Constant Time Algorithms for Computing the Contour of Maximal Elements on the Reconfigurable Mesh

M. Manzur Murshed and 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

Abstract

There has recently been an interest in the introduction of reconfigurable buses to existing parallel architectures. Among them Reconfigurable Mesh (RM) draws much attention because of its simplicity. This paper presents two $O(1)$ time algorithms to compute the contour of the maximal elements of N planar points on the RM. The first algorithm employs an RM of size $N \times N$ while the second one uses a 3-D RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$.

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* [15], the *Reconfigurable Mesh (RM)* [11], the *content addressable array processor* [16], and the *Polymorphic torus* [9]. A bus system is called *reconfigurable* if it can be dynamically changed according to either global or local information.

Jang *et al.* [17] have recently studied a number of constant time computational geometry algorithms on the reconfigurable mesh. In this paper we explore one further problem from a similar point of view. The problem, considered from computational geometry, is to compute the contour of the maximal elements of a given set of planar points (see Section 2.2). Solution to this problem is important in solving the *Largest Empty Rectangle Problem* [2, 5, 14] where a rectangle and a number of planar points in it are given and the problem is to compute the largest rectangle containing no points.

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 sequential computer [7]. Dehne [4] gives an efficient algorithm for solving this problem on a mesh of size $\sqrt{N} \times \sqrt{N}$ in $O(\sqrt{N})$ time. Whenever necessary N is assumed, throughout the paper, to be a perfect square without any loss of generality.

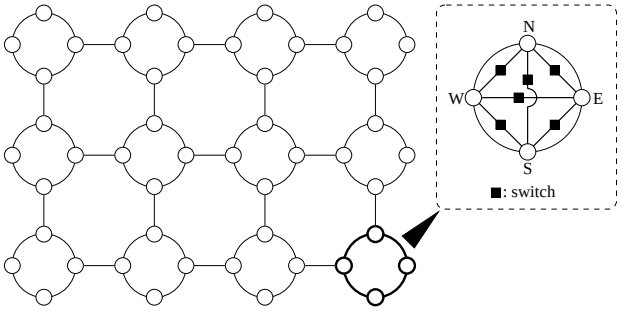

Figure 1: A reconfigurable mesh of size 3×4

In this paper we present two $O(1)$ time algorithms to compute the contour of the maximal elements of N planar points on the reconfigurable mesh. The first algorithm employs an RM of size $N \times N$ while the second one uses a 3-D RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$. To our knowledge this problem on the reconfigurable mesh is examined here for the first time.

This paper is organized as follows. In the next section we present the basic issues of RM as well as the definition of the problem. Constant time algorithms are developed in Section 3. Section 4 concludes the paper.

2 Preliminaries

For the sake of completeness, we briefly define the reconfigurable mesh and definitions of the problem of computing the contour of the maximal elements of a given set of planar points.

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

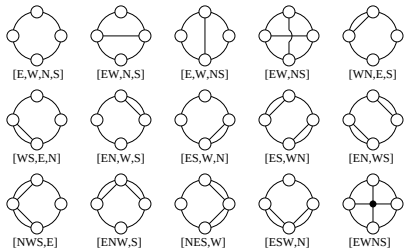

Figure 2: Possible internal connections between the four I/O ports of a processor

mesh connected computers. Processors of the RM of size $X \times Y$ are denoted by $PE_{i,j}$, $0 \leq i < X-1$, $0 \leq j < Y-1$. Each processor is connected to at most four neighbouring processors through fixed bus segments connected to four I/O ports $\mathcal{E}$ & $\mathcal{W}$ along dimension x and $\mathcal{N}$ & $\mathcal{S}$ 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 [1].

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.

Reconfigurable meshes of higher dimension can be constructed in a similar way. Processors of a 3-D RM of size $X \times Y \times Z$ are denoted by $PE_{i,j,k}$, $0 \leq i < X-1$, $0 \leq j < Y-1$, $0 \leq k < Z-1$. Each processor of a 3-D RM has two additional ports $\mathcal{U}$ and $\mathcal{D}$ along dimension z.

Many basic operations can be performed in constant time on RM. Below we briefly outline the results used in our algorithms in Section 3.

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:

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

Given N numbers of $\log N$ bits each, the problem of sorting these numbers on an RM of size $N \times N$ has been considered in [1, 6, 8, 13]. Several authors [3, 10, 13] consider an RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$ to sort N numbers. It is easy to show that these algorithms can be modified to sort N constant length records within a constant time reduction. Here the following lemmas are stated without proof.

Lemma 2 *Given N constant length records in a row, these records can be sorted in $O(1)$ time using an RM of size $N \times N$.*

Lemma 3 *Given N constant length records in a plane, these records can be sorted in $O(1)$ time using an RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$.*

2.2 Problem Definition

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.*)*

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 $p \prec q$.*

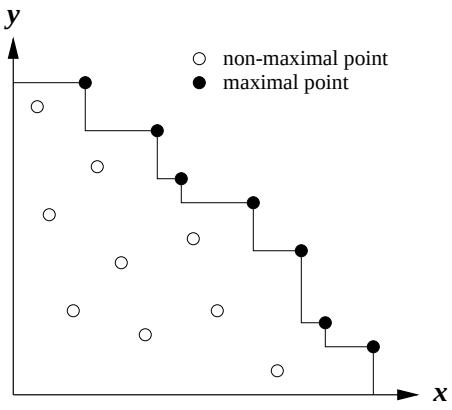

Figure 3: m-contour of a set of planar points

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 $m(S)$.

Two interesting observations on m-contour, connected to our algorithms, are given below:

Lemma 4 *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 5 *Given K sets s_0, s_1, $\ldots s_{K-1}$ of planar points such that $\forall k : 0 \leq k < K-1$, $max_x(s_k) \leq min_x(s_{k+1})$, 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, $p \in m(\bigcup_{k=0}^{K-1} s_k)$.*

Proof. The *necessity* part can be proved by arranging a contradiction of Lemma 4. To prove the *sufficiency* part we consider a point $p \in m(s_i)$, $\exists i : 0 \leq i < K-1 \wedge p \notin m(\bigcup_{k=0}^{K-1} s_k)$. Then by the definition of maximality we get $\exists q \in \bigcup_{k=i+1}^{K-1} s_k$ such that $p \prec q$, i.e., $y(p) \leq y(q)$. □

3 Constant Time Algorithms for Computing m-contour

We develop two constant time algorithms for computing the m-contour of a set of N planar points. The first algorithm MAXIMAL1 uses a 2-D RM of size $N \times N$ while the second algorithm MAXIMAL2 requires a 3-D RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$.

3.1 First Algorithm

The algorithm MAXIMAL1 is very straightforward. N planar points are given in the N processors at row 0. These points, after sorting, are distributed over the RM through column and row broadcast in such a way that each column i, $0 \leq i, < N$, of the RM computes the dominance of all other points over the ith point. Then each column computes the logical *or* of the previous dominance decision to assert whether the point represented by that column is a maximal point or not. As all the points are already sorted, the m-contour is obtained simply by following this sorted sequence. The detailed description of the algorithm is given below. Here and below we assume the following while presenting our algorithms for RM:

- In every step there may be at most four substeps labelled as "b:", "w:", "r:", and "c:" for the BUS, WRITE, READ, and COMPUTE substeps respectively.
- Any step without any labelled substep means the use of another algorithm, internal or external to this paper.
- In any step if the BUS substep is missing for a particular processor it is assumed that the local port interconnections of that processor remain unchanged.

Algorithm: MAXIMAL1

Precondition: registers r_0 and r_1 hold x- and y-coordinates respectively.

Postcondition: register r_2 holds the decision of maximality.

1. Sort the given N points at row 0 in ascending order of register r_0, i.e., in ascending order of the x-coordinates. The sorted list resides in the processors of row 0 in the ascending row-major order.
2. b: Every processor connects port $\mathcal{N}$ with $\mathcal{S}$;
   w: Every processor at row 0 writes register r_0 to port $\mathcal{N}$;

r: Every processor reads port $\mathcal{N}$ into register r_0;

3. w: Every processor at row 0 writes register r_1 to port $\mathcal{N}$;

   r: Every processor reads port $\mathcal{N}$ into register r_1;

4. b: Every processor connects port $\mathcal{E}$ with $\mathcal{W}$;

   w: Every processor $PE_{i,i}$, $0 \leq i < N$, writes register r_0 to port $\mathcal{E}$;

   r: Every processor reads port $\mathcal{E}$ into register r_2;

5. w: Every processor $PE_{i,i}$, $0 \leq i < N$, writes register r_1 to port $\mathcal{E}$;

   r: Every processor reads port $\mathcal{E}$ into register r_3;

6. b: Every processor $PE_{i,j}$, $0 \leq i,j < N$, does the following:

   if $i \neq j$ and $P(r_0, r_1) \prec P(r_2, r_3)$ **then**
   disconnect all the ports;
   else
   connect port $\mathcal{N}$ with $\mathcal{S}$;

   w: Every processor at row $N-1$ writes an arbitrary constant # to port $\mathcal{N}$;

   r: Every processor at row 0 reads port $\mathcal{S}$ into register r_2;

   c: Every processor at row 0 does the following:

   if $r_2 = \#$ **then**
   set register $r_2 = 1$;
   else
   set register $r_2 = 0$;

Theorem 1 *Given N planar points in a row, the m-contour of these points can be obtained in $O(1)$ time using an RM of size $N \times N$.*

Proof. Step 1 of algorithm MAXIMAL1 can be computed in constant time using Lemma 2. Steps 2-5 require $O(1)$ time. Step 6 is an elaboration of computing prefix-or partially and requires constant time by Lemma 1. □

3.2 Second Algorithm

In algorithm MAXIMAL2 we use the well-known divide-and-conquer approach to compute the m-contour. N points are given in the $(XY-plane)_{z=0}$. These points are sorted in order of x-coordinate to divide them into $\sqrt{N}$ disjoint sets of length $\sqrt{N}$ each. This division complies with the first condition in Lemma 5. Now the m-contour of these smaller sets are computed using all the 2-D submeshes lying parallel to $YZ-plane$. Now merging of the solutions of these smaller problems is done by carefully utilizing Lemma 5. Lemma 4 helps in getting the max_y of each smaller m-contour in constant time (Step 3). The ith max_y is then distributed over the $(XY\text{–}plane)_{z=i}$ (Step 4-6). Every point in each smaller m-contour then computes its overall maximality using the processors along the z-axis (Step 7-8). The detailed description of the algorithm is given below.

Algorithm: MAXIMAL2

Precondition: registers r_0 and r_1 hold x- and y-coordinates respectively.

Postcondition: register r_2 holds the decision of maximality.

1. Sort the given N points in the $(XY-plane)_{z=0}$ in ascending order of register r_0, i.e., in ascending order of the x-coordinates. The sorted list resides in the processors of the $(XY-plane)_{z=0}$ in ascending column-major order.

2. For every column i, $0 \leq i < \sqrt{N}$, of $(XY-plane)_{z=0}$ compute the m-contour of the $\sqrt{N}$ points residing in processors $PE_{i,j,0}$, $0 \leq j < \sqrt{N}$, using the algorithm MAXIMAL1 on the 2-D submesh lying in the $(YZ-plane)_{x=i}$.

3. b: Every processor in the $(XY-plane)_{z=0}$ does the following:

   if $r_2 = 0$ **then**
   connect port $\mathcal{N}$ with $\mathcal{S}$;
   else
   disconnect all the ports;

   w: Every processor in the $(XY-plane)_{z=0}$ does the following:

   if $r_2 = 1$ **then**
   write register r_1 to port $\mathcal{S}$;

   r: Every processor $PE_{i,0,0}$, $0 \leq i < \sqrt{N}$, reads port $\mathcal{S}$ into register r_3;

4. b: Every processor in the $(XZ-plane)_{y=0}$ connects port $\mathcal{U}$ with $\mathcal{D}$;

   w: Every processor $PE_{i,0,0}$, $0 \leq i < \sqrt{N}$, writes register r_3 to port $\mathcal{U}$;

   r: Every processor $PE_{i,0,i}$, $0 \leq i < \sqrt{N}$, reads port $\mathcal{U}$ into register r_3;

5. b: Every processor in the $(XZ-plane)_{y=0}$ connects port $\mathcal{E}$ with $\mathcal{W}$;

w: Every processor $PE_{i,0,i}$, $0 \leq i < \sqrt{N}$, writes register r_3 to port $\mathcal{E}$;

r: Every processor in the $(XZ\text{–}plane)_{y=0}$ reads port $\mathcal{E}$ into register r_3;

6. b: Every processor connects port $\mathcal{N}$ with $\mathcal{S}$;

w: Every processor in the $(XZ\text{–}plane)_{y=0}$ writes register r_3 to port $\mathcal{N}$;

r: Every processor reads port $\mathcal{N}$ into register r_3;

7. b: Every processor connects port $\mathcal{U}$ with $\mathcal{D}$;

w: Every processor in the $(XY - plane)_{z=0}$ writes register r_1 to port $\mathcal{U}$;

r: Every processor reads port $\mathcal{U}$ into register r_4;

8. b: Every processor $PE_{i,j,k}$, $0 \leq i,j,k < \sqrt{N}$, does the following:

if $k > i$ and $r_4 \leq r_3$ **then**
disconnect all the ports;
else
connect port $\mathcal{U}$ with $\mathcal{D}$;

w: Every processor in the $(XY\text{–}plane)_{z=\sqrt{N}-1}$ writes an arbitrary constant # to port $\mathcal{U}$;

r: Every processor in the $(XY\text{–}plane)_{z=0}$ reads port $\mathcal{D}$ into register r_4;

c: Every processor in the $(XY\text{–}plane)_{z=0}$ does the following:

if $r_2 = 1$ *and* $r_4 \neq \#$ **then**
set register $r_2 = 0$;

Theorem 2 *Given N planar points in a plane, the m-contour of these points can be obtained in $O(1)$ time using an RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$.*

Proof. Step 1 of algorithm MAXIMAL2 can be computed in constant time using Lemma 3. By Theorem 1 step 2 can also be done in constant time. It is obvious that the rest of the steps require $O(1)$ time. □

4 Conclusion

In this paper we have described two $O(1)$ time algorithms for computing the contour of the maximal elements of a given set of planar points. Hopefully these algorithms will have an application to solving the largest empty rectangle problem in constant time.

References

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

[2] B. Chazelle, R. L. Drysdale, and D. T. Lee. Computing the largest empty rectangle. *SIAM J. Comput.*, 15:300–315, 1986.

[3] Yen-Cheng Chen and Wen-Tsuen Chen. Constant time sorting on reconfigurable meshes. *IEEE Transactions on Computers*, 43:749–751, 1994.

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

[8] R Lin, Stephan Olariu, J. Schwing, and J. Zhang. A VLSI-optimal constant time sorting on reconfigurable mesh. In *Ninth European Workshop Parallel Computing*, pages 1–16, Spain, 1992.

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

[10] Mark S. Merry and Johnnie Baker. A constant time sorting algorithm for a three dimensional reconfigurable mesh and reconfigurable network. *Parallel Processing Letters*, 5:401–412, 1995.

[11] 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.

[12] 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.

[13] 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.

[14] M. Orlowski. A new algorithm for the largest empty rectangle problem. *Algorithmica*, 5:65–73, 1990.

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

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

[17] Ju wook Jang, Madhusudan Nigam, Viktor K. Prasanna, and Sartaj Sahni. Constant time algorithms for computational geometry on the reconfigurable mesh. *IEEE Transactions on Parallel and Distributed Systems*, 8:1–12, 1997.