

Short Paper

Optimal Algorithm for Matrix Transpose on Wormhole-Switched Meshes*

JYH-JONG TSAY, KUO-SHUN DING[†] AND WEN-TSONG WANG

Department of Computer Science and Information Engineering

National Chung Cheng University

Chiayi, 621 Taiwan

E-mail: tsay@cs.ccu.edu.tw

[†]*Department of Information Management*

Wu-Feng Institute of Technology

Chiayi, 621 Taiwan

E-mail: dks@mail.wfc.edu.tw

The mesh is an architecture that has many scientific applications, and matrix transpose is an important permutation frequently performed in various techniques involving systems of linear equations. In this paper, we present an optimal algorithm for performing matrix transpose on meshes that support wormhole switching. If N is even, our algorithm takes $\frac{N}{2+\sqrt{2}}$ communication steps to perform matrix transpose on an $N \times N$ mesh and requires only 3 more steps when the routing is restricted to XY routing, which is supported by most commercial mesh-connected parallel computers. The lower bound is $\frac{N-1}{2+\sqrt{2}}$ and the best previous bound is about $N/3.27$. The complexity of our algorithm almost matches the lower bound. Furthermore, our algorithm is simple and can be implemented on current mesh-connected parallel computers.

Keywords: matrix transpose, collective communication, mesh-connected computers, wormhole routing, parallel computing

1. INTRODUCTION

Matrix transpose is a permutation frequently performed in various techniques involving systems of linear equations. Partial differential equations are typically solved using the Alternating Direction Implicit (ADI) method by transposing the data between the solution phases in different directions [7]. Another example in which data transposition may be advantageous is solving Poisson's problem using the Fourier Analysis Cyclic Reduction (FACR) method. The mesh is one of the most important parallel architectures. A number of parallel computers with 2-dimensional or 3-dimensional mesh topol-

Received March 15, 2001; revised September 5 & October 23, 2001; accepted December 25, 2001.

Communicated by Gen-Huey Chen.

* Research partially supported by the ROC National Science Council under Contract NSC85-2213-E-194-021.

ogy have been built. Examples of 2-dimensional mesh computers include the Intel Delta, the Intel Paragon, the Symult 2010, and the Victor of IBM. The MP-1 of Maspar contains an 8-connected mesh topology. The J-machine developed at MIT has a 3-dimensional mesh topology, and the Cray T3D parallel system has a 3-dimensional torus topology.

Most newer generation hypercube and mesh parallel computers have replaced store-and-forward routing (used in the first generation machines) with wormhole, circuit-switched, or virtual cut-through routing. While there are differences between wormhole, circuit-switched and virtual cut-through routings, they all can be modeled by complexity estimates of the same form when there is no congestion (see the next section) [11]. In this paper, we assume wormhole switching on a mesh. We assume that communication is performed in a locked-step fashion; i.e., in one communication step, each node can send and receive a message.

Another orthogonal issue related to the routing policy is how the path is selected. In most existing mesh parallel computers, an oblivious routing is used, in which the path selection is fixed independent of other messages. For instance, the most popular XY routing routes a message first along the X dimension and then along the Y dimension. Ordering the dimensions in this way ensures that the XY routing will prevent deadlock (but adaptiveness is also lost). In this paper, we first develop an algorithm that achieves the optimal bound when there is no restriction on the path selection policy and then explain how it can be implemented when the selected paths are restricted to XY paths. In fact, 3 more steps are required when XY routing is assumed.

Efficient algorithms for matrix transpose have been proposed in a large number of papers. Matrix transpose (in an abstract form) can be formed recursively as described in [3]. Stone [12] gave algorithms for transposing a matrix on a shuffle-exchange network. Nassimi and Sahni [10] gave algorithms for performing the class of Bit-Permute-Complement operations, which include matrix transpose, on a mesh. Johnsson [5] and, independently, McBryan and Van de Velde [8] applied the Recursive Exchange Algorithm given in [3] to hypercubes. Later, Johnsson and Ho [6] and, independently, Stout and Wagar [13] gave transpose algorithms on hypercubes with lower communication complexity for an all-port model. All the above algorithms assume store-and-forward routing.

Matrix transpose algorithms on hypercubes with wormhole-like and e-cube routing and a 1-port model were given by Ho and Raghunath [4]. The transpose algorithm for meshes with wormhole routing was recently studied by Ding, Ho and Tsay [2]. They presented several algorithms based on the well-known Recursive Exchange Algorithm (REA) [3]. The best performance achieved by their algorithms is about $N/3.27$ steps. In this paper, we present an algorithm that almost achieves the optimal $\frac{N-1}{2+\sqrt{2}}$ bound, and requires only 3 more steps when the routing is restricted to XY routing, which is supported by most commercial mesh-connected parallel computers. Furthermore, our algorithm is simple and can be implemented on current mesh-connected parallel computers. The rest of this paper is organized as follows. Section 2 gives preliminaries, section 3 gives the algorithm, and section 4 concludes this paper.

2. PRELIMINARIES

Consider an $N \times N$ mesh of N^2 nodes each labeled with indices (i, j) , $0 \leq i, j \leq N - 1$,

where node (i, j) denotes the node in row i and column j of the mesh. A *diagonal* of the mesh consists of all the nodes (i, j) with $i - j = c$ for some integer $1 - N \leq c \leq N - 1$. The *main diagonal* is the diagonal with $i - j = 0$.

In this paper, we consider algorithms for matrix transpose on the mesh with wormhole switching. The matrix is assumed to be of size $M_1 \times M_2$, where M_1 and M_2 are multiples of N , and is partitioned into $N \times N$ blocks with block (i, j) , where $0 \leq i, j < N$, being stored at mesh node (i, j) . Thus, each node holds a submatrix of the form $\frac{M_1}{N} \times \frac{M_2}{N}$. For convenience, let $m = M_1 M_2 / N^2$ be the size of a submatrix allocated to each node. Each node (i, j) , where $i \neq j$, therefore needs to exchange a message of size m with node (j, i) , in order to form the transpose of the global matrix. For the purpose of complexity analysis, we consider only communication cost and ignore the local processing time (such as the local transpose time) as the latter is normally much smaller than the former.

When there is no congestion, wormhole, circuit-switched and virtual cut-through routings (all referred to as wormhole-like routings in this paper) can be modeled by the same form of complexity estimate as follows. To send a message of size m to a node which is d hops away when there is no congestion, the time estimate is $\tau + \delta(d - 1) + mt_c$, where τ is the initial start-up time (overhead), δ is the start-up time for each additional hop, and t_c is the data transfer time per data unit (say, a matrix element). In most real machines, τ is much larger than δ [4, 11]. Also, when m is large, which is typically the case when solving real problems in scientific computing, the third term mt_c will dominate. We thus consider the third term only in the complexity analysis of our algorithm. Since all the exchanged messages are of the same size (m), it is reasonable to assume that in one communication step, each node on the mesh can send and receive a message as long as there is no congestion. The complexity of our algorithm thus can be denoted by the number of communication steps performed by it.

Let $T_{lb}(N)$ denote the lower bound for transposing a matrix on an $N \times N$ mesh. A lower bound was proved in [2], but it is wrong. We correct it in the following proof.

Lemma 1

$$T_{lb}(N) \geq \frac{N - 1}{2 + \sqrt{2}}. \tag{1}$$

Proof: From [2], the lower bound is the minimum of $f(x) = [N^2/4 + xN - x(x + 1)]/(N + 2x)$. We will now derive x . Letting the derivative of the formula to 0, we get $N^2 - 2(2x + 1)N - 4x^2 = 0$, which has a root $x = (\sqrt{2N^2 - 2N} - N)/2$. When N is large, $N^2 \gg N$. We then approximate the root by discarding the term $2N$ in the square root and get $x = (\sqrt{2N^2} - N)/2 = (\sqrt{2} - 1)N/2$. Taking $x = (\sqrt{2} - 1)N/2$, we have $f(x) = \frac{N-1/2}{2+\sqrt{2}}$.

In fact, when $N \geq 10$, the difference between the exact value of $f(x)$ and our solution is small than 0.01. For safety, we take

$$\frac{N^2/4 + xN - x(x + 1)}{N + 2x} \geq \frac{N - 1}{2 + \sqrt{2}} \quad (\text{for } N \geq 10) \quad \square$$

3. THE ALGORITHM

In this section, we present an optimal algorithm which takes $\frac{N}{2+\sqrt{2}}$ communication steps to perform matrix transpose on an $N \times N$ mesh with wormhole switching. The basic idea of the algorithm is as follows. Let $x = \frac{N}{2+2\sqrt{2}}$, $y = (\sqrt{2}-1)x$, and let $4x + 2y = N$ (N must be even). We partition the lower half of the mesh into 15 regions as illustrated in Fig. 1. The transpose is then performed in two phases. In the first phase, all the nodes in the regions labeled from 1 to 10 complete the exchange with their corresponding nodes in the upper half. The exchange for the nodes on the unlabeled regions is performed in the second phase. The first phase is performed in x steps, and the second phase is performed in y steps. The algorithm thus takes $x + y = \sqrt{2}x = \frac{N}{2+\sqrt{2}}$ steps. We next gives details of each phase.

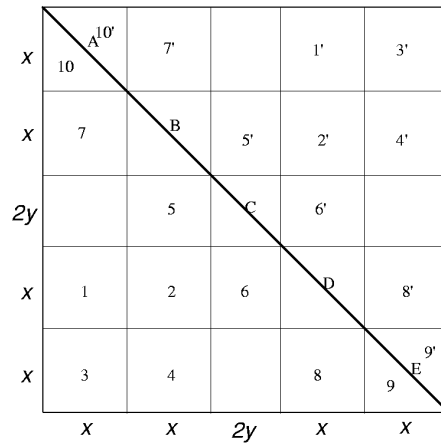


Fig. 1. Partitioning of the mesh in phase 1.

3.1 Phase 1

In phase 1, nodes of blocks labeled from 1 to 10 in Fig. 1 exchange their data with their corresponding nodes in the other half of the mesh in x communication steps. Note that every node in the main diagonal has two links connected to each half of the mesh except for the two nodes at the corner of the mesh. This implies that at every time step, there can be two messages originating from nodes in the same half and passing through the same node in the main diagonal to the other half except for the corner nodes. Our idea is to schedule the exchange so that the links of the nodes in the main diagonal are fully used. It is not clear how this can be achieved.

We partition the main diagonal into 5 portions and label them A , B , C , D , and E in order from the upper-left corner to the lower-right corner as shown in Fig. 1. Note that portion C contains $2y$ nodes in the main diagonal, and each of the other portions contains x nodes. We take the 4 horizontal lines as well as the 4 vertical lines passing through the shared boundaries of the above 5 portions and partition each half of the mesh into 10 rectangular blocks and 5 triangular blocks. Phase 1 performs the exchange for nodes in the blocks labeled 1 to 10 in Fig. 1.

Each step of phase 1 performs the following exchange is performed.

1. Choose a row in block 1. The message of each node in that row will go up to *A*, and then turn right and go to its destination in block 1'.
2. Choose a column in block 2. The message of each node in that column will go right to *D* and then turn up and go to its destination in block 2'.
3. Choose a column in block 3. The message of each node in that column will go right to *E* and then turn up and go to its destination in block 3'.
4. Choose a row in block 4. The message of each node in that row will go up to *B* and then turn right and go to its destination in block 4'.
5. Choose a column in block 5. The message of each node in that column will go right to *C* and then turn up and go to its destination in block 5'.
6. Choose a row in block 6. The message of each node in that row will go up to *C* and then turn right and go to its destination in block 6'.
7. Choose a column in block 7. The message of each node in that column will go right to *B* and then turn up and go to its destination in block 7'.
8. Choose a row in block 8. The message of each node in that row will go up to *D* and then turn right and go to its destination in block 8'.
9. Choose a row in block 9. The message of each node in that row will go up to *E* and then turn right and go to its destination in block 9'.
10. Choose a column in block 10. The message of each node in that column will go right to *A* and then turn up and go to its destination in block 10'.

Note that in each step, each pair of corresponding nodes exchanges their messages along the same path but in different directions. Fig. 2 illustrates the paths selected for exchanging the messages between each pair of blocks. It is clear that there is no edge congestion. Since each of blocks 1, 4, 6, 8, and 9 contains x rows, and each of blocks 2, 3, 5, 7, and 10 contains x columns, above operations can be iterated x times to complete phase 1.

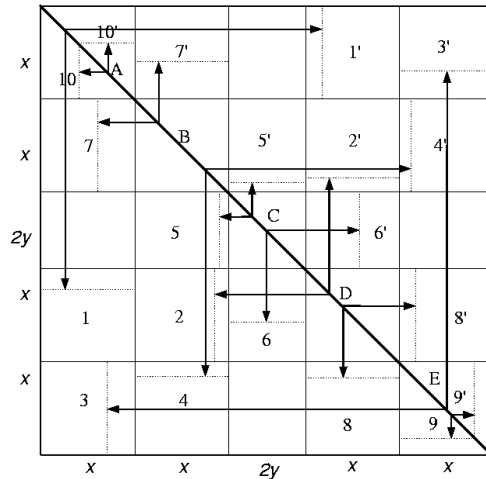


Fig. 2. Paths selected in phase 1.

Lemma 2 Phase 1 can be done in x steps.

Note that in the above process some messages, for example, the messages from block 1, take YX routes, which requires 2 communication steps in XY routing. By pipelining the communications initiated in different iterations, phase 1 can be done in $x + 1$ communication steps while restricted to XY routing. We thus have the following lemma.

Lemma 3 Phase 1 can be done in $x + 1$ steps when the routing is restricted to XY routing.

3.2 Phase 2

In phase 2, we perform exchange for nodes in the remaining blocks. The remaining blocks are partitioned into 18 smaller blocks as in Fig. 3. Phase 2 performs the following operations.

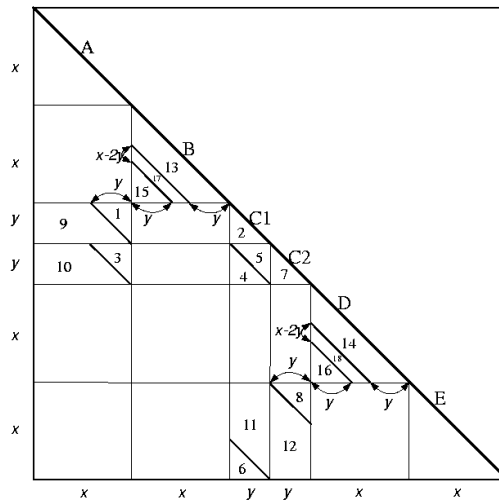


Fig. 3. Partitioning of the mesh in phase 2.

- We combine block 1 and block 2 into a virtually rectangular block. Nodes in the combined rectangular block exchange their messages with their corresponding nodes in the upper half column by column as follows. Choose a column from the combined block. The message of each node in that column will go right to C_1 , and then go up to its destination node in the upper half. Nodes in the upper half use the same paths but in different direction to send their messages in the lower half. See Fig. 4 for an illustration.
- Block 3 and block 4 are combined into a rectangular block, and exchange their messages with their corresponding nodes in a way similar to block 1 and 2. The message of each node in the chosen column will go right to C_2 , and then go up to its destination node in the upper half.

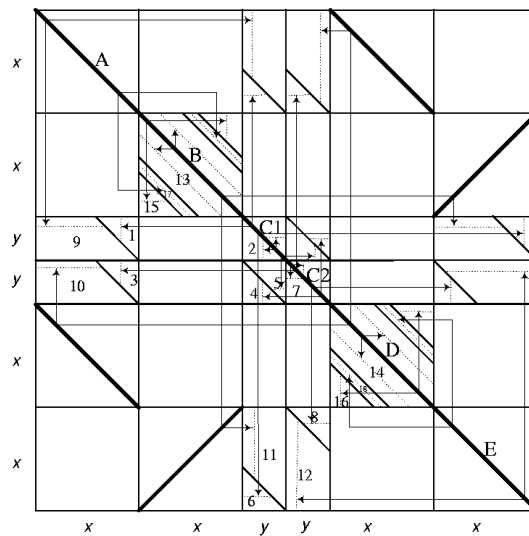


Fig. 4. Paths selected in phase 2.

- Block 5 and block 6 are combined into a rectangular block, and work similarly to block 1 and block 2. At each communication step, we choose a row from the combined block. The message of each node in that row will go up to C_1 , and then go right to its destination node in the upper half.
- Block 7 and block 8 are combined into a rectangular block, and work similarly to block 1 and block 2. At each communication step, we choose a row from the combined block. The message of each node in that row will go up to C_2 , and then go right to its destination node in the upper half.
- Nodes in block 9 performs the exchange row by row. At each communication step, we choose a row from block 9. The message of each node in that row will go up to A , and then go right to its destination node in the upper half.
- Nodes in block 10 performs the exchange row by row. At each communication step, we choose a row from block 10. As illustrated in Fig. 4, the message of each node in that row will go down to the diagonal of the block immediately below block 10, then go right to D , and then follow a symmetric path in the upper half to reach its destination node.
- Nodes in block 11 performs the exchange column by column in a way similar to nodes in block 10. At each communication step, we choose a column from block 11. As illustrated in Fig. 4, the message of each node in that column will go left to the diagonal of the block immediately left to block 11, then go up to B , and then follow a symmetric path in the upper half to reach its destination node.
- Nodes in block 12 performs the exchange column by column. At each communication step, we choose a column from block 12. The message of each node in that column will go right to E , and then go up to its destination node in the upper half.
- Note that nodes in block 13 consists of y lines parallel to the main diagonal. For simplicity, each line is referred as a diagonal. Nodes in block 13 performs the exchange diagonal by diagonal. At each communication step, we choose a diagonal from block

13. The message of each node in that diagonal will go right to B , and then go up to its destination node in the upper half.

- Nodes in block 14 performs the exchange diagonal by diagonal. At each communication step, we choose a diagonal from block 14. The message of each node in that diagonal will go up to D , and then go right to its destination node in the upper half.

We have explained how to perform the exchange for nodes in blocks 1 to 14. We next explain how to perform the exchange for nodes in blocks 15, 16, 17 and 18.

Since block 15 is equivalent to block 6, we combine block 15 and block 11 to form a virtually rectangular block in which a row of block 15 is concatenated to a column of block 11 to form a virtual column in the combined block. When a column in block 11 is chosen to send out messages, nodes in the row of block 15 concatenated to that column also start to send out their messages. The messages from each row of block 15 will go up to B , and then turn right to their destinations. See Fig. 4 for an illustration. Similarly, we combine block 16 and block 10, and whenever a row of block 10 is scheduled to perform the exchange, the column of block 16 concatenated to that row is also scheduled to perform the exchange.

Nodes in block 17 are combined with nodes in block 9. Consider a row of block 9 containing $x - k$ nodes. That row will use links in column 0 to $x - k - 1$ in the lower half, and links in rows 0 to $x - k - 1$ in the upper half to perform the exchange. This implies that links in column $x - k$ to $x - 1$ in the lower half, and links in rows $x - k$ to $x - 1$ are not used by that row. We then choose k nodes from block 17 to used those used links to perform the exchange. Namely, the message of each of the k chosen nodes will go left to a used column first, then go up to A , and then follow a symmetric path in the upper half to reach its destination. See Fig. 4 for an illustration. To guarantee that no edge congestion occurs, those k nodes should be chosen from k different columns and k different rows. To achieve this, we order the nodes in block 17 in diagonal-major order as illustrated in Fig. 5, and schedule nodes in block 17 according to that order. Namely, we choose the first k unscheduled nodes to perform the exchange. Since $k \leq y$ and each diagonal in block 17 contains at least y nodes, it is clear that in diagonal-major order, any k consecutive nodes are in k different rows and in k different columns. Since the number of nodes in block 17 is equivalent to the number of nodes in block 1, we can combine block 17 and block 9 to perform the exchange in y communication steps. Similarly, we combine 18 and block 12.

6				
3	7			
1	4	8		
	2	5	9	

Fig. 5. Diagonal-major order.

Lemma 4 Phase 2 can be performed in y steps.

Note that each path selected to exchange a pair of messages in phase 2 consists of at most 3 XY paths. By pipelining the exchange initiated in different step, phase 2 can be performed in $y + 2$ communication steps. We thus have the following lemma.

Lemma 5 Phase 2 can be performed in $y + 2$ steps when the routing is restricted to XY routing.

Since $x + y = x + (\sqrt{2} - 1)x = \sqrt{2}x = \frac{N}{2+\sqrt{2}}$, we have the following theorem.

Theorem 1 Matrix transpose on $N \times N$ mesh with wormhole routing can be performed in $\frac{N}{2+\sqrt{2}}$ steps.

Theorem 2 Matrix transpose on $N \times N$ mesh with wormhole and XY routing can be performed in $\frac{N}{2+\sqrt{2}} + 3$ steps.

According to our mesh partition, the equation, $4x + 2y = N$, implies that N must be an even number. If N must be odd, we apply our algorithm to the $(N - 1) \times (N - 1)$ sub-mesh and use extra one step to transpose other nodes' message. The total steps are $\frac{N-1}{2+\sqrt{2}} + 1 = \frac{N}{2+\sqrt{2}} + \frac{1+\sqrt{2}}{2+\sqrt{2}}$.

Theorem 3 When N must be odd, matrix transpose on $N \times N$ mesh with wormhole routing can be performed in $\frac{N}{2+\sqrt{2}} + \frac{1+\sqrt{2}}{2+\sqrt{2}}$ steps.

4. FURTHER REMARKS

In this paper, we have presented an optimal algorithm that takes $\frac{N}{2+\sqrt{2}}$ communication steps to transpose a matrix on an $N \times N$ (N is even) mesh with wormhole switching. The performance almost matches the lower bound by a constant factor. When the routing is restricted to XY routing, the algorithm takes only 3 steps more.

Throughout the paper, we have implicitly assumed that each node can somehow comply to the globally assigned schedule to avoid the message congestion [15]. There are at least two approaches in implementing the algorithm on an asynchronous mesh. One is to perform a global synchronization before the matrix transpose, then some appropriate amount of idle time is inserted to respective nodes before sending out their messages. Another approach is to add explicit synchronization messages to control the initiation of real messages. It is possible to do so such that the term for t_c in the overall communication complexity remains unchanged and the term for τ is increased only by a very small constant factor.

It is interesting to study whether or not optimal performance can be achieved other networks such as torus.

REFERENCES

1. C. Calvin and D. Trystram, "Matrix transpose for block allocations on torus and de Bruijn networks," LMC-IMAG Technical Report, 1995.
2. K. S. Ding, C. T. Ho, and J. J. Tsay, "Matrix transpose on meshes with wormhole and XY routing," *Discrete Applied Mathematics*, Vol. 83, 1998, pp. 41-59.
3. J. O. Eklundh, "A fast computer method for matrix transposing," *IEEE Transactions on Computers*, Vol. 21, 1972, pp. 801-803.
4. C. T. Ho and M. T. Raghunath, "Efficient communication primitives on hypercubes," *Journal of Concurrency: Practice and Experience*, Vol. 4, 1992, pp. 427-457.
5. S. L. Johnsson, "Communication efficient basic linear algebra computations on hypercube architectures," *Journal of Parallel Distributed Computing*, Vol. 4, 1987, pp. 133-172.
6. S. L. Johnsson and C. T. Ho, "Matrix transposition on boolean n-cube configured ensemble architectures," *SIAM Journal on Matrix Analysis Applications*, Vol. 9, 1988, pp. 419-454.
7. S. L. Johnsson and C. T. Ho, "Optimizing tridiagonal solvers for alternating direction methods on boolean cube multiprocessors," *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, 1990, pp. 563-592.
8. O. A. McBryan and E. F. Van de Velde, "Hypercube algorithms and implementations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 8, 1987, pp. s227-s287.
9. P. McKinley, Y. Tsai, and D. Robinson, "A survey of collective communication in wormhole-routed massively parallel computers," Technical Report, MSU-CPS-94-35, Dept. of Computer Science, Michigan State University, June 1994.
10. D. Nassimi and S. Sahni, "An optimal routing algorithm for mesh-connected parallel computers," *Journal of ACM*, Vol. 27, 1980, pp. 6-29.
11. L. M. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computers*, Vol. 26, 1993, pp. 62-76.
12. H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers*, Vol. 20, 1971, pp. 153-161.
13. Q. F. Stout and B. Wagar, "Intensive hypercube communication: Prearranged communication in link-bound machines," *Journal of Parallel and Distributed Computing*, 1990, pp. 167-181.
14. Y. Tsai and P. McKinley, "An extended dominating node approach to collective communication in all-port wormhole-routed 2D meshes," in *Proceedings of the Scalable High Performance Computing Conference, IEEE*, 1994, pp. 199-206.
15. Y. C. Tseng, "A dilated-diagonal-based scheme for broadcast in a wormhole-routed 2D torus," *IEEE Transactions on Computers*, Vol. 46, 1997, pp. 947-952.

Jyh-Jong Tsay (蔡志忠) received the B.E. in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1982, and the M.S. and Ph.D. in Computer Science from Purdue University, West Lafayette, in 1988 and 1990, respectively. In August 1990, he joined National Chung Cheng University,

Chiayi, Taiwan, R.O.C., where he is currently an associate professor of Computer Science and Information Engineering. His research interests include the design and analysis of algorithms, parallel computing, text categorization, and data mining.

Kuo-Shun Ding (丁國順) received his B.S. degree in Computer Science and Information Engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1992, and Ph.D. degree in Computer Science and Information Engineering from National Chung-Cheng University, Chiayi, Taiwan, R.O.C., in 2002. In August 2002, he joined Wu-Feng Institute of Technology, Chiayi, Taiwan, where he is currently an assistant professor of Information Management. His research interests are parallel computing, distributed system, computer architecture, and high-speed networking.

Wen-Tsong Wang (王文聰) Received his B.E. degree in Computer and Information Science from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1995 and M.S. degree in Computer Science and Information Engineering from National Chung Cheng University, Chiayi, Taiwan, R.O.C., in 1997.