Tight Lower Bound for Matrix Transposition on the Reconfigurable Mesh

T. H. Kaskalis, V. Dagdilelis, G. Evangelidis, K. G. Margaritis

Abstract— **The problem of matrix transposition is known to be solved in $O(n)$ time steps on a mesh connected processor array enhanced by reconfigurable buses. This paper explores the absolute minimum, within these limits, that can be achieved on a *Reconfigurable Mesh*. Assuming an $(n \times n)$ matrix, we investigate the optimum time required to calculate its transposed respective. We conclude in a tight lower bound, which refines those referred in the literature.**

Keywords— **Reconfigurable Mesh, Parallel Matrix Transpose.**

I. Introduction

ALTHOUGH recent development of VLSI technology paved the way towards parallel machines, featuring large number of processors, practice indicates that this increase in raw computational power does not always translate into increased performance of the same order of magnitude. Among the massively parallel architectures, the *mesh* has been successfully applied to a number of domains, the foremost being image processing, computer vision, pattern recognition, digital and computational geometry [1], [8], [13], [25], [26], [28]. Its regular structure and simple interconnection topology makes the mesh particularly well suited for VLSI implementation, with several models built over the years. Examples include the ILLIAC IV, the CM-2, the MPP and the MasPar [3], [10], [11], [12].

In cases, where most of the computations are local to each processing element (PE), or involve only regular pattern or nearest neighbor communication, the mesh emerges as one of the natural choices. However, if computations involve data items spread over processing elements far apart, the mesh architecture becomes less attractive, because of its large computational diameter, i.e. the maximum of the minimum distance between any two processors in the network.

To remedy this situation, several researchers have proposed to augment the mesh architecture with high-speed buses that allow fast communication between processors located in different areas of the mesh. A common feature of these bus structures is that the communication patterns among processor elements cannot be modified during the execution of the algorithm. In an attempt to alleviate this problem and further enhance the capabilities of the mesh, researchers consider adding reconfigurable features to parallel computers.

The *reconfigurable mesh* architecture [2], [27], [15], [34] consists of processing elements arranged in a grid. The PEs are connected to a grid shaped reconfigurable bus and each PE has locally controllable bus switches, allowing the broadcast bus to be divided into subbuses.

All authors are with the University of Macedonia, Thessaloniki, Greece. E-mail: {kaskalis,dagdil,gevan,kmarg}@uom.gr

This model of computation actually captures the fundamental properties of the CHiP computer [38], mesh connected computers augmented with broadcast buses [33], the bus automaton [36], the polymorphic-torus network [22] and the corterie network in the latest version of the Content Addressable Array Parallel Processor (CAAPP) [41]. Such an architecture has been realized using Field Programmable Gate Arrays (FPGAs) [40]. The reconfigurable mesh combines two desirable features of massively parallel architectures: constant diameter and a dynamical reconfigurable bus system.

These features are exploited here, in order to solve the fundamental problem of matrix transposition in optimal time complexity, on the aforementioned architecture. A great number of linear algebra problems, including congruence transformations, equations systems solving, matrix inversion e.t.c. [9], have already made the problem of matrix transposition a key matter in the process of efficient problem solving.

In that context, we will proceed by briefly introducing the reconfigurable mesh architecture, followed by a simple and immediate mapping of the matrix transposition algorithm. Subsequently, we will present our step by step approach towards the best possible time complexity, for the problem. This discussion leads us to a definite minimum, as regards the time requirements for the transposition of matrices. This paper concludes with a comparison of our approach as regards previously reported results.

II. The Reconfigurable Mesh Model

The $n \times n$ reconfigurable mesh consists of an $n \times n$ array of processing elements connected to a grid-shaped reconfigurable broadcast bus. A 3×3 reconfigurable mesh is shown in Fig. 1.a. Each PE has locally controllable bus switches. Internal connections among the four ports (North, South, East, West) of a PE can be configured during the execution of algorithms.

There are 15 possible connection patterns, appearing in Fig. 1.b. The configuration is specified by grouping together ports, which have been shorted. Thus, if a PE's ports are all electrically isolated, its configuration is denoted {N,S,E,W}. Similarly, if the North and South ports are shorted and so do the East and West ports, then the configuration is denoted {NS,EW}.

A single time step on a reconfigurable mesh is composed of the following four phases:

- **Phase 1**: Change the configuration of a reconfigurable bus system by connecting or disconnecting its own ports

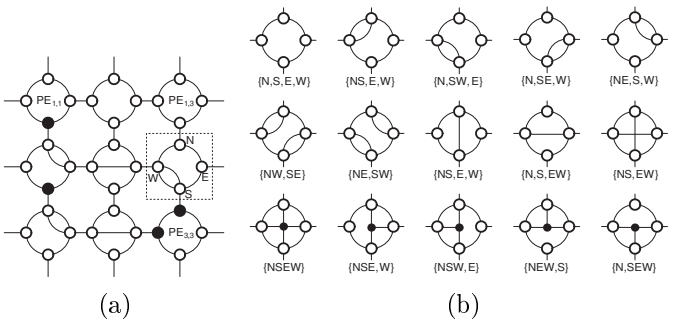

Fig. 1. A 3 × 3 Reconfigurable Mesh and the Possible Connection Configurations

with buses internally.

• **Phase 2**: Send data to each port. When more than one processor attempts to send data along the same bus, a collision occurs and the data being transmitted will be discarded by all processors connected to the bus.

• **Phase 3**: Receive data from each port. Some or all processors connected to the bus read the data sent in phase 2.

• **Phase 4**: A constant-time local computation is done by each processor.

We, therefore, understand that if PE (1,1) of Fig. 1.a has to send a data item to PE (3,3), which, respectively, has to send another data item to PE (2,1), the phases are as follows. First, a bus connection scheme, similar to that of Fig. 1.a, has to be established (Phase 1). Then, the PEs (1,1) and (3,3) write their data items to their South and West ports, respectively (Phase 2). Processing Elements (3,3) and (2,1) read data from their North and South ports, respectively (Phase 3) and a possible local computation takes place, which may involve the newly arrived data items (Phase 4). The ports, actually writing or reading data, during this example, are denoted black in Fig. 1.a.

Other than the buses and switches, the reconfigurable mesh is similar to the standard 2-dimensional mesh in that it has $\Theta(n^2)$ area, under the assumption that PEs, switches and a link between adjacent PEs occupy unit area in the word model of VLSI. Moreover, we assume that the size of the local storage in each PE is O(1) words, where each word is $O(\log_2 n^2)$, which is also the bit value broadcast. Finally, we assume that the connections in the edges of the mesh wrap around to their respective row and column elements, forming an overall torus scheme. We should note that the reconfigurable mesh algorithms, presented in this paper can be simulated in the MRN [29] and LRN [4] models without slowdown. These models accept the first 10 possible connection schemes of Fig. 1.b (one-to-one port connection) and in this paper we only employ connection patterns of this kind.

Many different algorithms have been developed for the reconfigurable mesh parallel architecture. For example, a number of researchers, including Reisis [35], have proposed constant time reconfigurable mesh algorithms to compute the geometry problem of convex hull of a set of points in the plane. Olariu et al. [31], [32] have proposed a number of constant-time algorithms for geometry problems involving convex polygons and a number of low-level computer vision problems. ElGindy and Wetherall propose a Voronoi diagram algorithm. Chao et al. [6], in their work, have solved the multiple search problem on the reconfigurable mesh. Jang et al. [14], [16] have proposed constant time solutions to the problems of computing all nearest neighbors of a set of points in the plane, computing the maximal elements in 3-d, and 2-dominance counting. Jenq and Sahni [17] have computed the Hough-transform on the reconfigurable mesh and Kim and Park [21] presented efficient list ranking algorithms, on the respective model.

Its low wiring cost and regular structure make the reconfigurable mesh suitable for VLSI implementation. In addition, it is not hard to see that the reconfigurable mesh can be used as a universal chip capable of simulating any equivalent area architecture without loss of time. It is worth mentioning that at least three VLSI implementations have demonstrated the feasibility and benefits of the reconfigurable mesh: These are the YUPPIE chip [23], the GCN chip [39] and the PPA chip [24]. These implementations suggested that the broadcast delay, although not constant, is very small. For example, only 16 machine cycles are required to broadcast on a 10^6-processors YUPPIE. The GCN has further shortened the broadcast delay by adopting precharged circuits. Moreover, it has been shown that the broadcast delay can be reduced even further if the reconfigurable bus system is implemented using optical fibers as the underlying global bus system and using electrically controlled directional coupler switches for connecting or disconnecting two fibers [37].

III. Parallel Matrix Transposition on Meshes

An $n \times n$ matrix A is given, for example ($n = 3$):

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The transpose of matrix A, denoted A^T, is produced by changing the rows of A with its corresponding columns:

$$A^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}$$

The elements of A can be data objects of any type. Thus, a_{ij} can be an integer, a real, a character and so on.

The following code computes the transpose of an $n \times n$ matrix, sequentially:

Procedure Transpose ($A[n \times n]$)
for $i = 1$ to $n - 1$ do
    for $j = i + 1$ to n do
        swap(a_{ij},a_{ji})
    end for
end for

The algorithm transposes A in place, which means that it returns A^T in the same memory locations previously occupied by A. Assuming that one time step is needed for

every element exchange, then the above sequential algorithm takes $(n^2-n)/2$ time steps to be completed.

Many researchers have considered solving the problem using different parallel architectures. O'Leary [30] implemented an algorithm for transposing an $n \times n$ matrix in $3n-1$ time steps using a rather simple systolic array of $n \times n$ switching processors and $n \times n$ bit buffers. The matrix enters the systolic array by rows and exits by diagonals (the reverse process also exists), rather than being preloaded and transposed in place. The processor elements have no memory capabilities and act as simple switches, which control the flow of data at each time step, where the buffers act as delays for the synchronization of the transposition process. The idea of using the switching processor elements give the ability to reconfigure the communication network of this systolic processor array during program execution by shifting the data from rows to columns and vice versa.

In [18] Johnsson described a matrix transpose algorithm on an $n \times n$ grid in $2n-1$ steps of direct neighbor communications, by shifting successively superdiagonals in the direction of decreasing column indices and then in the direction of increasing row indices. Symmetrically, subdiagonals are shifted first in the direction of decreasing row indices and then in the direction of increasing column indices.

Calvin and Trystram [5], have presented algorithms for the matrix transpose problem on distributed memory parallel machines. With the use of Johnsson's algorithm, Calvin and Trystram have shown that the transposition problem of an $n \times n$ matrix on a square mesh of processors takes $n-1$ steps for the simple case and $(n-1)/2$ for a torus network.

The most recent results on 2-D mesh matrix transposition were presented by Ding et al. [7] and Kaufmann et al. [20]. In the first paper, the authors present a lower bound for the time steps required to transpose an $n \times n$ matrix on a mesh of the same size:

$$\frac{n^2/4 + xn - x(x+1)}{n+2x} \quad \text{where} \quad x = \frac{\sqrt{2n^2-2n}-n}{2} \qquad (1)$$

We should note that this number is actually doubled, since one time step is considered to contain a whole swap operation. If we consider the four phases, presented in the previous section, we understand that a swap operation actually demands two time steps. Ding et al. utilize the Recursive Exchange Algorithm (REA) in their approach [7] and introduce a number of algorithms, the best of which is away from optimal by about 5%. Kaufmann et al. [20] refine these results, proposing a more balanced scheme, which achieves the lower bound of time steps, presented in (1). In the conclusions section of this paper we will return in the discussion of these papers, comparing them with our approach.

Finally, Kao et al. [19] applied several algorithms on the CRAP architecture (cross-bridge reconfigurable array of processors), including the transpose and untranspose operations. In their paper they show that on a 3-D reconfigurable mesh of size $n \times n \times m$, a matrix of size $n \times m$ can be transposed in O(1) steps.

IV. Tight Lower Bound of Time Steps

Consider an $n \times n$ reconfigurable mesh with wraparound connections (torus), containing the elements on an $n \times n$ matrix A. At first, each Processing Element (PE) (i,j) contains the respective matrix element a_{ij}. At the end of the operation the (i,j) PE should contain a_{ji}, implementing the transpose operation.

Concentrating on the part of the mesh above the main diagonal, we find $n-1$ super-diagonals $U_1, U_2, \ldots, U_{n-1}$ containing $n-1, n-2, \ldots, 1$ elements, respectively. We define the *combined diagonals* as:

$$C_1^U = \{U_1, U_{n-1}\} \ , \ C_2^U = \{U_2, U_{n-2}\} \ , \ \ldots \ , \\ C_{\lfloor n/2 \rfloor}^U = \{U_{\lfloor n/2 \rfloor}, U_{\lceil n/2 \rceil}\} \qquad (2)$$

each of which contains n elements, except the last ($C_{\lfloor n/2 \rfloor}^U$), which contains n elements for an odd n or $n/2$ elements for an even n. In a similar manner we can define $C_1^L, C_2^L, \ldots, C_{\lfloor n/2 \rfloor}^L$, for the part of the mesh below the main diagonal. We understand that all the elements of a combined diagonal have the same distance from the main diagonal, taking under consideration the torus scheme of the architecture.

In one time step, all the elements of a combined diagonal can swap places utilizing all the available ports of the elements of the main diagonal. Following a somewhat modified scheme of the Johnsson's algorithm [18] we can see in Fig. 2 the transposition of the elements of C_2^U and C_2^L on an 8×8 mesh. Following this simple approach, we can transpose the matrix in a number of time steps equal to the number of combined diagonals, i.e. $\lfloor n/2 \rfloor$.

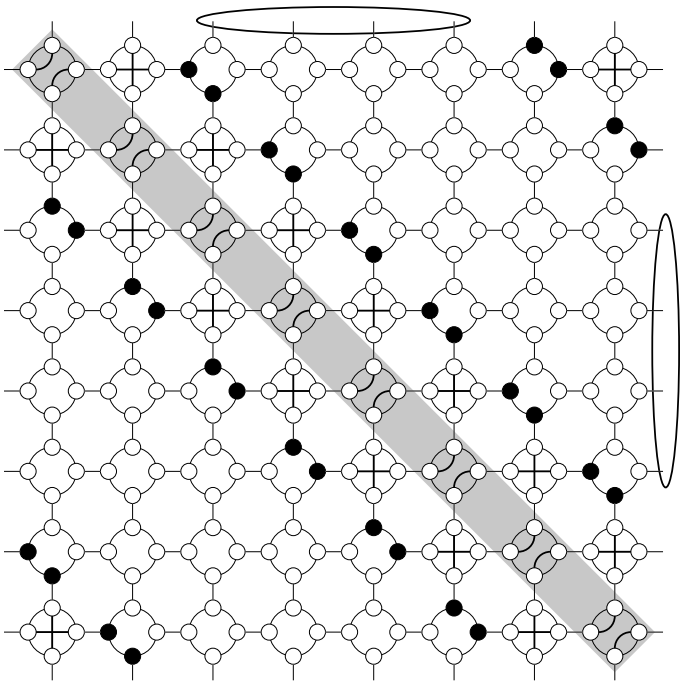

Fig. 2. An 8×8 Reconfigurable Mesh Performing a Simple Transposition Step

On the other hand, we can see that there is a number of wrap around port links, which could be utilized to transpose more matrix elements in each time step. Denoting the pair C_i^U, C_i^L with C_i, we can see that during the transposition of C_i, there are $n-2i$ links in each side of the mesh, which can be further used. We highlight these links in the upper part of Fig. 2 with two circles, implying the respective ones in the lower part.

Focusing again in the upper part, we can accept that for reasons of symmetry the links of one side (e.g. north side) of the mesh will be *output* links, with the *input* links being those of the other side (e.g. east side). By that we mean that data items move out from this upper part of the mesh through one side, while the respective items, coming from the lower part of the mesh, enter through the other side.

It is clear by now that the elements of some combined diagonals will travel through the main diagonal, while the remaining elements will use the rest of the wrap around links. The question, which naturally arises, is *which combined diagonals should we select to send through the main diagonal, in order to minimize the effective time steps?* An intuitive answer leads us to select those diagonals closer to the main, in order to allocate as few PE ports as possible. This will also be proven in the following.

Assume t the number of time steps needed to perform the matrix transpose operation. In each of the t steps, the transposition of the combined diagonal C_i takes place and we can transpose $2(n-2i)$ more matrix elements, through the "long" paths, bypassing the main diagonal. Let us denote $i_1, i_2, \ldots, i_t$ the subscripts of the combined diagonals selected to pass through the main diagonal. Then:

$$nt + (n-2i_1) + \cdots + (n-2i_t) = \frac{n^2-n}{2}$$
$$\Rightarrow \quad t = \frac{n-1}{4} + \frac{i_1+i_2+\cdots+i_t}{2n} \tag{3}$$

In order to minimize t, we understand that $i_1 = 1$, $i_2 = 2$, ... , $i_t = t$. This justifies our first intuitive answer. Therefore:

$$(3) \quad \Rightarrow \quad t = \frac{n-1}{4} + \frac{t(t+1)}{2n}$$
$$\Rightarrow \quad t^2 - (2n-1)t + \frac{n^2-n}{2} = 0 \tag{4}$$

which, for a minimum t, leads us to:

$$t = \frac{2n-1-\sqrt{2n^2-2n+1}}{2} \tag{5}$$

Equation (5) represents the absolute minimum of the time steps required to perform the matrix transposition on the reconfigurable mesh with wrap around links. Moreover, since the number of steps is an integer, the correct way to express (5) is:

$$t = \left\lceil \frac{2n-1-\sqrt{2n^2-2n+1}}{2} \right\rceil \tag{6}$$

The fact that (5) is the tight lower bound of the operation can also be proved through the following argument. In each step of the operation we would require that all the available PE ports are utilized in the best possible manner. This means that the communicating PEs use the minimum number of hops to exchange their data items and all the available ports are being used. Through the above mentioned scheme, every pair of PEs, which communicate through the main diagonal, utilize $4i$ port-pairs, where i is the subscript of the combined diagonal they belong to (see also Fig. 2). Moreover, every pair of PEs which communicate, bypassing the main diagonal, utilize the constant number of $2n$ port-pairs. Summing up these port-pairs, being utilized in step i of the process, we obtain: $4in + (n-2i)2n = 2n^2$, which is actually the total number of port-pairs available in an $n \times n$ reconfigurable mesh.

This last argument also justifies our approach of selecting this method of combined diagonals. We wish to utilize the most possible (the target is all) port links available in the network, in every time step. Moreover, we wish to follow such allocation schemes that the PE communication follows the minimum possible number of intermediate hops. The communication through the main diagonal relates the number of hops with the distance of the PEs from it. On the other hand, the communication through the rest of the wrap around links (bypassing the main diagonal) assumes a constant minimum of hops, which is not related to the position of the PEs in the mesh. We, therefore, understand that the choice of the "combined diagonals" scheme (same distance from main diagonal), in accordance with those diagonals closer to the main, comes naturally.

TABLE I
MATRIX DIMENSIONS RELATED TO TIME STEPS

Matrix Dimension	Time Steps
3 , 4	1
5 , 6 , 7	2
8 , 9 , 10	3
11 , 12 , 13 , 14	4
15 , 16 , 17	5
18 , 19 , 20 , 21	6
22 , 23 , 24	7
25 , 26 , 27	8
⋮	⋮

(a)

Matrix Dimension	Time Steps
4	1
21	6
120	35
697	204
4060	1189
23661	6930
137904	40391
803761	235416
⋮	⋮

(b)

Moving forward, we can construct a table which can illustrate the minimum number of steps required to perform the matrix transposition operation, for increasing dimension values n. Table I.a can be read as follows: for a given matrix dimension n, the transposition operation cannot be performed in less than t time steps. It is interesting to see if there are certain values of n, for which (5) and (6) produce the same result. In a situation like that, we expect the tight lower bound of (5) to be achieved exactly and every

port of the mesh to be utilized in every effective time step. Table I.b lists the values of n for which the absolute minimum of time steps can actually be achieved in the most efficient way.

Concluding this section, we should consider the case when the dimensions of matrix A do not match those of the given reconfigurable mesh architecture. For an $m_1 \times m_2$ matrix A we allocate a submatrix of the form $m_1/n \times m_2/n$ to each Processing Element. Denoting $m = m_1 m_2/n^2$ the size of each submatrix, we understand that every transposition step now assumes the exchange of m data values. Considering only the communication cost and ignoring the local computation cost [7], [20], the overall time steps are given by multiplying the results of (5) and (6) with the factor m.

V. Conclusions and Future Work

We presented the tight lower bound of time steps, when performing matrix transposition on the reconfigurable mesh with wrap around connections (torus). The most recent research on the matter presented lower bound results on mesh connected computers (not torus-like) [7] and proposed efficient ways in accomplishing these results [20]. If we translate this effort in our model (two time steps for the transposition of an element pair) the reported lower bound can be written as:

$$t = 2 \cdot \frac{n^2/4 + xn - x(x+1)}{n + 2x} \text{ where } x = \frac{\sqrt{2n^2 - 2n} - n}{2} \tag{7}$$

Since we employ a torus scheme, one would expect to produce a model operating in half the time steps of (7). This is because the torus generally halves the diameter of the mesh and produces algorithms expected to perform two times faster on it than on a simple mesh [5].

Our approach, however, is quite different than the one presented in [7]. We introduce the lower bound of:

$$t = \frac{2n - 1 - \sqrt{2n^2 - 2n + 1}}{2} \tag{8}$$

which is very close, but *not* exactly the half of (7), rather than slightly less. Our result is actually the achievable refinement of the previously reported lower bound.

Equation (7) does not produce an integer number of time steps, for any n. As a result, we cannot actually "reach" the lower bound, but rather "stay closely above it". On the contrary, as Table I.b presents, (8) produces numbers that can lead to the realization of the actual lower bound. For the same value of n, Equation 7 leads to $t = 2 \cdot 7$ time steps, exhibiting the difference of the two lower bounds.

The natural question, arising from the discussion above, is whether the lower bound presented can actually be achieved on the reconfigurable mesh. The problem becomes harder, if we focus on the matrix dimensions presented in Table I.b. For these values of n there is no communication port left unutilized in any time step and the problems of data congestion are likely to appear. There is strong evidence, however, that a systematic allocation scheme can actually be devised. This task is included in our immediate future research plans, in order to introduce a full, systematic, algorithmic approach for the problem at hand.

References

[1] S.G. Akl, *Parallel Computation: Models and Methods*, Upper Saddle River, Prentice Hall, 1993.

[2] H.M. Alnuweiri, M. Alimuddin, H. Aljunaidi, "Switch Model and Reconfigurable Netwroks: Tutorial and Partial Survey", *Proc. Workshop Reconfigurable Architectures, Eighth Int. Parallel Processing Symposium*, 1994.

[3] K.E. Batcher, "MPP: A high speed image processor", *Algorithmically Specialized Parallel Computers*, Academic Press, 1985.

[4] Y. Ben-Asher, D. Gorden, A. Schuster, "Optimal Simulations in Reconfigurable Arrays", Technical Report No. 716, Dept. of Computer Science, Technion-Israel Inst. Technology, 1992.

[5] C. Calvin, D. Trystram, "Matrix Transpose for Block Allocations on Torus and de Bruijn Networks", *Journal of Parallel and Distributed Computing*, vol. 34, no. 1, pp. 36-49, 1996.

[6] C.C. Chao, W.T. Chen, G.H. Chen, "Multiple Search Problen on Reconfigurable Meshes", *Information Processing Letters*, vol. 58, no. 2, pp. 65-69, 1996.

[7] K.S. Ding, C.T. Ho, J.J. Tsay, "Matrix Transpose on Meshes with Wormhole and XY Routing", *Discrete Applied Mathematics*, vol. 83, pp. 41-59, 1998.

[8] M.J.B. Duff, "CLIP 4: A Large Scale Integrated Circuit Array Parallel Processor", *Proc. Third Int. Joint Conf. Patern Recognition*, pp. 728-733, 1976.

[9] G.H. Golub, C.F. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, 1996.

[10] W.D. Hillis, *The Connection Machine*, MIT Press, 1985.

[11] W.D. Hillis, "The Connection Machine", *Scientific American*, vol. 256, pp. 108-115, 1987.

[12] R.W. Hockney, C.R. Jesshope, *Parallel Computers*, Adam Hilger Ltd., 1981.

[13] J.A. Holey, O.H. Ibarra, "Iterative Algorithms for Planar Convex Hull on Mesh Connected Arrays", *Proc. Int. Conf. Parallel Processing*, pp. 102-109, 1990.

[14] J. Jang, V.K. Prasanna, "Efficient Parallel Algorithms for Some Geometric Problems on the Reconfigurable Mesh", *Proc. Int. Conf. Parallel Processing*, vol. III, pp. 127-130, 1992.

[15] J. Jang, H. Park, V.K. Prasanna, "A Bit Model of Reconfigurable Mesh", *Proc. Workshop Reconfigurable Architectures, Eighth Int. Parallel Processing Symposium*, 1994.

[16] J. Jang, M. Nigam, V.K. Prasanna, S. Sahni, "Constant Time Algorithms for Computational Geometry on the Reconfigurable Mesh", *IEEE Trans. Parallel and Distributed Systems*, vol. 8, no. 1, pp. 1-12, 1997.

[17] J.F. Jenq, S. Sahni, "Reconfigurable Mesh Algorithms for the Hough Transform", *Int. Conf. Parallel Processing*, vol.3: Algorithms and Applications, pp. 34-41, 1991.

[18] S.L. Johnsson, "Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures", *J. of Parallel and Distributed Computing*, vol. 4, pp. 133172, 1987.

[19] T.W. Kao, S.J. Horng, Y.L. Wang, H.R.Tsai, "Designing Efficient Parallel Algorithms on CRAP", *IEEE Trans Parallel and Distributed Systems*, vol. 6, no. 5, pp. 554-560, 1995.

[20] M. Kaufmann, U. Meyer, J.F. Sibeyn, "Matrix Transpose on Meshes: Theory and Practice", *Computers and Artificial Intelligence*, vol. 16, pp. 107-140, 1997.

[21] S.R. Kim, K. Park, "Efficient List Ranking Algorithms on Reconfigurable Mesh", D.Z. Du, P. Eades, V. Estivill-Castro, X. Lin, Arun Sharma (Eds.): Computing and Combinatorics, 6th Annual International Conference, COCOON 2000, *Proc. Lecture Notes in Computer Science*, vol. 1858, pp. 262-271, Springer, 2000.

[22] H. Li, M. Maresca, "Polymorhic-Torus Network", *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1345-1351, 1989.

[23] M. Maresca, H. Li, "Connection Autonomy and SIMD Computers: a VLSI Implementation", *J. of Parallel and Distributed Computing*, vol. 7, pp. 302-320, 1989.

[24] M. Maresca, H. Li, P. Baglietto, "Hardware Support for Fast Reconfigurabilityin Processor Arrays", *Proc. Int. Conf. on Parallel Processing*, pp. 282-289, 1993.

[25] R. Miller, Q.F. Stout, "Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer", *IEEE Trans. Patern Analysis, Macine Intelligence*, vol. 7, pp. 216-228, 1985.

[26] R. Miller, Q.F. Stout, "Mesh Computer Algorithms for Computational Geometry", *IEEE Trans. Computers*, pp. 321-340, 1989.

[27] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, Q.F. Stout, "Parallel Computations on Reconfigurable Meshes", *IEEE Trans. Computers*, vol. 42, no. 6, pp. 678-692, 1993.

[28] R. Miller, Q.F. Stout, *Parallel Algorithms for Regular Architectures : Meshes and Pyramids*, MIT Press, 1996.

[29] M. Nigam, S. Sahni, "Sorting n numbers on $n \times n$ Reconfigurable Meshes with Buses", *Proc. Int. Parallel Processing Symposium*, pp. 73-78, 1993.

[30] D.P. O'Leary, "Systolic Arrays for Matrix Transpose and Other Reorderings", *IEEE Transactions on Computers*, vol. 36, no. 1. pp. 117-122, 1987.

[31] S. Olariu, J.L. Schwing, J. Zhang, "Fast Computer Vision Algorithms for Reconfigurable Meshes", Image and Vision Computing, pp. 610-616, 1992.

[32] S. Olariu, J.L. Schwing, J. Zhang, "Time-Optimal Convex Hull Algorithms on Enhanced Meshes", BIT, vol. 33, pp. 396-410, 1993.

[33] V.K. Prasanna-Kumar, C.S. Raghavendra, "Array Processor with Multiple Broadcasting", *J. of Parallel and Distributed Computing*, vol. 4, pp. 173-190, 1987.

[34] D.I. Reisis, V.K. Prasanna-Kumar, "VLSI Arrays with Reconfigurable Buses", *Proc. Int. Conf. Supercomputing*, 1987.

[35] D.I. Reisis, "An Efficient Convex Hull Computation on the Reconfigurable Mesh", *Proc. Int. Parallel Processing Symposium*, pp. 142-145, 1992.

[36] J. Rothstein, "Bus Automata, Brains and Mental Models", *IEEE Trans. Systems, Man and Cybernetics*, vol. 18, no. 4, pp. 522-531, 1988.

[37] A. Schuster, Y. Ben-Asher, "Algorithms and Optic Implementation for Reconfigurable Networks", *Proc. Fifth Jerusalem Conf. on Information Technology*, 1990.

[38] L. Snyder, "Introduction to the Configurable Highly Parallel Computer", *Computer*, vol. 15, no. 1, pp. 47-56, 1982.

[39] D.B. Shu, L.W. Chow, J.G. Nash, "A Content Addressable, Bit Serial Associate Processor", *Proc. IEEE Workshop on VLSI Signal Processing*, 1988.

[40] M. Thornburg, S. Casselman, "Transformable Computers", *Proc. Int. Parallel Processing Symposium*, 1994.

[41] C.C. Weems, S.P. Levitan, A.R. Hanson, E.M. Riseman, J.G. Nash, D.B. Sheu, "The Image Understanding Architecture", *Int. J. of Computer Vision*, vol. 2, pp. 251-282, 1989.