THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-98-02

Adaptive AT^2 Optimal Algorithms on Reconfigurable Meshes

M. Manzur Murshed and Richard P. Brent

March 1998

Joint Computer Science Technical Report Series

Department of Computer Science
Faculty of Engineering and Information Technology

Computer Sciences Laboratory
Research School of Information Sciences and Engineering

This technical report series is published jointly by the Department of Computer Science, Faculty of Engineering and Information Technology, and the Computer Sciences Laboratory, Research School of Information Sciences and Engineering, The Australian National University.

Please direct correspondence regarding this series to:

Technical Reports
Department of Computer Science
Faculty of Engineering and Information Technology
The Australian National University
Canberra ACT 0200
Australia

or send email to:

`Technical.Reports@cs.anu.edu.au`

A list of technical reports, including some abstracts and copies of some full reports may be found at:

`http://cs.anu.edu.au/techreports/`

Recent reports in this series:

TR-CS-98-01 Scott Milton. *Thread migration in distributed memory multicomputers*. February 1998.

TR-CS-97-21 Ole Møller Nielsen and Markus Hegland. *A scalable parallel 2D wavelet transform algorithm*. December 1997.

TR-CS-97-20 M. Hegland, S. Roberts, and I. Altas. *Finite element thin plate splines for surface fitting*. November 1997.

TR-CS-97-19 Xun Qu, Jeffrey Xu Yu, and Richard P. Brent. *Implementation of a portable-IP system for mobile TCP/IP*. November 1997.

TR-CS-97-18 Richard P. Brent. *Stability of fast algorithms for structured linear systems*. September 1997.

TR-CS-97-17 Brian Murphy and Richard P. Brent. *On quadratic polynomials for the number field sieve*. August 1997.

Adaptive AT^2 Optimal Algorithms on Reconfigurable Meshes

M. Manzur Murshed* Richard P. Brent

Computer Sciences Lab, Research School of Information Sciences & Engg.

The Australian National University, Canberra ACT 0200, Australia

E-mail: {murshed,rpb}@cslab.anu.edu.au

March 14, 1998

Abstract

Recently a few 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 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. We have also conjectured that to obtain an AT^2 algorithm to solve a problem of size n with $I(n)$ information content on an RM of size $p \times q$ where $pq = kI(n)$, it is sufficient to form buses of length $O(k)$. This conjecture has been supported by deriving a new algorithm, from our adaptive algorithm, to compute the contour of maximal elements of n planar points on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$.

*Corresponding author.

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 recently to a number of parallel machines to address this problem. Examples include the *Bus Automaton* [21], the *Reconfigurable Mesh (RM)* [13], the *content addressable array processor* [25], and the *Polymorphic torus* [12]. Among them RM draws much attention because of its simplicity. 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 communicational diameter of any network of processors to a constant. This fact has greatly influenced the researchers around the world and a large collection of constant time algorithms have already been developed [18]. 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, 17] with optimal slowdown for various models of RM.

In this paper, we have pointed out that self-simulation even with optimal slowdown compromises with the AT^2 [23, 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 an adaptive AT^2 optimal sorting algorithm to sort n items in $\frac{q}{k}$ time on an RM of size $p \times q$, $pq = kn$, $1 < p \leq q \leq n^2$, and $k \geq 1$. By using the same RM, we have also developed an adaptive AT^2 optimal algorithm to compute the contour of maximal elements of n planar points in $\frac{q}{k}$ time.

In developing our adaptive algorithm idea, we have conjectured that to obtain an AT^2 algorithm to solve a problem of size n with information content $I(n)$ on an RM of size $p \times q$ where $pq = kI(n)$, it is sufficient to form buses of length $O(k)$. This conjecture has been supported by deriving a new algorithm, from our adaptive algorithm, to compute the contour of maximal elements of n planar points on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$.

The paper is organised 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. A new algorithm is presented in Section 6 to compute the contour of maximal elements of n planar points on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$. Section 7 concludes the paper.

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. Throughout the paper, we use $\Theta()$ to mean

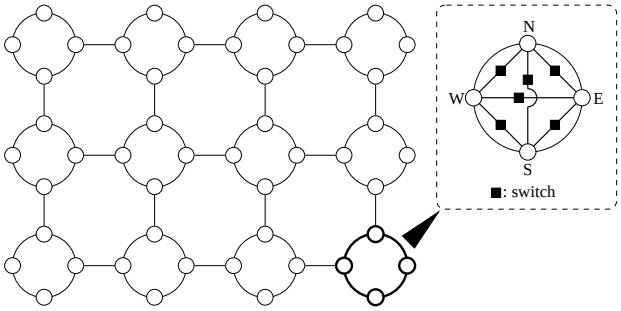

Figure 1: A reconfigurable mesh of size 3×4

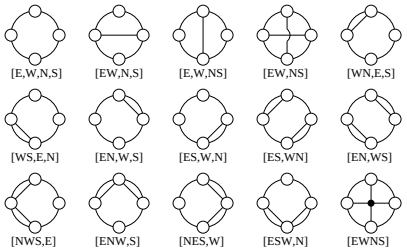

Figure 2: Possible internal connections between the four I/O ports of a processor

"order exactly," $O()$ to mean "order at most," and $\Omega()$ to mean "order at least."

2.1 Reconfigurable Mesh

The reconfigurable mesh 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 $PE_{i,j}$, $0 \leq i < X-1$, $0 \leq j < Y-1$ where processor $PE_{0,0}$ resides in the south-western corner. Each processor is connected to at most four neighbouring 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 [2].

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. A single time step of an RM is composed of the following four substeps:

BUS substep. Every processor switches the internal connectors between I/O ports by local decision.

WRITE substep. Along each bus, one or more processors on the bus transmit a message of length bounded by the bandwidth of the fixed bus segments as well as the switches. These processors are called the *speakers*. It is assumed that a collision between several speakers will be detected by all the processors connected to the bus and the transmitted message will be discarded.

READ substep. Some or all the processors connected to a bus read the message

transmitted by a single speaker. These processors are called the *readers*.

COMPUTE substep. A constant-time local computation is done by each processor.

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 [23], under the assumption that processors, switches, and links between adjacent switches occupy unit area.

2.2 Optimality Issues in 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.

Let $RM_C^{A\times B}$ denote a reconfigurable mesh of A rows and B columns with each PE having C registers.

Definition 1 *The self-simulation problem of RM is to step-by-step simulate* $RM_R^{M\times N}$ *by* $RM^{P\times Q}_{\Theta\left(R\lceil\frac{M}{P}\rceil\lceil\frac{N}{Q}\rceil\right)}$ *where* $P \leq M$, $Q \leq N$, *and the computing power of the PEs and the bus bandwidth (not less than* $\log MN$*) are assumed to be equivalent in both the meshes.*

To simplify the exposition $\frac{M}{P}$ and $\frac{N}{Q}$ are assumed to be integers. If the memory requirement of the simulating RM is bounded as in the above definition then the *slowdown* remains as the key issue.

Definition 2 *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 $RM_R^{M\times N}$ by $RM^{P\times Q}_{\Theta\left(R\frac{M}{P}\frac{N}{Q}\right)}$, $P \leq M$ and $Q \leq N$, is said to be optimal if the slowdown is $\Theta\left(\frac{M}{P}\frac{N}{Q}\right)$. A smaller slowdown would lead to a serial algorithm contradicting lower bound.

Ben-Asher *et al.* [1] first present the concept of self-simulation for RM and develop some self-simulation algorithms with optimal slowdown. In [17] 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 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 [6, 20, 19] 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 [10, 11, 22] 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* [23, pages 51-54]. If this problem $\mathcal{P}$ is realised in a VLSI circuit with aspect ratio α then, by Ullman [23, 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 $k \geq 1$. We are interested in developing an algorithm to solve $\mathcal{P}$ which will remain AT^2 optimal for all p and q.

Let T be the time to solve $\mathcal{P}$ in an RM of size $p \times q$. 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,

$$T = 1 \Leftrightarrow p = q = k = I(n). \qquad (2)$$

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:

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

Observation (2) represents the AT^2 optimal constant time algorithm, if it exists, with minimum area and observation (3) represents the AT^2 optimal mesh algorithm, if it exists, with minimum time. We now want to use the available algorithms complying with these observations (2) and (3) to develop AT^2 optimal algorithms for $1 \leq k \leq I(n)$ with solution time $1 \leq T \leq \sqrt{I(n)}$. Obviously minimum AT^2 lower bound can only be achieved when $p = q$.

Let the RM of size $p \times q$ be divided into $\frac{q}{k}$ submeshes of size $p \times k$ each. $I(n)$ information content should now be distributed in such a way that each submesh of size $p \times k$ receives exactly p elements of information content in a column of the submesh. The $\frac{q}{k}$-ary divide-and-conquer technique seems to be the most feasible where the problem will be divided into $\frac{q}{k}$ subproblems which are solved in the submeshes of size $p \times k$ in time $O(\frac{p}{k})$ in parallel. In the next steps the data are either redistributed (see Section 4) or merged (see Section 5) in time $O(\frac{q}{k})$ using the entire RM of size $p \times q$. There may be a constant number of similar iterations.

The idea of developing adaptive AT^2 optimal algorithms was originated in the work of Beresford-Smith *et al.* [3] in which they developed optimal algorithms for constrained reconfigurable meshes where only buses of some constant length is allowed. In fact the adaptive optimal sorting algorithm presented in the next section has been included from [3] where Beresford-Smith *et al.* did not point out the inner strength of their algorithm.

The value $k = \frac{pq}{I(n)}$ may have extra significance. Perhaps it can be conjectured, in the light of this paper and [3], that to obtain AT^2 optimal algorithm on RM of size $p \times q$, it is sufficient to form buses of length $O(k)$.

4 Adaptive Optimal Sorting Algorithm

Here we plan to develop an adaptive algorithm which will connect the AT^2 optimal sorting algorithm of Marberg and Gafni [11] on ordinary mesh to any constant time sorting algorithm on RM.

Lemma 1 *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 [9, pages 139–144] solves the problem. □

Lemma 2 *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 [6, 19, 20]. □

The algorithm of Marberg and Gafni [11] 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 Adaptive Optimal 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 [14]. Now, the emulation of the sorting algorithm of Marberg and Gafni [11] 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 $\frac{p}{k}$ items in a row using a submesh of size $1 \times p$.
4. Rotating $\frac{p}{k}$ items in a row using a submesh of size $1 \times p$.
5. Sorting $\sqrt{\frac{p}{k}}$ items in a column using a submesh of size $\sqrt{\frac{p}{k}} \times k$.

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

6. Sorting $\frac{p}{k}$ items in a row using a submesh of size $1 \times p$.
7. Rotating $\sqrt{k}$ items in a column using a submesh of size $\sqrt{k} \times k$.
8. Sorting k items in a column using a submesh of size $k \times k$.
9. Rotating k items in a column using a submesh of size $k \times k$.
10. 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, 5, 7, 8, and 9 can be done in $O(1)$ time by Lemma 2. Using Lemma 1 it can be shown that operation 2 can be done in $O(\sqrt{\frac{p}{k}})$ time and operations 3, 4, 6, and 10 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 Adaptive Optimal 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, the emulation of the sorting algorithm of Marberg and Gafni [11] needs only the following basic operations:

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

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

5. Sorting $\sqrt{\frac{q}{k}}$ items in a column using a submesh of size $\sqrt{\frac{q}{k}} \times k$.

Else $\Rightarrow \frac{q}{k} > \sqrt{p}$

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

Operations 1, 8, and 9 can be done in $O(1)$ time by Lemma 2 if $k \geq p$, else these can be done in $O(\frac{p}{k})$ time by Theorem 1. Using similar arguments it can be shown that operations 5 and 7 can be done in $O(\frac{p}{k})$ and $O(\frac{q}{k^2})$ respectively. Using Lemma 1, operations 3, 4, 6, and 10 can be done in $O(\frac{q}{k})$ time and operation 2 can be done in $O(\sqrt{\frac{q}{k}})$ time. Since $p \leq q$, it follows from the above argument that:

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 Optimal m-contour Algorithm

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 $y(P(i, j)) = j$.

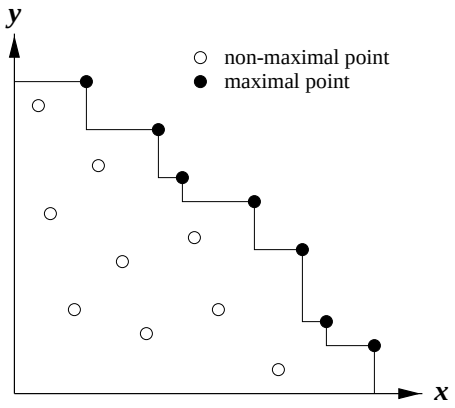

Figure 3: m-contour of a set of planar points

Definition 3 *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.*)*

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 4 *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 [15, 16] which are given below for the sake of completeness.

Lemma 3 *Every m-contour is sorted in descending order of the y-coordinates.*

Proof. Suppose the contrary. 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 4 *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 3. 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 [7, 8, 26]. Computation of maximal elements is important in solving the *Largest Empty Rectangle Problem* [5] where a rectangle R, and a number of planar points $S \in R$, are given and the problem is to compute the largest rectangle $r \subseteq R$ that contains no point in S and whose sides are parallel to those of R. If R is divided into four quadrants then the maximal elements w.r.t. the northeast(NE), northwest(NW), southwest(SW), and southeast(SE) directions as depicted in Figure 4 remain the only candidates to be the supporting elements of the empty rectangles lying in all the four quadrants.

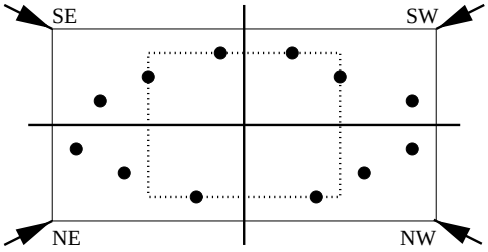

Figure 4: Importance of maximal elements in computing largest empty rectangle

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 [8]. 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 $\Omega(n^2)$ [23, page 56]. Dehne [4] 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 [15, 16] 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 [24], it can easily be shown that all the three algorithms in [15, 16] are AT^2 optimal.

We now plan to develop an adaptive algorithm based on our AT^2 optimal constant time m-contour algorithm presented in [15, 16].

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 [14] it is easy to show that:

Lemma 5 *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 [14]. □

Lemma 6 *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 2. 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 5. See in [15, 16] for detail. □

5.1 Adaptive Optimal 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 [14]. 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 4, 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 6 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 the following single step using Lemma 3:

1. b: Any processor in column j containing a point $\in m(S_j)$ disconnects all port interconnections while the rest of the processors connect port **N** with **S** for all $0 \leq j < \frac{p}{k}$.

   w: Any processor in column j containing a point $\in m(S_j)$ now writes the y-coordinate of the point to port §0 for all $0 \leq j < \frac{p}{k}$.

r: Every processor in the first row reads port **S** in.

Here, the substeps are labelled as "b:", "w:", "r:", and "c:" to denote the BUS, WRITE, READ, and COMPUTE substeps respectively.

Now, the m-contour of the entire p points can be computed in the following steps using Lemma 4:

1. Iterate the following for $t = 0$ to $\left\lceil \frac{p}{k^2} \right\rceil - 1$ in step 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 4 to eliminate d by computing prefix-or over the comparison values of at most k pairs in constant time by Lemma 5.

It is very easy to show that the above iteration takes $O(\left\lceil \frac{p}{k^2} \right\rceil)$ 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 Adaptive Optimal 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 4, 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 $p \times q$, $p \leq q$ and $pq = kn$, the m-contour of these points can be computed in $O(\frac{q}{k})$ time, which is AT^2 optimal.* □

6 A New AT^2 Optimal Algorithm for Computing m-contour of n Planar Points on $\sqrt{n} \times \sqrt{n}$ Ordinary Mesh

If we set $k = 1$ and $p = q$ in the adaptive algorithm associated with Theorem 4 we get an m-contour algorithm on an RM of size $\sqrt{n} \times \sqrt{n}$ in $O(\sqrt{n})$ time. But this algorithm cannot be used on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$ in $O(\sqrt{n})$ time as the max_y's of the m-contour of all the $\sqrt{n}$ subproblems must be broadcast sequentially to all other processors. This will result in an algorithm of order $O(n)$ which is not AT^2 optimal.

But in Section 3 we have conjectured that there should exist AT^2 optimal m-contour algorithm on ordinary mesh, where buses of length $O(1)$ are allowed to form, as we already have AT^2 optimal m-contour algorithm on RM with k = 1. In fact such an algorithm [4] exists and we have already mentioned it in Section 5. In [4], Dehne develop a recursive algorithm, on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$, which divides the m-contour problem of n points into two subproblems of equal size and then marge the solutions of these subproblems in affordable order $O(\sqrt{n})$ so that the overall order, $O(\sqrt{n})$, remains AT^2 optimal.

Is it possible to derive an $O(\sqrt{n})$ order algorithm from our adaptive m-contour algorithm associated with Theorem 4 with $k = 1$ and $p = q$ which will compute the m-contour of n points using buses of length $O(1)$? The following lemmas answer this question in affirmation.

Lemma 7 *Given p planar points, one in each processor of an ordinary linear array of p processors, the m-contour of these points can be computed in $O(p)$ time.*

Proof. Once the points are sorted w.r.t. x-coordinate in $O(p)$ time by Lemma 1, all the points are systolically shifted to the left for at most p times and each processor try to eliminate the point, it contains, from the m-contour by comparing it with the shifted points. □

Lemma 8 *Let m items $x_0, x_1, \ldots, x_{m-1}$ be given in the first row of an ordinary mesh of size $m \times m$ where item x_j resides in processor $PE_{0,j}$, $0 \leq j < m$. Now, consider the problem of distributing item x_j among all the processors $PE_{i,r}$, $0 \leq i < m$ and $0 \leq r < j$, for all $0 \leq j < m$. This problem can be solved in $O(m)$ time.*

Proof. Broadcast each item x_j, in parallel, to all the processors $PE_{i,j}$, $0 \leq i < m$, along the column j, $0 \leq j < m$. This takes $O(m)$ time. Now, systolically shift the items in each column to the left for most $O(m)$ time. □

7 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. We have also conjectured, with support from examples, that to obtain an AT^2 algorithm to solve a problem of size n with $I(n)$ information content on an RM of size $p \times q$ where $pq = kI(n)$, it is sufficient to form buses of length $O(k)$. We also have presented here a new algorithm to compute the contour of maximal elements of n planar points on an ordinary mesh of size $\sqrt{n} \times \sqrt{n}$.

References

[1] Yosi Ben Asher, Dan Gordon, and Assaf Schuster. Efficient self-simulation algorithms for reconfigurable arrays. *Journal of Parallel and Distributed Computing*, 30:1–22, 1995.

[2] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.

[3] B. Beresford-Smith, O. Diessel, and H. ElGindy. Optimal algorithms for constrained reconfigurable meshes. *Australian Computer Science Communications*, 17:32–41, 1995.

[4] Frank Dehne. $O(n^{1/2})$ algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer. *Information Processing Letters*, 22:303–306, 1986.

[5] Frank Dehne. Computing the largest empty rectangle on one- and two-dimensional processor arrays. *Journal of Parallel and Distributed Computing*, 9:63–68, 1990.

[6] Ju-Wook Jang and Viktor K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 25:31–41, 1995.

[7] 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*, pages 117–121, Oct. 1974.

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

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

[10] Tom Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C-34:344–354, 1985.

[11] John M. Marberg and Eli Gafni. Sorting in constant number of row and column

phases on a mesh. *Algorithmica*, 3:561–572, 1988.

[12] Massimo Maresca. Polymorphic processor arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4:490–506, 1993.

[13] Russ Miller, V. K. Prasanna Kumar, Dionisios I. Reisis, and Quentin F. Stout. Data movement operations and applications on reconfigurable VLSI arrays. In *Proc. International Conference on Parallel Processing*, pages 205–208, 1988.

[14] Russ Miller, V. K. Prasanna-Kumar, Dionisios I. Reisis, and Quentin F. Stout. Parallel computations on reconfigurable meshes. *IEEE Transactions on Computers*, 42:678–692, 1993.

[15] M. Manzur Murshed and Richard P. Brent. Constant time algorithm for computing the contour of maximal elements on the reconfigurable mesh. To appear in Parallel Processing Letters, special issue on Computing on Bus-Based Architecture.

[16] M. Manzur Murshed and Richard P. Brent. Constant time algorithms for computing the contour of maximal elements on the reconfigurable mesh. In *Proceedings of the 1997 International Conference on Parallel and Distributed Systems*, pages 172–177, Seoul, Korea, November 1997. Korea University, IEEE Computer Society.

[17] M. Manzur Murshed and Richard P. Brent. Algorithms for optimal self-simulation of some restricted reconfigurable meshes. In *Proceedings of the 2nd International Conference on Computational Intelligence and Multimedia Applications 1998*, pages 734–744, Gippsland, Australia, February 1998. Monash University, World Scientific.

[18] Koji Nakano. A bibliography of published papers on dynamically reconfigurable architectures. *Parallel Processing Letters*, 5:111–124, 1995.

[19] Madhusudan Nigam and Sartaj Sahni. Sorting n numbers on $n \times n$ reconfigurable meshes with buses. *Journal of Parallel and Distributed Computing*, 23:37–48, 1994.

[20] Stephan Olariu and James L. Schwing. A novel deterministic sampling scheme with applications to broadcast-efficient sorting on the reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 32:215–222, 1996.

[21] J. Rothstein. Bus automata, brains, and mental models. *IEEE Trans. Syst. Man Cybern*, 18:522–531, 1988.

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

[23] Jeffrey D. Ullman. *Computational Aspects of VLSI*. Computer Science Press, Rockville, Maryland, 1984.

[24] Ramachandran Vaidyanathan and Jerry L. Trahan. Optimal simulation of multidimensional reconfigurable meshes by two-dimensional reconfigurable meshes. *Information Processing Letters*, 47:267–273, 1993.

[25] C. C. Weems et al. The image understanding architecture. *Internat. J. of Comput. Vision*, 2:251–282, 1989.

[26] F. F. Yao. On finding the maximal elements in a set of plane vectors. Technical report, Comput. Sci. Dep. Rep., U. of Illinois at Urbana-Champaign, 1974.