ADAPTIVE AT^2 OPTIMAL ALGORITHMS ON RECONFIGURABLE MESHES

M. MANZUR MURSHED
Comp. Sci. Lab, The Australian National University
Canberra ACT 0200, Australia
murshed@cslab.anu.edu.au

RICHARD P. BRENT
Oxford University Computing Laboratory
Oxford, OX1 3QD, U.K.
Richard.Brent@comlab.ox.ac.uk

ABSTRACT

Recently self-simulation algorithms have been developed to execute algorithms on a reconfigurable mesh (RM) of size smaller than recommended in those algorithms. Optimal slowdown, in self-simulation, has been achieved with the compromise that the resultant algorithms fail to remain AT^2 optimal. In this paper we introduce, for the first time, the idea of adaptive *algorithm which runs on RM of variable sizes without compromising the AT^2 optimality. We support our idea by developing adaptive algorithms for sorting items and computing the contour of maximal elements of a set of planar points on RM.*

1 INTRODUCTION

It is well-known that interprocessor communications and simultaneous memory accesses often act as bottlenecks in present-day parallel machines. Bus systems have been introduced to a number of parallel computers [10, 11, 22] to address this problem. A bus system is called *reconfigurable* if it can be dynamically changed according to either global or local information. Introduction of reconfigurable bus systems reduces the virtual communication diameter of any network of processors to a constant. This fact has greatly influenced researchers around the world and a large collection of constant time algorithms have already been developed [16]. To realise these constant time algorithms we need to use more processors than we usually use to solve the same problems on ordinary meshes. In fact, we can easily observe that the ratio of the number of processors used in a constant time algorithm to the number of processors used in an ordinary mesh algorithm solving the same problem is polynomial in problem size. Ben-Asher *et al.* [1] present the idea of self-simulation where the existing RM algorithms are executed with slowdown on an RM of size smaller than intended for those algorithms. A few self-simulation techniques appear in [1, 15] with optimal slowdown for various models of RM.

In this paper, we have pointed out that self-simulation even with optimal slowdown compromises the AT^2 [20, chapter 2] optimality of the resultant algorithm. To overcome this limitation of self-simulation, we have presented a new idea of developing algorithms on RM which will be adaptive in the sense that these algorithms can be executed on RM of various size keeping the AT^2 measures unaffected by the size. To illustrate our idea we have developed adaptive AT^2 optimal algorithms for sorting items and computing the contour of maximal elements of a set of planar points.

The paper is organized as follows. In the next section we present the key issues associated with RM and of its self-simulation. The idea of adaptive optimal algorithm is developed in Section 3. In Section 4 we develop an adaptive AT^2 optimal sorting algorithm. The problem of computing the contour of maximal elements of a set of planar points is defined in Section 5 and an adaptive AT^2 optimal algorithm to solve the problem is developed in the same section.

2 PRELIMINARIES

For the sake of completeness, here we briefly define the reconfigurable mesh and self-simulation of RM and then describe the optimality issues associated with self-simulation.

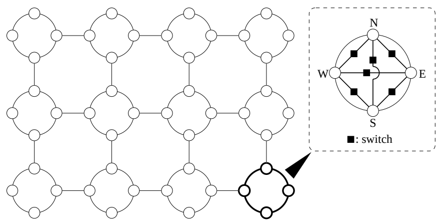

Figure 1: A reconfigurable mesh of size 3×4.

2.1 RECONFIGURABLE MESH

The reconfigurable mesh [22] is primarily a two-dimensional mesh of processors connected by reconfigurable buses. In this parallel architecture, a processor element is placed at the grid points as in the usual mesh connected computers. Processors of the RM of size $X \times Y$ are denoted by $PE_{i,j}$, $0 \leq i < X$, $0 \leq j < Y$ where processor $PE_{0,0}$ resides in the south-western corner. Each processor is connected to at most four neighboring processors through fixed bus segments connected to four I/O ports **E** & **W** along dimension x and **N** & **S** along dimension y. These fixed bus segments are building blocks of larger bus components which are formed through switching, decided entirely on local data, of the internal connectors (see Figure 1) between the I/O ports of each processor. The fifteen possible interconnections of I/O ports through switching are shown in Figure 2. Like all bus systems, the behaviour of RM relies on the assumption that the transmission time of a message along a bus is independent of the length of the bus.

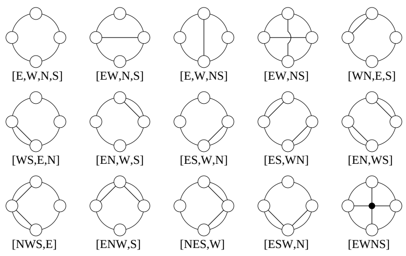

Figure 2: Possible internal connections between the four I/O ports of a processor.

A reconfigurable mesh operates in the single instruction multiple data (SIMD) mode. Besides the reconfigurable switches, each processor has a computing unit with a fixed number of local registers. Other than the buses and switches the RM of size $p \times q$ is similar to the standard mesh of size $p \times q$ and hence it has $\Theta(pq)$ area in VLSI embedding [20], under the assumption that processors, switches, and links between adjacent switches occupy unit area.

2.2 SELF-SIMULATION OF RM

Introduction of reconfigurable buses reduces the virtual communicational diameter of regular parallel architecture to a constant and thus leads to the simplest architecture, the mesh. Can reconfigurable mesh be the basis for the design of the next generation of massively parallel computers? Perhaps the answer depends on the most fundamental issue of self-simulation, i.e., simulation of large RMs by smaller ones.

We say that reconfigurable mesh R_1 is simulated by R_2 with slowdown S if the result for any algorithm A_1 on R_1 is achieved through the execution of a step-by-step simulation algorithm A_2 on R_2 in which each step of A_1 is simulated with slowdown at most S. Obviously the self-simulation of an $M \times N$ RM by a $P \times Q$ RM is said to be optimal if the slowdown is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$, $P \leq M$ and $Q \leq N$.

Ben-Asher *et al.* [1] first present the concept of self-simulation for RM and develop some self-simulation algorithms with optimal slowdown. In [15] we present optimal self-simulation algorithms of some restricted reconfigurable meshes. Optimal slowdown in self-simulation is the bottom line we can achieve but it tells a little of the optimality of the resultant algorithm.

2.2.1 AT^2 OPTIMALITY ISSUE

Is the resultant algorithm in self-simulation of RM with optimal slowdown AT^2 optimal?

We have a negative answer. Consider any problem of size n whose $AT^2 = \Omega(n^2)$, say, sorting of n elements of size $\log n$ bits each. Now, we have some sorting algorithms [4, 18, 17] which can sort n elements on RM of size $n \times n$ in constant time. Obviously the AT^2 measures of these algorithms are $\Theta(n^2)$ and thus these algorithms are AT^2 optimal.

Suppose one of this AT^2 optimal sorting algorithm is self-simulated, with optimal slowdown $\Theta(\frac{n^2}{m^2})$, in an RM of size $m \times m$ where $m < n$. The AT^2 measure of the resultant sorting algorithm then becomes $\Theta(\frac{n^4}{m^2})$ which is not optimal for $m \ll n$.

On the other hand, we have many AT^2 optimal sorting algorithms [8, 9, 19] to sort n elements on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$ in $O(\sqrt{n})$ time.

This anomaly suggests the development of adaptive algorithms which will remain AT^2 optimal while running on RM of various sizes.

3 ADAPTIVE AT^2 OPTIMAL ALGORITHMS

Let a problem $\mathcal{P}$ of size n have $I(n)$ *information content* [20, pages 51-54]. If this problem $\mathcal{P}$ is realized in a VLSI circuit with aspect ratio α then, by Ullman [20, page 57], AT^2 lower bound of $\mathcal{P}$ will be $\Omega(\alpha I^2(n))$. Now, consider an RM of size $p \times q$ where $pq = kI(n)$, $1 < p \leq q \leq I^2(n)$, and $k \geq 1$.

Let $\mathcal{P}$ be solved, AT^2 optimally, on an RM of size $p \times q$ in $O(T)$ time. Then

$$pqT^2 = \frac{q}{p}I^2(n) \ .$$

Which implies

$$T = \frac{I(n)}{p} = \frac{q}{k} \ . \qquad (1)$$

Observe that T is independent of q, the length of the larger side of the VLSI circuit. Now,

$$T = 1 \Leftrightarrow p = I(n) \ .$$

Thus, development of constant time algorithm is feasible whenever $p = I(n)$ for any $q \geq p$. As we are interested in keeping the area at minimum, the minimum possible value of q should be considered. So,

$$T = 1 \Leftrightarrow p = q = k = I(n) \ . \tag{2}$$

Lemma 1 *To solve $\mathcal{P}$ AT^2 optimally in constant time, an RM of size at least $I(n) \times I(n)$ is required.* □

Again,

$$k = 1 \Leftrightarrow T = q \ .$$

This implies that whenever the area of the VLSI circuit equals the information content of the problem to be solved, the time of solution depends only on q, the length of the larger side of the VLSI circuit. As we are interested in keeping the time at minimum, the minimum possible value of q should be considered. So, $pq = I(n)$ and $p \leq q$ derive the following:

$$k = 1 \Leftrightarrow p = q = T = \sqrt{I(n)} \ . \tag{3}$$

Lemma 2 *To solve $\mathcal{P}$ AT^2 optimally on an RM of size $\sqrt{I(n)} \times \sqrt{I(n)}$ requires $\Omega\left(\sqrt{I(n)}\right)$ time.* □

Lemma 2 has extra significance. Communication diameter of an ordinary mesh of size $\sqrt{I(n)} \times \sqrt{I(n)}$ is $\Theta\left(\sqrt{I(n)}\right)$. Thus, the power of reconfigurability becomes absolutely useless when $\mathcal{P}$ is tried to be solved AT^2 optimally on an RM of size $\sqrt{I(n)} \times \sqrt{I(n)}$.

We are interested in developing an *adaptive* algorithm to solve $\mathcal{P}$ on an RM of size $p \times q$, $\sqrt{I(n)} \leq p \leq q \leq I(n)$ such that the time required to solve $\mathcal{P}$ is $O\left(\frac{q}{k}\right)$, which is AT^2 optimal, where $k = \frac{pq}{I(n)}$. Obviously minimum AT^2 lower bound can only be achieved when $p = q$. The algorithm is called adaptive as it remains optimal for all p and q.

4 ADAPTIVE SORTING ALGORITHMS

Here we plan to develop adaptive algorithms which will connect the AT^2 optimal sorting algorithm of Marberg and Gafni [9] on ordinary mesh to any constant time sorting algorithm on RM. The algorithms presented in this section were developed in [2] for a different architecture with different objective.

Lemma 3 *Let s items are stored in some s processors of a linear array of $m \geq s$ processors with reconfigurable bus. Then these items can be sorted in $O(s)$ time.*

Proof. A straightforward emulation of odd-even transposition sort [7, pages 139–144] solves the problem. □

Lemma 4 *Sorting m items in the first row of an RM of size $m \times m$ can be done in $O(1)$ time.*

Proof. See in [4, 17, 18]. □

The algorithm of Marberg and Gafni [9] uses a fixed number of phases of row/column sorting/rotating to sort ab items in $O(a+b)$ time on a mesh of size $a \times b$ where $a \geq \sqrt{b}$. If $a \ngeq \sqrt{b}$ then $b > \sqrt{a}$ and thus sorting can be done simply by transposing all row/column operations into column/row operations in the algorithm.

4.1 ***SORTING OF*** p ***ITEMS ON AN RM OF SIZE*** $k \times p$, $k \leq p$

Let the RM of size $k \times p$ be divided into $\frac{p}{k}$ submeshes of size $k \times k$ each and the given p items in the first row be distributed in such a way that each processor $PE_{i,jk}$, $0 \leq i < k$ and $0 \leq j < \frac{p}{k}$, receives an item. It is obvious that such a redistribution of elements can be carried out in constant time using a column broadcast followed by a row broadcast with bus splitting [12]. Now, the emulation of the sorting algorithm of Marberg and Gafni [9] needs only the following basic operations:

If $k \geq \sqrt{\frac{p}{k}}$

1. Sorting k items in a column using a submesh of size $k \times k$.
2. Rotating $\sqrt{\frac{p}{k}}$ items in a row using a submesh of size $1 \times k\sqrt{\frac{p}{k}}$.
3. Sorting/rotating $\frac{p}{k}$ items in a row using a submesh of size $1 \times p$.
4. Sorting $\sqrt{\frac{p}{k}}$ items in a column using a submesh of size $\sqrt{\frac{p}{k}} \times k$.

Else $(\Rightarrow \frac{p}{k} > \sqrt{k})$

5. Sorting $\frac{p}{k}$ items in a row using a submesh of size $1 \times p$.
6. Rotating $\sqrt{k}$ items in a column using a submesh of size $\sqrt{k} \times k$.
7. Sorting/rotating k items in a column using a submesh of size $k \times k$.
8. Sorting $\sqrt{k}$ items in a row using a submesh of size $1 \times k\sqrt{k}$.

The problem of rotation can always be transformed into a sorting problem without any slowdown. A rotation, therefore, takes as much time as it does to sort. Now, Operations 1, 4, 6, and 7 can be done in $O(1)$ time by Lemma 4. Using Lemma 3 it can be shown that operation 2 can be done in $O(\sqrt{\frac{p}{k}})$ time and operations 3, 5, and 8 can be done in $O(\frac{p}{k})$ time. Hence follows:

Theorem 1 *Given p items in the first row of an RM of size $k \times p$, $k \leq p$, these items can be sorted in $O(\frac{p}{k})$ time, which is AT^2 optimal.* □

4.2 SORTING OF n ITEMS ON AN RM OF SIZE $p \times q$, $p \leq q$ AND $pq = kn$

Let the RM of size $p \times q$ be divided into $\frac{q}{k}$ submeshes of size $p \times k$ each as suggested in Section 3 and the given n items in the first $\frac{n}{p}$ columns be distributed in such a way that each processor $PE_{i,jk}$, $0 \leq i < p$ and $0 \leq j < \frac{q}{k}$, receives an item. It can easily be shown that such a redistribution can be carried out in $O(\frac{n}{p}) = O(\frac{q}{k})$ time using only row broadcasts. Again, using the very similar techniques in Section 4.1, it can be shown that the emulation of the sorting algorithm of Marberg and Gafni on this $p \times q$ RM needs $O(\frac{q}{k})$ time as $p \leq q$.

Theorem 2 *Given n items in the first $\frac{n}{p}$ columns of an RM of size $p \times q$, $p \leq q$ and $pq = kn$, these items can be sorted in $O(\frac{q}{k})$ time, which is AT^2 optimal.* □

5 ADAPTIVE M-CONTOUR ALGORITHMS

Let the planar point at coordinate (i, j) be defined as $P(i, j)$. Again, let for any point p, $x(p)$ denote the x-coordinate and $y(p)$ denote the y-coordinate of p, e.g., $x(P(i, j)) = i$ and $y(P(i, j)) = j$.

Definition 1 *A point p* dominates *a point q (denoted by $q \prec p$) if $x(q) \leq x(p)$ and $y(q) \leq y(p)$. (The relation "$\prec$" is naturally called* dominance.*)*

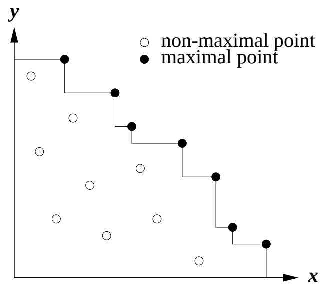

Figure 3: m-contour of a set of planar points.

Let S be a set of N planar points. To simplify the exposition of our algorithms, the points in S are assumed to be distinct.

Definition 2 *A point $p \in S$ is* maximal *if there is no other point $q \in S$ with $p \prec q$.*

We are interested in the contour spanned by the maximal elements of S, called the *m-contour* of S which can be obtained by simply sorting the maximal elements in ascending order of their x-coordinates (Figure 3). Let the m-contour of a set S be denoted as $m(S)$.

We have mentioned two interesting observations on m-contour in our paper [13, 14] which are given below for the sake of completeness.

Lemma 5 *Every m-contour is sorted in descending order of the y-coordinates.*

Proof. Suppose the contrary holds. Then there exists at least one pair of maximal elements p and q such that $y(p) < y(q)$ while $x(p) \leq x(q)$, which contradicts with the assumption that point p is maximal. □

Let for any set S of some planar points functions $min_x(S)$ and $max_x(S)$ denote the *minimum* and *maximum* x-coordinates in the set respectively. Let two more functions $min_y(S)$ and $max_y(S)$ be defined similarly w.r.t. y-coordinate.

Lemma 6 *Given K sets $S_0, S_1, \ldots S_{K-1}$ of planar points such that $\forall t : 0 \leq t < K-1$, $max_x(S_t) \leq min_x(S_{t+1})$, then $\forall i : 0 \leq i < K-1$, $\forall p \in m(S_i) \wedge y(p) > max_y(m(S_j))$, $\forall j > i$, if and only if, $p \in m(\bigcup_{t=0}^{K-1} S_t)$.*

Proof. The *necessity* part can be proved by arranging a contradiction of Lemma 5. To prove the *sufficiency* part we take a point $p \in m(S_i)$, $\exists i : 0 \leq i < K-1 \wedge p \notin m(\bigcup_{t=0}^{K-1} S_t)$. Then by the definition of maximality we get $\exists q \in \bigcup_{t=i+1}^{K-1} S_t$ such that $p \prec q$, i.e., $y(p) \leq y(q)$. □

The m-contour problem is also known as finding the maxima of a set of vectors and has been extensively explored for serial computers in [5, 6]. It is well known that the time complexity for computing the contour of the maximal elements of n planar points is $\Theta(n \log n)$ using a serial computer [6]. This lower boundary can be concluded from the fact that the problem of sorting can be easily transformed into an m-contour problem. The information content in computing m-contour of n planar points is $\Omega(n)$ and hence the AT^2 lower bound is $\Omega(n^2)$ [20, page 56]. Dehne [3] gives an AT^2 optimal algorithm for solving m-contour problem on a mesh of size $\sqrt{n} \times \sqrt{n}$ in $O(\sqrt{n})$ time. In [13, 14] we have presented three constant time m-contour algorithms on RM of various dimensions. Using the result of optimal simulation of multidimensional RM by two dimensional RM in [21], it can easily be shown that all the three algorithms in [13, 14] are AT^2 optimal.

We now plan to develop adaptive algorithms based on our AT^2 optimal constant time m-contour algorithm presented in [13, 14].

Given a binary sequence, b_j, $0 \leq j < N$, the *prefix-and computation* is to compute, $\forall i : 0 \leq i < N$, $b_0 \wedge b_1 \wedge \cdots \wedge b_i$. Similarly the *prefix-or computation* computes $b_0 \vee b_1 \vee \cdots \vee b_i$, $\forall i : 0 \leq i < N$. Adapting the technique of bus splitting [12] it is easy to show that:

Lemma 7 *Given a binary sequence of length m in the only row of an RM of size $1 \times m$, both the prefix-and and the prefix-or of the elements in the sequence can be computed in $O(1)$ time.*

Proof. See in [12]. □

Lemma 8 *Computing m-contour of m planar points in the first row of an RM of size $m \times m$ can be done in $O(1)$ time.*

Proof. First the points are sorted w.r.t. x-coordinate in constant time using Lemma 4. Using a column broadcast and a row broadcast, all the points are distributed in such a way that each column represents possible m pair-wise comparisons of a single point with the rest. The m-contour is then determined by computing the prefix-and of the comparison values in $O(1)$ time by Lemma 7. See in [13, 14] for detail. □

5.1 COMPUTING OF THE M-CONTOUR OF p PLANAR POINTS ON AN RM OF SIZE $k \times p$, $k \leq p$

Let the RM of size $k \times p$ be divided into $\frac{p}{k}$ submeshes of size $k \times k$ each and the given p planar points in the first row be distributed in such a way that each processor $PE_{i,jk}$, $0 \leq i < k$ and $0 \leq j < \frac{p}{k}$, receives a point. It is obvious that such a redistribution of elements can be carried out in constant time using a column broadcast followed by a row broadcast with bus splitting [12]. Now, we sort the points w.r.t. x-coordinate in column-major order by Theorem 1 in $O(\frac{p}{k})$ time.

Let the points residing in column jk be denoted by the set S_j, $0 \leq j < \frac{p}{k}$. Clearly these $\frac{p}{k}$ sets of planar points follow the condition of Lemma 6, i.e., $\forall j : 0 \leq j < \frac{p}{k} - 1$, $max_x(S_j) \leq min_x(S_{j+1})$. The m-contours $m(S_j)$, $0 \leq j < \frac{p}{k}$, are now computed in parallel using a submesh of size $k \times k$ for each computation. By Lemma 8 this operation takes only $O(1)$ time. Now, we transfer the $max_y(m(S_j))$ values to the first row of the RM in a single step by bus-splitting [12] using Lemma 5.

The m-contour of the entire p points can now be computed in the following steps using Lemma 6:

1. Iterate the following for $t = 0, 1, \ldots, \lceil \frac{p}{k^2} \rceil - 1$:
   - 1.1 Copy $max_y(m(S_{tk+j}))$ to processors $PE_{i,j+rk}$, $0 \leq i < k$, $0 \leq r < \frac{p}{k}$, for all $0 \leq j < k$, using a column broadcast then a row broadcast and finally a column broadcast.
   - 1.2 Copy the y-coordinate of the point residing in processor $PE_{i,kj}$ to the processors $PE_{i,kj+r}$, $0 \leq r < k$, for all $0 \leq j < \frac{p}{k}$, $0 \leq i < k$, using a row broadcast.
   - 1.3 Now in the jth submesh of size $k \times k$, the ith row contains k max_y values paired with the y-coordinate of a particular point, say d. Now, apply Lemma 6 to eliminate d by computing prefix-or over the comparison values of at most k pairs in constant time by Lemma 7.

It is very easy to show that the above iteration takes $O(\lceil \frac{p}{k^2} \rceil)$ time and thus it can be concluded that:

Theorem 3 *Given p planar points in the first row of an RM of size $k \times p$, $k \leq p$, the m-contour of these points can be computed in $O(\frac{p}{k})$ time, which is AT^2 optimal.* □

5.2 COMPUTING OF THE M-CONTOUR OF n PLANAR POINTS ON AN RM OF SIZE $p \times q$, $p \leq q$ AND $pq = kn$

Let the RM of size $p \times q$ be divided into $\frac{q}{k}$ submeshes of size $p \times k$ each and the given n planar points in the first $\frac{n}{p}$ columns be distributed in such a way that each processor $PE_{i,jk}$, $0 \leq i < p$ and $0 \leq j < \frac{q}{k}$, receives a point. It can easily be shown that such a redistribution can be carried out in $O(\frac{n}{p}) = O(\frac{q}{k})$ time using only row broadcasts. Now, we sort the points w.r.t. x-coordinate in column-major order by Theorem 2 in $O(\frac{q}{k})$ time.

Let the points residing in column jk be denoted by the set S_j, $0 \leq j < \frac{q}{k}$. Clearly these $\frac{q}{k}$ sets of planar points follow the condition of Lemma 6, i.e., $\forall j : 0 \leq j < \frac{q}{k} - 1$, $max_x(S_j) \leq min_x(S_{j+1})$. The m-contours $m(S_j)$, $0 \leq j < \frac{q}{k}$, are now computed in parallel using a submesh of size $p \times k$ for each computation. By Theorem 3 this operation takes only $O(\frac{p}{k})$ time.

Now taking very similar steps as used in Section 5.1 it can be shown that:

Theorem 4 *Given n planar points in the first $\frac{n}{p}$ columns of an RM of size $p \times q$, $p \le q$ and $pq = kn$, the m-contour of these points can be computed in $O(\frac{q}{k})$ time, which is AT^2 optimal.* □

6 CONCLUSION

In this paper we have shown that even with optimal slowdown, the resultant algorithm fails to remain AT^2 optimal when the reconfigurable mesh is self-simulated. To overcome this, we have introduced, for the first time, the idea of adaptive algorithm which runs on RM of variable sizes without compromising the AT^2 optimality. We have supported our idea by developing adaptive algorithms for sorting items and computing the contour of maximal elements of a set of planar points on RM.

REFERENCES

[1] Y.B. Asher, D. Gordon, and A. Schuster, Efficient self-simulation algorithms for reconfigurable arrays, *J. of Paral. and Dist. Comput.*, 30, 1995, 1–22.

[2] B. Beresford-Smith, O. Diessel, and H. ElGindy, Optimal algorithms for constrained reconfigurable meshes, *J. of Paral. and Dist. Comput.*, 39, 1996, 74–78.

[3] F. Dehne, $O(n^{1/2})$ algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer, *Info. Proc. Lett.*, 22, 1986, 303–306.

[4] J.-W. Jang and V.K. Prasanna, An optimal sorting algorithm on reconfigurable mesh, *J. of Paral. and Dist. Comput.*, 25, 1995, 31–41.

[5] H.T. Kung, On the computational complexity of finding the maxima of a set of vectors, In *15th Annual IEEE Symp. on Switching and Automata Theory*, 1974, 117–121.

[6] H.T. Kung, F. Luccio, and F.P. Preparata, On finding the maxima of a set of vectors, *J. ACM*, 22, 1975, 469–476.

[7] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, (San Mateo, California, USA: Morgan Kaufmann Publishers, 1992).

[8] T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE Trans. on Comput.*, C-34, 1985, 344–354.

[9] J.M. Marberg and E. Gafni, Sorting in constant number of row and column phases on a mesh, *Algorithmica*, 3, 1988, 561–572.

[10] M. Maresca, Polymorphic Processor Arrays, *IEEE Trans. on Paral. and Dist. Sys.*, 4, 1993, 490–506.

[11] R. Miller, V.K.P. Kumar, D.I. Reisis, and Q.F. Stout, Data movement operations and applications on reconfigurable VLSI arrays, In *Proc. Int. Conf. on Paral. Proc.*, 1988, 205–208.

[12] R. Miller, V.K. Prasanna-Kumar, D.I. Reisis, and Q.F. Stout, Parallel computations on reconfigurable meshes, *IEEE Trans. on Comput.*, 42, 1993, 678–692.

[13] M.M. Murshed and R.P. Brent, Constant time algorithm for computing the contour of maximal elements on the reconfigurable mesh, To appear in *Paral. Proc. Lett.*, special issue on Computing on Bus-Based Architecture.

[14] M.M. Murshed and R.P. Brent, Constant Time algorithms for computing the contour of maximal elements on the reconfigurable mesh, In *Proc. of the 1997 Int. Conf. on Paral. and Dist. Sys.*, Seoul, Korea, 1997, 172–177.

[15] M.M. Murshed and R.P. Brent, Algorithms for optimal self-simulation of some restricted reconfigurable meshes, In *Proc. of the 2nd Int. Conf. on Comput. Intelligence and Multimedia Appl. 1998*, Gippsland, Australia, 1998, 734–744.

[16] K. Nakano, A bibliography of published papers on dynamically reconfigurable architectures, *Paral. Proc. Lett.*, 5, 1995, 111–124.

[17] M. Nigam and S. Sahni, Sorting n numbers on $n \times n$ reconfigurable meshes with buses, *J. of Paral. and Dist. Comput.*, 23, 1994, 37–48.

[18] S. Olariu and J.L. Schwing, A novel deterministic sampling scheme with applications to broadcast-efficient sorting on the reconfigurable mesh, *J. of Paral. and Dist. Comput.*, 32, 1996, 215–222.

[19] C. Thompson and H. Kung, Sorting on a mesh-connected parallel computer, *Comm. of the ACM*, 20, 1977, 263–271.

[20] J.D. Ullman, *Computational Aspects of VLSI*, (Rockville, Maryland: Computer Science Press, 1984).

[21] R. Vaidyanathan and J.L. Trahan, Optimal simulation of multidimensional reconfigurable meshes by two-dimensional reconfigurable meshes, *Info. Proc. Lett.*, 47, 1993, 267–273.

[22] B.-F. Wang and G.-H. Chen, Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems, *IEEE Trans. on Paral. and Dist. Sys.*, 1, 1990, 500–507.