Parallel Processing Letters

CONSTANT TIME ALGORITHMS FOR COMPUTING THE CONTOUR OF MAXIMAL ELEMENTS ON A RECONFIGURABLE MESH*

M. MANZUR MURSHED
Computer Sciences Lab, Research School of Information Sciences & Engineering
Australian National University, Canberra ACT 0200, Australia
E-mail: murshed@cslab.anu.edu.au

and

RICHARD P. BRENT†
Computer Sciences Lab, Research School of Information Sciences & Engineering
Australian National University, Canberra ACT 0200, Australia
E-mail: rpb@cslab.anu.edu.au

Received (received date)
Revised (revised date)
Communicated by (Name of Editor)

ABSTRACT

There has recently been an interest in the introduction of reconfigurable buses to existing parallel architectures. Among them the Reconfigurable Mesh (RM) draws much attention because of its simplicity. This paper presents three constant 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 $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$. We further extend the result to k-D RM of size $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$.

Keywords: Reconfigurable mesh; Parallel algorithm; Computational geometry

1 Introduction

It is well known that interprocessor communications and simultaneous memory accesses often act as bottlenecks in present-day parallel machines [1]. Bus systems have been introduced recently to a number of parallel machines to address this problem. Examples include the *Bus Automaton* [2], the *Reconfigurable Mesh (RM)* [3], the *content addressable array processor* [4], and the *Polymorphic torus* [5]. A bus system is called *reconfigurable* if it can be dynamically changed according to either global or local information.

*A preliminary version of this paper was presented at the 1997 International Conference on Parallel and Distributed Systems, Seoul, Korea.

†Current address: Oxford University Computing Laboratory, Oxford, OX1 3QD, England. E-mail: Richard.Brent@comlab.ox.ac.uk.

Jang *et al.* [6] 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). This problem is also known as finding the maxima of a set of vectors and has been extensively explored for serial computers in [7,8,9]. Computation of maximal elements is important in solving the *Largest Empty Rectangle Problem* [10] 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 Fig. 1 remain the only candidates to be the supporting elements of the empty rectangles lying in all the four quadrants.

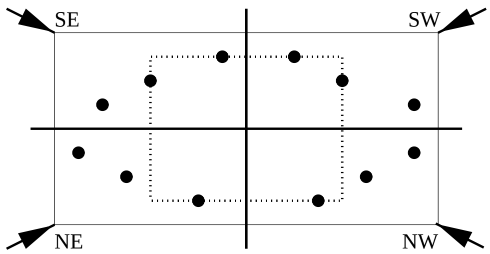

Fig. 1: Importance of maximal elements in computing largest empty rectangle

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 [7]. Dehne [11] gives an efficient algorithm for solving this problem on a mesh of size $\sqrt{N} \times \sqrt{N}$ in $O(\sqrt{N})$ time.

In this paper we present three constant 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 $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$. The second algorithm is then further extended on a k-D RM of size $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$. An $O(4^k)$ time algorithm is developed using only $O(N^{1+1/(k-1)})$ processors, where $0 < 1/(k-1) \ll 1$ for large N. To our knowledge this problem on the reconfigurable mesh is examined here for the first time.

Whenever necessary $N^{1/(k-1)}$ is assumed to be an integer where k is the dimension of the RM under consideration. This implies $k \leq \log_2 N + 1$.

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 give definitions of the problem of computing the contour of the maximal elements of a given set of planar points.

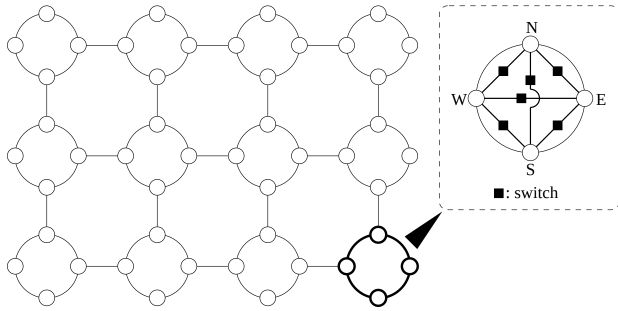

Fig. 2: A reconfigurable mesh of size 3×4

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 $PE_{i,j}$, $0 \le i < X-1$, $0 \le j < Y-1$. 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 Fig. 2) between the I/O ports of each processor. The fifteen possible interconnections of I/O ports through switching are shown in Fig. 3.The analysis of the behaviour of RM, like all bus systems, relies on the assumption that the transmission time of a message along a bus is independent of the length of the bus [12].

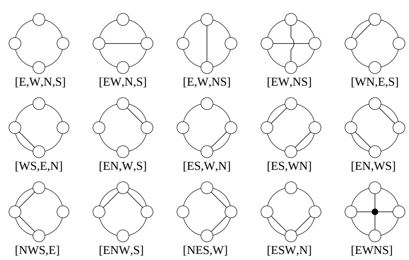

Fig. 3: Possible internal connections between the four I/O ports of a processor

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. 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. For example, processors of a 3-D RM of size $X \times Y \times Z$ are denoted by $PE_{i,j,k}$, $0 \leq i < X-1$, $0 \leq j < Y-1$, $0 \leq k < Z-1$ and each processor has two additional ports $\mathcal{U}$ and $\mathcal{D}$ along dimension z.

For convenience, we use the notation $PE_{*,i_2,i_3,\ldots,i_k}$ to denote the set of processors $\forall i_1 : PE_{i_1,i_2,i_3,\ldots,i_k}$. Similarly $PE_{*,i_2,*,i_4,\ldots,i_k}$ denotes the set of processors $\forall i_1 \forall i_3 : PE_{i_1,i_2,i_3,\ldots,i_k}$.

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 $\forall i : 0 \leq i < N$, $b_0 \vee b_1 \vee \cdots \vee b_i$. Adapting the technique of *bus splitting* [13] 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 [12,14,15,16]. Several authors [16,17,18] consider an RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$ to sort N numbers. Chen *et al.* [17] has further developed a constant time sorting algorithm on a k-D RM of size $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$. It is easy to show that these algorithms can be modified to sort N constant length records within a constant time reduction (the constant may depend on k). Here the following lemmas are stated without proof.

Lemma 2 *Given N constant length records in a row of processors, 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 of processors, these records can be sorted in $O(1)$ time using a 3-D RM of size $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$.*

Lemma 4 *Given N constant length records in a hyperplane of processors, these records can be sorted in $O(4^k)$ time using a k-D RM of size $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$, where $k \geq 4$.*

2.2 Problem Definition

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

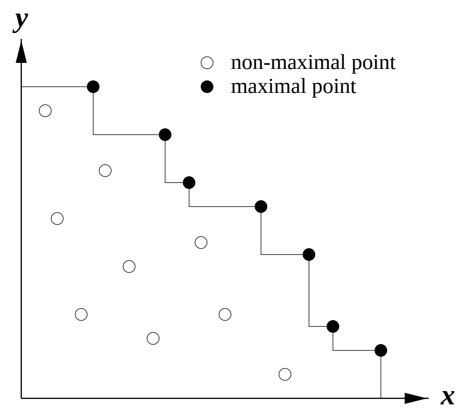

Fig. 4: m-contour of a set of planar points

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 $p \prec q$.*

The definition above actually defines maximality w.r.t. NE direction as depicted in Fig. 1. The definitions of maximality w.r.t. other directions are then obvious.

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 (Fig. 4). Let the m-contour of a set S be denoted by $m(S)$.

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

Lemma 5 *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 the assumption that point p is maximal. □

For any set S of some planar points, let functions $min_x(S)$ and $max_x(S)$ denote the *minimum* and *maximum* x-coordinates respectively in the set. 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 $\forall k : 0 \leq k < K-1$, $max_x(S_k) \leq min_x(S_{k+1})$, then $\forall i : 0 \leq i < K-1$, $\forall p \in m(S_i)(y(p) > max_y(m(S_j))$, $\forall j > i$, 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 5. To prove the *sufficiency* part we take a point $p \in m(s_i)$, $\exists i : 0 \leq i < K-1 \wedge p \notin m(\bigcup_{k=0}^{K-1} s_k)$. 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 three 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 $\sqrt{N} \times \sqrt{N} \times \sqrt{N}$. The third algorithm MAXIMAL3 is an extension of algorithm MAXIMAL2 to higher dimensions. An algorithm similar to algorithm MAXIMAL1 appears in [6] to compute three dimensional maxima.

3.1 First Algorithm

The algorithm MAXIMAL1 is very straightforward. N planar points are given in the row of processors $PE_{*,0}$. These points, after sorting, are distributed over the RM through column and row broadcast in such a way that each column of processors $PE_{i,*}$, $0 \leq i, < N$, compute the dominance of all other points over the ith point. Then each column computes the logical *and* 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 in the row of processors $PE_{*,0}$ in ascending order of register r_0, i.e., in ascending order of the x-coordinates. The sorted list also resides in the row of processors $PE_{*,0}$ in the ascending row-major order.

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

   w: Every processor $\in PE_{*,0}$ writes register r_0 to port $\mathcal{N}$;

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

3. w: Every processor $\in PE_{*,0}$ 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 $PE_{i,i}$, $0 \le i < N$, writes register r_0 to port $\mathcal{E}$;

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

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

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

6. b: Every processor $PE_{i,j}$, $0 \le i, j < N$, 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 $\in PE_{*,N-1}$ writes an arbitrary constant # to port $\mathcal{N}$;

   r: Every processor $\in PE_{*,0}$ reads port $\mathcal{S}$ into register r_2;

   c: Every processor $\in PE_{*,0}$ 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 of processors, 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 plane of processors $PE_{*,*,0}$. 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 6. Now the m-contour of the ith smaller set is computed using the 2-D submesh of processors $PE_{i,*,*}$, $0 \le i < \sqrt{N}$. Merging of the solutions of these smaller problems is then done by carefully utilizing Lemma 6. Lemma 5 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 plane of processors $PE_{*,*,i}$ (Steps 4–6). Every point in each smaller m-contour then computes its overall maximality using the processors along the z-axis (Steps 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 plane of processors $PE_{*,*,0}$ in ascending order of register r_0, i.e., in ascending order of the x-coordinates. The sorted list also resides in the plane of processors $PE_{*,*,0}$ in ascending column-major order.

2. For every column i, $0 \leq i < \sqrt{N}$, the m-contour of the $\sqrt{N}$ points residing in the ith column of processors $PE_{i,*,0}$ is computed using the algorithm MAXIMAL1 on the 2-D submesh of processors $PE_{i,*,*}$. Here step 1 of algorithm MAXIMAL1 should be ignored.

3. b: Every processor $\in PE_{*,*,0}$ does the following:
   if $r_2 = 0$ **then**
   connect port $\mathcal{N}$ with $\mathcal{S}$;
   else
   disconnect all the ports;

   w: Every processor $\in PE_{*,*,0}$ does the following:
   if $r_2 = 1$ **then**
   write register r_1 to port $\mathcal{S}$;

   r: Every processor $\in PE_{*,0,0}$ reads port $\mathcal{S}$ into register r_3;

4. b: Every processor $\in PE_{*,0,*}$ connects port $\mathcal{U}$ with $\mathcal{D}$;

   w: Every processor $\in PE_{*,0,0}$ 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 $\in PE_{*,0,*}$ 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 $\in PE_{*,0,*}$ reads port $\mathcal{E}$ into register r_3;

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

   w: Every processor $\in PE_{*,0,*}$ 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 $\in PE_{*,*,0}$ writes register r_1 to port $\mathcal{U}$;

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

8. b: Every processor $PE_{i,j,k}$, $0 \le i, j, k < \sqrt{N}$, does the following:
if $k > i$ and $r_4 \le r_3$ **then**
disconnect all the ports;
else
connect port $\mathcal{U}$ with $\mathcal{D}$;

w: Every processor $\in PE_{*,*,\sqrt{N}-1}$ writes an arbitrary constant # to port $\mathcal{U}$;

r: Every processor $\in PE_{*,*,0}$ reads port $\mathcal{D}$ into register r_4;

c: Every processor $\in PE_{*,*,0}$ does the following:
if $r_2 = 1$ *and* $r_4 \ne \#$ **then**
set register $r_2 = 0$;

Theorem 2 *Given N planar points in a plane of processors, the m-contour of these points can be obtained in* $O(1)$ *time using a 3-D 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. □

3.3 Extension to Higher Dimensions

Based on Lemma 5 and Lemma 6, the m-contour problem can be solved in a recursive way. We derive the m-contour algorithm on k-D RM from the m-contour algorithm on $(k-1)$-D RM. The size of the k-D RM adopted here is $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$. The algorithm developed for k-D RM is very similar to algorithm MAXIMAL2. In fact the technique applied in algorithm MAXIMAL2 is simply extended to higher dimensions. This extension is made possible by the availability of the sorting algorithm stated in Lemma 4. The detailed description of the algorithm is given below.

Algorithm: MAXIMAL3

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 hyperplane of processors $PE_{*,*,\ldots,*,0}$ in ascending order of register r_0, i.e., in ascending order of the x-coordinates. The sorted list also resides in the hyperplane of processors $PE_{*,*,\ldots,*,0}$.

2. For every $(k-2)$-flat i, $0 \le i < N^{1/(k-1)}$, the m-contour of the $N^{1/(k-1)}$ points residing in the ith $(k-2)$-flat of processors $PE_{i,*,\ldots,*,0}$ is computed on the $(k-1)$-D submesh of processors $PE_{i,*,\ldots,*,*}$. If $k-1 \ge 4$ we simply recursively use the algorithm MAXIMAL3 else algorithm MAXIMAL2 is used. Here step 1 of algorithms MAXIMAL2 and MAXIMAL3 should be ignored.

3. Using $k-2$ bus splittings based on register r_2, one along each axis except the 1st and the kth axes, the max_y of the m-contour of the ith subset, which is the first maximal element according to Lemma 5, is read into register r_3 of the processor $PE_{i,0,\ldots,0,0}$, $0 \leq i < N^{1/(k-1)}$.

4. Using a single broadcast along the kth axis transfer the content of register r_3 of the processor $PE_{i,0,\ldots,0,0}$ into register r_3 of the processor $PE_{i,0,\ldots,0,i}$, $0 \leq i < N^{1/(k-1)}$.

5. Using $k-1$ broadcasts, one along each axis except the kth axis, transfer the content of register r_3 of the processor $PE_{i,0,\ldots,0,i}$ into the register r_3 of the hyperplane of processors $PE_{*,*,\ldots,*,i}$, $0 \leq i < N^{1/(k-1)}$.

6. b: Every processor connects ports along the kth axis;

   w: Every processor $\in PE_{*,*,\ldots,*,0}$ writes register r_1 to the top port along the kth axis;

   r: Every processor reads the top port along the kth axis into register r_4;

7. Using a single bus splitting along the kth axis, based on registers r_3 and r_4, compute the overall maximality decision into register r_2 of all the processors $\in PE_{*,*,\ldots,*,0}$.

Theorem 3 *Given N planar points in a hyperplane of processors, the m-contour of these points can be obtained in $SORT(N,k) + O(k^2)$ time using a k-D RM of size $N^{1/(k-1)} \times N^{1/(k-1)} \times \cdots \times N^{1/(k-1)}$, $k \geq 4$ where $SORT(N,k)$ denotes the time required by the sorting algorithm used in step 1.*

Proof. Step 1 of algorithm MAXIMAL3 is used only once. Steps 3–7 take only $O(k)$ time. Considering the recursion in step 2, the overall complexity of algorithm MAXIMAL3 then can be expressed as $SORT(N,k)+O(k)+O(k-1)+\cdots+O(1) \cong SORT(N,k) + O(k^2)$. □

From Lemma 4, $SORT(N,k) = O(4^k)$. Thus, Theorem 3 gives constant time complexity for any fixed $k \geq 4$.

4 Conclusion

In this paper we have described three constant 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.

Acknowledgement

The authors express their thanks to the anonymous referees for their valuable comments.

References

1. Koji Nakano. A bibliography of published papers on dynamically reconfigurable architectures. *Parallel Processing Letters*, 5:111–124, 1995.
2. J. Rothstein. Bus automata, brains, and mental models. *IEEE Trans. Syst. Man Cybern*, 18:522–531, 1988.
3. 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.
4. C. C. Weems et al. The image understanding architecture. *Internat. J. of Comput. Vision*, 2:251–282, 1989.
5. Massimo Maresca. Polymorphic processor arrays. *IEEE Transactions on Parallel and Distributed Systems*, 4:490–506, 1993.
6. 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.
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. 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.
9. 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.
10. Frank Dehne. Computing the largest empty rectangle on one- and two-dimensional processor arrays. *Journal of Parallel and Distributed Computing*, 9:63–68, 1990.
11. 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.
12. Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.
13. 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.
14. Ju-Wook Jang and Viktor K. Prasanna. An optimal sorting algorithm on reconfigurable mesh. *Journal of Parallel and Distributed Computing*, 25:31–41, 1995.
15. 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.
16. 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.
17. Yen-Cheng Chen and Wen-Tsuen Chen. Constant time sorting on reconfigurable meshes. *IEEE Transactions on Computers*, 43:749–751, 1994.
18. 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.