 & C_{11}{}^3 & C_{12}{}^3 & C_{13}{}^3 & C_{14}{}^3 & C_{15}{}^3 & C_{16}{}^3 \\
C_9{}^4 & C_{10}{}^4 & C_{11}{}^4 & C_{12}{}^4 & C_{13}{}^4 & C_{14}{}^4 & C_{15}{}^4 & C_{16}{}^4
\end{array}$$

The column elements $C_1{}^1$, $C_2{}^1$, $C_3{}^1$, and $C_4{}^1$ will constitute the initial contents of the processor columns in block B(1, 1). Similarly, the block B(1, 2) will initially contain the elements of $C_5{}^1$, $C_6{}^1$, $C_7{}^1$, and $C_8{}^1$, and so on. In general, the n^4 elements of the given matrix will be initially stored in n^4 processors of the MM network so that the n columns of the block B(i, j) will contain the elements of $C^i_{n(j-1)+1}, C^i_{n(j-1)+2}, \cdots, C^i_{nj}$, respectively. The algorithm for matrix transpose is now given below.

Algorithm T

Step 1: Move the column elements of each block through n steps along the vertical cycles (refer to Lemma 2).

Example. After this step, the 16×16 matrix in the above example takes the form:

$$\begin{array}{cccc|cccc}
C_1{}^1 & C_1{}^2 & C_1{}^3 & C_1{}^4 & C_5{}^1 & C_5{}^2 & C_5{}^3 & C_5{}^4 \\
C_2{}^1 & C_2{}^2 & C_2{}^3 & C_2{}^4 & C_6{}^1 & C_6{}^2 & C_6{}^3 & C_6{}^4 \\
C_3{}^1 & C_3{}^2 & C_3{}^3 & C_3{}^4 & C_7{}^1 & C_7{}^2 & C_7{}^3 & C_7{}^4 \\
C_4{}^1 & C_4{}^2 & C_4{}^3 & C_4{}^4 & C_8{}^1 & C_8{}^2 & C_8{}^3 & C_8{}^4 \\
\hline
C_9{}^1 & C_9^2 & C_9^3 & C_9^4 & C_{13}^1 & C_{13}^2 & C_{13}^3 & C_{13}^4 \\
C_10{}^1 & C_{10}^2 & C_{10}^3 & C_{10}^4 & C_{14}^1 & C_{14}^2 & C_{14}^3 & C_{14}^4 \\
C_11{}^1 & C_{11}^2 & C_{11}^3 & C_{11}^4 & C_{15}^1 & C_{15}^2 & C_{15}^3 & C_{15}^4 \\
C_12{}^1 & C_{12}^2 & C_{12}^3 & C_{12}^4 & C_{16}^1 & C_{16}^2 & C_{16}^3 & C_{16}^4
\end{array}$$

Step 2: Transpose each block in parallel as in a 2D mesh. This will take $2(n-1)$ routing steps.

Example. Denoting the transpose of $C_j{}^k$ by $C^t{}_j{}^k$ (a row vector), the matrix elements of the above example are then redistributed as:

$$\begin{array}{cccc|cccc}
C^t_1{}^1 & C^t_5{}^1 & C^t_9{}^1 & C^t_{13}{}^1 & C^t_2{}^1 & C^t_6{}^1 & C^t_{10}{}^1 & C^t_{14}{}^1 \\
C^t_1{}^2 & C^t_5{}^2 & C^t_9{}^2 & C^t_{13}{}^2 & C^t_2{}^2 & C^t_6{}^2 & C^t_{10}{}^2 & C^t_{14}{}^2 \\
C^t_1{}^3 & C^t_5{}^3 & C^t_9{}^3 & C^t_{13}{}^3 & C^t_2{}^3 & C^t_6{}^3 & C^t_{10}{}^3 & C^t_{14}{}^3 \\
C^t_1{}^4 & C^t_5{}^4 & C^t_9{}^4 & C^t_{13}{}^4 & C^t_2{}^4 & C^t_6{}^4 & C^t_{10}{}^4 & C^t_{14}{}^4 \\
\hline
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

The elements $C^t{}_1{}^1$, $C^t{}_1{}^2$, $C^t{}_1{}^3$, and $C^t{}_1{}^4$ are the current contents of the block B(1, 1), and so on.

Step 3: Move the row elements of each block through n steps along the horizontal cycles (refer to Lemma 1).

Example. The matrix in the above example takes the following form after this step:

$$\begin{array}{cccc}
C^t_1{}^1 & C^t_1{}^2 & C^t_1{}^3 & C^t_1{}^4 \\
C^t_5{}^1 & C^t_5{}^2 & C^t_5{}^3 & C^t_5{}^4 \\
C^t_9{}^1 & C^t_9{}^2 & C^t_9{}^3 & C^t_9{}^4 \\
C^t_{13}{}^1 & C^t_{13}{}^2 & C^t_{13}{}^2 & C^t_{13}{}^4 \\
\hline
C^t_2{}^1 & C^t_2{}^2 & C^t_2{}^3 & C^t_2{}^4 \\
C^t_6{}^1 & C^t_6{}^2 & C^t_6{}^3 & C^t_6{}^4 \\
C^t_{10}{}^1 & C^t_{10}{}^2 & C^t_{10}{}^3 & C^t_{10}{}^4 \\
C^t_{14}{}^1 & C^t_{14}{}^2 & C^t_{14}{}^3 & C^t_{14}{}^4 \\
\hline
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots
\end{array}$$

Step 4: Move the column elements of each block through n steps along the vertical cycles.

Example. It can be checked that, after step 4, the matrix in the above example takes the following form:

$$\begin{array}{cccc}
R_1^1 & R_5^1 & R_9^1 & R_{13}^1 \\
R_1^2 & R_5^2 & R_9^2 & R_{13}^2 \\
R_1^3 & R_5^3 & R_9^3 & R_{13}^3 \\
R_1^4 & R_5^4 & R_9^4 & R_{13}^4 \\
\hline
R_2^1 & R_6^1 & R_{10}^1 & R_{14}^1 \\
R_2^2 & R_6^2 & R_{10}^2 & R_{14}^2 \\
R_2^3 & R_6^3 & R_{10}^3 & R_{14}^3 \\
R_2^4 & R_6^4 & R_{10}^4 & R_{14}^4 \\
\hline
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots
\end{array}$$

Note that the elements $R_1{}^1$, $R_1{}^2$, $R_1{}^3$, and $R_1{}^4$ are now stored in block B(1, 1).

Step 5: Transpose each block in parallel using $2(n-1)$ steps.

Example. The matrix in the above example takes the following form, where $R^t_i{}^k$ is the transpose (a column vector) of $R_i{}^k$:

$$\begin{array}{cccc|cccc}
R^t_1{}^1 & R^t_1{}^2 & R^t_1{}^3 & R^t_1{}^4 & R^t_5{}^1 & R^t_5{}^2 & R^t_5{}^3 & R^t_5{}^4 \\
R^t_2{}^1 & R^t_2{}^2 & R^t_2{}^3 & R^t_2{}^4 & R^t_6{}^1 & R^t_6{}^2 & R^t_6{}^3 & R^t_6{}^4 \\
R^t_3{}^1 & R^t_3{}^2 & R^t_3{}^3 & R^t_3{}^4 & R^t_7{}^1 & R^t_7{}^2 & R^t_7{}^3 & R^t_7{}^4 \\
R^t_4{}^1 & R^t_4{}^2 & R^t_4{}^3 & R^t_4{}^4 & R^t_8{}^1 & R^t_8{}^2 & R^t_8{}^3 & R^t_8{}^4 \\
\hline
\cdots & R^t_{13}{}^1 & R^t_{13}{}^2 & R^t_{13}{}^3 & R^t_{13}{}^4 & & & \\
\cdots & R^t_{14}{}^1 & R^t_{14}{}^2 & R^t_{14}{}^3 & R^t_{14}{}^4 & & & \\
\cdots & R^t_{15}{}^1 & R^t_{15}{}^2 & R^t_{15}{}^3 & R^t_{15}{}^4 & & & \\
\cdots & R^t_{16}{}^1 & R^t_{16}{}^2 & R^t_{16}{}^3 & R^t_{16}{}^4 & & &
\end{array}$$

Step 6: Move the column elements of each block again through n steps along the vertical cycles.

Example. After Step 6, the 16×16 matrix in the above example takes the following form, which is nothing but the transpose of the original matrix:

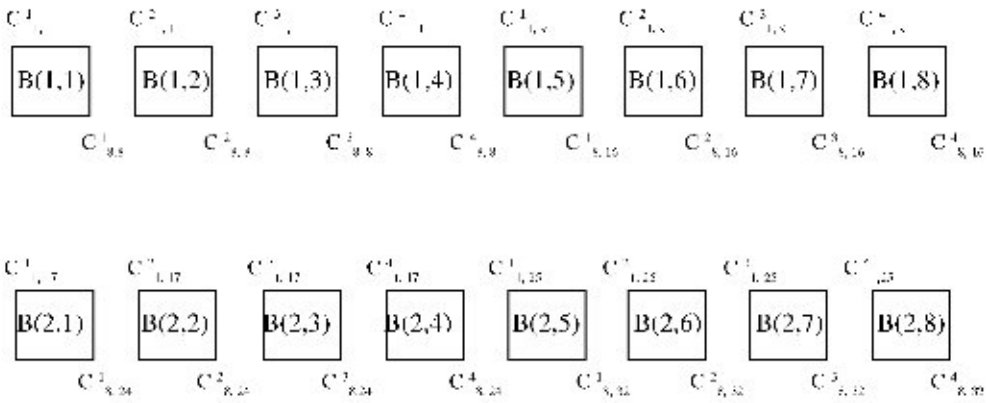

Fig. 9. Computation of partial products in the first and second rows of processor blocks.

$$
\begin{array}{cccc|cccc}
R_1^1{}^1 & R_2^1{}^1 & R_3^1{}^1 & R_4^1{}^1 & R_5^1{}^1 & R_6^1{}^1 & R_7^1{}^1 & R_8^1{}^1 \\
R_1^1{}^2 & R_2^1{}^2 & R_3^1{}^2 & R_4^1{}^2 & R_5^1{}^2 & R_6^1{}^2 & R_7^1{}^2 & R_8^1{}^2 \\
R_1^1{}^3 & R_2^1{}^3 & R_3^1{}^3 & R_4^1{}^3 & R_5^1{}^3 & R_6^1{}^3 & R_7^1{}^3 & R_8^1{}^3 \\
R_1^1{}^4 & R_2^1{}^4 & R_3^1{}^4 & R_4^1{}^4 & R_5^1{}^4 & R_6^1{}^4 & R_7^1{}^4 & R_8^1{}^4
\end{array}
$$

$$
\begin{array}{cccc}
\cdots & R_{13}^1{}^1 & R_{14}^1{}^1 & R_{15}^1{}^1 & R_{16}^1{}^1 \\
\cdots & R_{13}^1{}^2 & R_{14}^1{}^2 & R_{15}^1{}^2 & R_{16}^1{}^2 \\
\cdots & R_{13}^1{}^3 & R_{14}^1{}^3 & R_{15}^1{}^3 & R_{16}^1{}^3 \\
\cdots & R_{13}^1{}^4 & R_{14}^1{}^4 & R_{15}^1{}^4 & R_{16}^1{}^4
\end{array}
$$

Time Complexity: Steps 1, 3, 4, and 6 each require n steps of data communication, whereas Steps 2 and 5 each require $2(n-1)$ routing steps. Therefore, the total time taken to transpose the $n^2 \times n^2$ matrix in an MM network $= 4n + 2.2(n-1) = 8n - 4$ communication steps.

8.4 Matrix Multiplication

We now describe the outline of the algorithm for multiplying two matrices by the MM network. We would multiply two square matrices A and B, each of size $p \times p$, using an MM network with n^4 processors, such that $p^3 = n^5$. The algorithm aims at exploiting the advantages of the interblock links in an appropriate way, resulting in $O(p^{0.6})$ computational time using $p^{2.4}$ processors. We would, however, assume that each block of the MM network can handle input/output operations through one of its row boundaries (e.g., top row boundary) and one column boundary (e.g., left column boundary). It may be noted that two $p \times p$ matrices can be multiplied in a simple 2D mesh in $O(p)$ time. The AT-cost of the proposed algorithm is identical with that of matrix multiplication on a 2D mesh, i.e., $O(p^3)$.

To begin with, let $p = m^5$ and let us consider an MM network containing m^6 blocks, each block being an $m^3 \times m^3$ mesh. In contrast to the matrix multiplication on a 2D mesh, where only one processor is fully responsible for computing one element C_{ij} of the product matrix $C(= A \times B)$, we plan to compute C_{ij} partially in several processors and then sum these parts together. To be precise, C_{ij} is broken down into m^2 parts denoted by C_{ij}^1, C_{ij}^2, $C_{ij}^3, \cdots, C_{ij}^{m^2}$. The scheme is illustrated in Figs. 8 and 9 for $p = 32$, i.e., $m = 2$, and $n = 8$. C_{11} is computed in four parts on four processors P(1, 1, 1, 1), P(1, 2, 1, 1), P(1, 3, 1, 1), and P(1, 4, 1, 1), respectively. The rows of matrix A and the columns of matrix B have been partitioned accordingly to be input to different blocks of processors. Fig. 8 shows the necessary data inputs for the block B(1, 1) of the MM network. The different parts of C_{ij}s, $1 \le i \le 8$, $1 \le j \le 32$, computed by different processors of the first and second rows of blocks have been indicated in Fig. 9 (in the figure, the parts of C_{ij}s computed only by the top left corner processor and the bottom right corner processor of every block have been shown). As shown in Fig. 9, the processor blocks B(1, 1) through B(1, 4) compute C_{ij}s, $1 \le i \le 8$, $1 \le j \le 8$ and the blocks B(1, 5) through B(1, 8) compute C_{ij}s, $1 \le i \le 8$, $9 \le j \le 16$. Similarly, the processor blocks B(2, 1) through B(2, 4) compute C_{ij}s, $1 \le i \le 8$ and $17 \le j \le 24$, while the blocks B(2, 5) through B(2, 8) compute C_{ij}s, $1 \le i \le 8$ and $25 \le j \le 32$. Hence, eight such rows are enough to compute all the C_{ij}s in parallel.

The motivation behind such a partitioning scheme can be understood as follows: Computation of C_{ij} requires m^5 multiplications and m^5 additions. But, computing the partial product C_{ij}^k, $1 \le k \le m^2$, needs only m^3 multiplication and m^3 additions and, to obtain C_{ij} from the C_{ij}^ks, one needs to perform only m^2 additions. If all the C_{ij}^ks can be computed in parallel, each in a different processor, then the time needed for computing C_{ij} will be reduced from $O(m^5)$ to $O(m^3)$.

For clarity of understanding, we have introduced the idea of O-blocks. We divide the output matrix C into m^4 smaller blocks, each of size $m^3 \times m^3$. Each of these blocks will be referred to as an O-block, which contains m^6 elements of the product matrix. An example of such a partitioning into O-blocks has been shown in Fig. 10 for $p = 32$. Each O-block is designated using two coordinates r and s, $1 \le r, s \le m^2$, such that the O-block(r, s) consists of the C_{ij}'s for $(r-1)m^3 + 1 \le i \le rm^3$ and $(s-1)m^3 + 1 \le j \le sm^3$. As described above, all the m^2 different partial products of C_{ij} will be computed by m^2 processors in parallel. Thus, to compute one O-block, we need m^8 processors. One row of blocks in the MM network consisting of $m^6 \times m^3 = m^9$ processors will be able to compute m O-blocks in parallel. There are m^3 rows of blocks in the MM network. Hence, all the m^4 O-blocks can be computed in parallel, in a time upper bounded by the time needed for computing m O-blocks by one row of blocks in the MM network. Again, inside one row, each of these m O-blocks can be computed in parallel. Finally, we conclude that the time taken for obtaining all the elements of the product matrix is upper bounded by the time needed for computing one O-block, i.e., $O(m^3)$.

We shall now examine in detail how the interblock links of the MM network help us in achieving the required parallelism. Any two processors in a row of blocks can communicate between them without using any of the vertical interblock links coming out of those blocks. This leads to the first stage of parallelism. Computation of O-blocks in different rows can be carried out in parallel without interacting with any other row. We use m^2 blocks in B$(\alpha, *)$ for computing one O-block. The next m^2 blocks will compute the next O-block and so on. As an example, let the processor blocks B$(\alpha, 1)$ through B(α, m^2) compute the elements of the O-block(r, s), $1 \le r, s \le m^2$. The partial products C_{ij}^k, $1 \le k \le m^2$, are computed in the processors situated in different blocks. Thus, P(α, β, x, y) computes the partial product

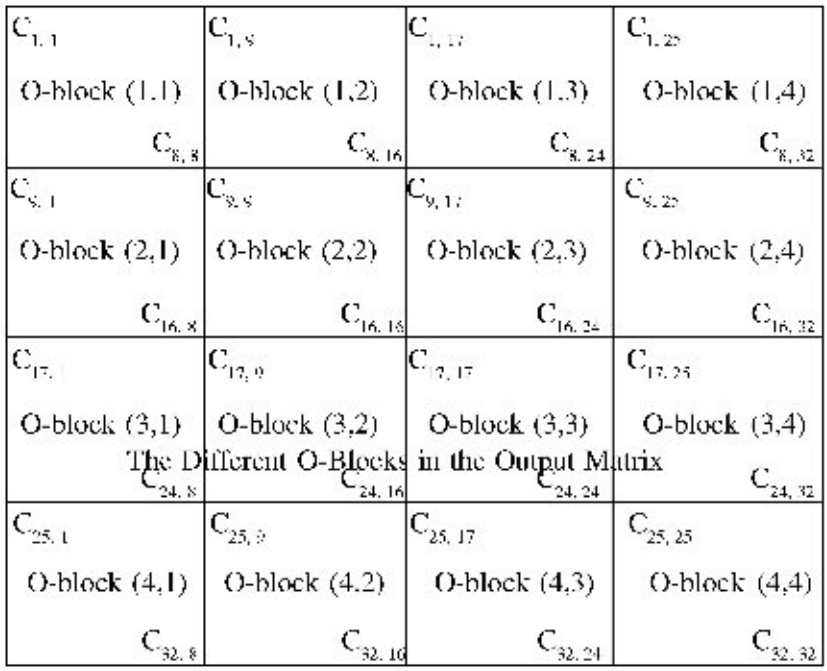

Fig. 10. The different O-blocks in the output matrix.

$C^{\beta}_{(r-1)m^3+x,(s-1)m^3+y}$. Hence, in each block, m^6 different partial products, one each for m^6 output elements, can be computed in parallel.

We need to add these partial products selectively to obtain the output elements. The m^2 partial products needed for computing one output element reside in the processors which lie in m^2 different processor blocks in the same row of blocks. Hence, we need to bring the contents of the m^3 processors, designated as $P(\alpha, \beta, x, y)$, $1 \le \beta \le m^3$, into one column of a processor block. By taking advantage of the horizontal cycles of length n and $2n$, we can accomplish this data transfer in m^3 time. Referring to Fig. 2, as an example, our goal is to bring each of A_is, B_is, C_is, P_is, Q_is, R_is, X_is, Y_is, and Z_is ($1 \le i \le 3$) to a single column. The necessary data movements for bringing the corresponding terms into the same column of a processor block are performed in three communication steps, as shown in the figure.

In general, processor $P(\alpha, \beta, x, y)$ sends its data value to its adjacent processor $P(\alpha, \beta, x, y-1)$ and receives the data sent by the processor $P(\alpha, \beta, x, y+1)$, employing the intrablock link or interblock link, whichever is present. This step is repeated m^3 times by each processor independently. Finally, the processor in a column of a particular block in the MM network designated by $P(\alpha, \beta, x, y)$, $1 \le \beta,\ x, y \le m^3$, contains the content of the processor $P(\alpha, x, \beta, n-y+1)$. Out of the m^3 partial products brought to a specific column, the set of consecutive m^2 partial products are to be added to generate m different $C_{i\,j}$s in each column.

8.4.1 Exact Timing Analysis

In a block $B(\alpha, \beta)$ of the MM network, each of the m^6 processors independently computes one partial product. The time needed for computing all these partial products is $3m^3 - 2$ [1]. The required partial products are brought to the same column of a processor block in another m^3 time units. To generate each $C_{i\,j}$, only (m^2-1) summations are to be performed. Hence, the total time needed will be $(4m^3 + m^2 - 3)$. In other words, we can multiply two $p \times p$ matrices in $O(p^{0.6})$ time using $p^{2.4}$ processors of the MM network. Thus, the AT-cost of the algorithm is $O(p^3)$, which is identical with that in the case of a 2D mesh.

8.4.2 Multiplication of Higher Order Matrices

Two square matrices A and B, each of size $kp \times kp, k > 1$, can also be multiplied by using an MM-network with only n^4 processsors. In this case, C_{ij} is broken down into km^2 parts denoted by C^1_{ij}, $C^2_{ij}, \cdots, C^{km^2}_{ij}$, where $1 \le i, j \le km^5$. Assuming that the output matrix C is partitioned in k^2 different blocks, each of size $p \times p$, the C_{ij}s belonging to each of these k^2 blocks are successively computed in k^2 stages. The km^2 different components of each C_{ij} are computed by m^2 different blocks, as before, in k successive steps.

Example. For $m = 2$ and $k = 2$, each of the matrices A, B, and C is a 64×64 matrix. C_{11} will now have $C^1_{11}, C^2_{11}, \cdots, C^8_{11}$ as the eight components. The blocks B(1, 1) through B(1, 4) will compute C^1_{11} through C^4_{11} in the first step and C^5_{11} through C^8_{11} in the second step, which are summed together to compute C_{11}. Considering all the $k^2(=4)$ blocks of the C matrix, the processor block B(1, 1) will be responsible for successively computing the following components of C_{ij}s in four different stages:

Stage 1: $C^1_{1\,1} \cdots C^1_{8\,8}$; $C^5_{1\,1} \cdots C^5_{8\,8}$
Stage 2: $C^1_{1,33} \cdots C^1_{8,40}$; $C^5_{1,33} \cdots C^5_{8,40}$
Stage 3: $C^1_{33,1} \cdots C^1_{40,8}$; $C^5_{33,1} \cdots C^5_{40,8}$
Stage 4: $C^1_{33,33} \cdots C^1_{40,40}$; $C^5_{33,33} \cdots C^5_{40,40}$

In general, it is thus possible to multiply two $kp \times kp$ matrices in $O(k^3 m^3) = O(k^{2.4}(kp)^{0.6})$ time on an MM network having only $p^{2.4}$ processors.

8.5 Discrete Fourier Transform

Since DFT computation can be formulated in terms of a matrix by vector multiplication, it follows from the above technique for matrix multiplication that the discrete Fourier transform of p sample points can be computed on an MM network containing $p^{2.4}$ processors in $O(p^{0.6})$ time. This can also be compared to $O(p)$ time on a $p \times p$ mesh.

9 Generalized MM Network

The difference between the number of processors of two successive MM networks (i.e., for two consecutive values of n) is

TABLE 1

m	n	Number of Processors	m	n	Number of Processors
3	3	81	4	7	784
3	4	144	5	6	900
3	5	225	4	8	1024
4	4	256	5	7	1225
3	6	324	6	6	1296
4	5	400	5	8	1600
3	7	441	6	7	1764
3	8	576	6	8	2304
4	6	576	7	7	2401
5	5	625	7	8	3136
3	9	729	8	8	4096

$(n+1)^4 - n^4$, which increases as $O(n^3)$. This difference can, however, be reduced if, instead of taking $n \times n$ meshes as the constituent blocks, we take $m \times n$ meshes for any $m, n \geq 3$, arranging mn number of such meshes in the form of an $n \times m$ matrix. The interblock links can be defined in the same manner as in Section 2. The diameter of such a structure can be found to be $m+n$. Table 1 shows the total number of processors (i.e., m^2n^2) for different values of m and n. We see that the number of processors is 576 for both $m=3, n=8$ and $m=4, n=6$. In such cases where the number of processors is same for different pairs of values for m and n, one can choose the structure with m and n closer to each other so that the diameter is minimum among all such possible structures. The algorithms for different types of data communication, as well as for various real-life applications, can be suitably restructured to fit into the generalized version of the MM network.

10 Conclusion

A new topology for processor interconnection, called MM network, with n^4 processors and a uniform node degree of four, has been proposed in this paper. The diameter of the network is $2n$ and, under single node failure, the diameter increases only by 6. The network has a Hamiltonian cycle. Some important communication algorithms, like point-to-point data communication, one-to-all broadcast, and multicast on this network, have been discussed. Wormhole routing for complete exchange in this network has been discussed in [5]. An algorithm for emulating an $n^2 \times n^2$ mesh by the proposed network has also been presented. Simple algorithms for finding the sum, average, minimum, and maximum of n^4 data values, which involve communications among the farthest processors of the network, has been shown to be implemented in $O(n)$ time on this network. Algorithms for nontrivial problems, e.g., Lagrange's interpolation, matrix transpose, matrix multiplication, DFT, have also been efficiently implemented on this network. Another paper [6] shows that n^4 data elements can be sorted on this network in $O(n)$ time.

Acknowledgments

A preliminary small version of some aspects of this work appeared in the *Procedings of the International Parallel Processing Symposium* pp. 17-21, held in Santa Barbara, California, 25-28 April 1995.

References

[1] S.G. Akl, *The Design and Analysis of Parallel Algorithms*. New York: Prentice Hall International, Inc., 1989.
[2] M. Atallah and R. Kosaraju, "Graph Problems on a Mesh Connected Processor Array," *J. ACM*, vol. 31, pp. 649-667, 1983.
[3] S.H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus," *IEEE Trans. Computers*, vol. 33, no. 2, pp. 133-139, Feb. 1984.
[4] M. De, "Design of Efficient Parallel Algorithms and Architectures for Some Numeric and Non-Numeric Problems," PhD thesis, J.U., Calcutta, 1997.
[5] M. De, B. Kundu, and B.P. Sinha, "Wormhole Routing for Complete Exchange in Multi-Mesh," *Proc. Fourth Int'l Conf. High-Performance Computing*, pp. 432-437, Bangalore, India, 18-21 Dec. 1997.
[6] M. De, D. Das, M. Ghosh, and B.P. Sinha, "An Efficient Sorting Algorithm on the Multi-Mesh Network," *IEEE Trans. Computers*, vol. 46, no. 10, pp. 1,132-1,137, Oct. 1997.
[7] M. De, D. Das, M. Ghosh, and B.P. Sinha, "Efficient Sorting on the Multi-Mesh Topology," *Proc. Second Int'l Conf. High-Performance Computing*, pp. 707-712, New Delhi, India, 2-30 Dec. 1995.
[8] C.R. Dyer, "A VLSI Pyramid Machine for Hierarchical Parallel Image," *Proc. IEEE Conf. Pattern Recognition and Image Processing*, 1981.
[9] W.M. Gentleman, "Some Complexity Results for Matrix Computations on Parallel Processors," *J. ACM*, vol. 25, pp. 112-115, 1978.
[10] F.B. Hildebrand, *Introduction to Numerical Analysis*. New York: McGraw-Hill, 1956.
[11] K. Hwang and F.A. Briggs, *Computer Architecture and Parallel Processing*, New York: McGraw-Hill, 1989.
[12] R. Miller and Q.F. Stout, "Data Movement Techniques for the Pyramid Computer," *SIAM J. Computing*, vol. 16, no. 1, pp. 38-60, Feb. 1987.
[13] D. Nath, S.N. Maheshwari, and P.C.P. Bhatt, "Efficient VLSI Networks for Parallel Processing Based on Orthogonal Trees," *IEEE Trans. Computers*, vol. 32, no. 6, pp. 569-581, June 1983.
[14] V.K. Prasanna-Kumar and M. Eshaghian, "Parallel Geometric Algorithms for Digitized Pictures on Mesh of Tree Organization," *Proc. Int'l Conf. Parallel Processing*, pp. 270-273, Aug. 1986.
[15] V.K. Prasanna-Kumar and D. Reisis, "Parallel Image Processing on Enhanced Arrays," *Proc. Int'l Conf. Parallel Processing*, pp. 909-912, Aug. 1987.
[16] C. Savage, J. Jájá, "Fast Efficient Parallel Algorithms for Some Graph Problems," *SIAM J. Computing*, vol. 10, pp. 682-691, 1981.
[17] Q.F. Stout, "Mesh Connected Computers with Broadcasting," *IEEE Trans. Computers*, vol. 32, pp. 826-830, 1983.
[18] C.D. Thompson and H.T. Kung, "Sorting on a Mesh-Connected Parallel Computer," *Comm. ACM*, vol. 20, pp. 263-271, 1977.