THE AUSTRALIAN NATIONAL UNIVERSITY

TR-CS-98-06

Optimal Computation of the Contour of Maximal Elements on Mesh-Connected Computers

M. Manzur Murshed and Markus Hegland

July 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-98-05 M. Manzur Murshed. *Optimal computation of the contour of maximal elements on constrained reconfigurable meshes*. May 1998.

TR-CS-98-04 M. Wilson and K. Yap. *ACSys/RDN experiences with Telstra's experimental broadband network, second progress report*. April 1998.

TR-CS-98-03 M. D. Wilson, S. R. Taylor, M. Rezny, M. Buchhorn, and A. L. Wendelborn. *ACSys/RDN experiences with Telstra's experimental broadband network, first progress report*. April 1998.

TR-CS-98-02 M. Manzur Murshed and Richard P. Brent. *Adaptive AT^2 optimal algorithms on reconfigurable meshes*. March 1998.

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.

Optimal Computation of the Contour of Maximal Elements on Mesh-Connected Computers

M. Manzur Murshed and Markus Hegland

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

The Australian National University, Canberra ACT 0200, Australia

E-mail: murshed@cslab.anu.edu.au

July 1, 1998

Abstract

Dehne presented an optimal algorithm to compute the contour of the maximal elements of n planar points on a $\sqrt{n} \times \sqrt{n}$ mesh. We have calculated that Dehne's algorithm requires $23\sqrt{n}$ steps and we have been able to reduce the required steps to $19\sqrt{n}$ through pre-sorting and using an efficient strategy in dividing the mesh into halves. It has also been established that any implementation of Dehne's algorithm requires at least $15\sqrt{n}$ steps. We have further developed a new optimal algorithm which requires at most $10\sqrt{n}$ steps.

1 Introduction

Despite the large communication diameter, the mesh-connected computer, defined in Section 2.1, has been given considerable attention because of its simplicity, regularity of interconnection pattern, and modularity of the layout which make it an ideal model for VLSI applications. A large number of efficient algorithms have been developed on meshes for a variety of problems. Dehne [1] studied the problem of computing the contour of the maximal elements of a given set of planar points (Section 2.2) on a mesh. There he presented an optimal algorithm to compute the maximal contour of n planar points on a mesh of size $\sqrt{n} \times \sqrt{n}$.

Computation of the maximal contour is important in solving the *Largest Empty Rectangle Problem* [2] 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

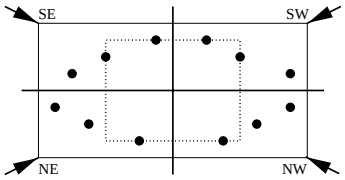

Figure 1: Importance of maximal elements in computing largest empty rectangle.

northeast(**NE**), northwest(**NW**), southwest(**SW**), and southeast(**SE**) directions as depicted in Figure 1 remain the only candidates to be the supporting elements of the empty rectangles lying in all the four quadrants.

In this paper we have estimated that the straighforward implementation of the recursive binary divide-and-conquer algorithm developed by Dehne in [1] to compute the maximal contour of n planar points on a $\sqrt{n} \times \sqrt{n}$ mesh requires $23\sqrt{n}$ steps. Utilising pre-sorting and an efficient strategy in dividing the mesh into halves, we have been able to reduce this complexity to $19\sqrt{n}$. It has also been established that any implementation of Dehne's algorithm must take at least $15\sqrt{n}$ steps. In order to further reduce the constant factor in the complexity, We have developed a new optimal non-recursive $\sqrt{n}$-ary divide-and-conquer algorithm which requires only $10\sqrt{n}$ steps.

This paper is organised as follows. In the next section we present the basic issues of the mesh-connected computers and give the definition of the problem of computing the maximal contour. Some relevant results and algorithms on sorting, transposing, broadcasting, and finding maxima on meshes are also discussed in the next section. Dehne's [1] optimal maximal contour algorithm is presented in Section 3 and the minimum achievable constant factor in the highest order term in the complexity of this algorithm has been worked out. A new optimal algorithm for the same problem is developed in Section 4 and it is shown that the constant factor in the highest order term in the complexity of this algorithm is much lower than that of Dehne's algorithm.

2 Preliminaries

For the sake of completeness, we briefly define the mesh-connected computer and give the definition of the problem of computing the contour of maximal elements of a set of planar points. We also discuss some relevant results and algorithms for the mesh.

In most of the cases, throughout this paper, we consider the constant of the highest order term but ignore the lower order additive terms in the complexity functions. To simplify exposition, we also assume $n = 4^k$ for some integer k whenever necessary.

2.1 Mesh-Connected Computer

The *mesh-connected computer* (*mesh*) of size $r \times c$ is a parallel computer with rc processors which are arranged in a $r \times c$ lattice. Let $PE_{i,j}$ denote the processor at row i and column j of a mesh of size $r \times c$ for all $0 \leq i < r$, $0 \leq j < c$ and let the processor $PE_{0,0}$ reside in the south-western corner. Every processor $PE_{i,j}$, for all $0 \leq i < r$, $0 \leq j < c$, is connected, via bidirectional unit-time communication links, to its four neighbours, processors $PE_{i\pm1,j\pm1}$, assuming they exist (Figure 2).

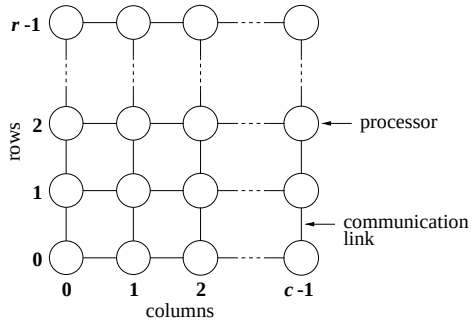

Figure 2: A mesh of size $r \times c$.

Each processor has a fixed number of registers (words) and can perform standard arithmetic and logical operations in unit-time. Each processor can also send or receive a word of data from each of its neighbour in unit-time.

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 finite set of 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 Figure 1. The definitions of maximality w.r.t. other directions are then obvious.

Let the set of all the maximal elements of S be denoted as $m(S)$. 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

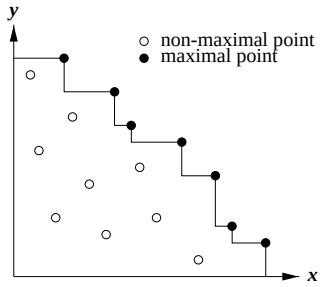

Figure 3: Maximal contour of a set of planar points.

elements in ascending order of their x-coordinates (Figure 3). Let $\tilde{m}(S)$ denote the m-contour of S. Then both $m(S)$ and $\tilde{m}(S)$ represent the same set except that $\tilde{m}(S)$ is ordered.

We have mentioned two interesting observations on m-contours in our papers [7, 8] which are given below for the sake of completeness.

Lemma 1 *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 P of some planar points functions $min_x(P)$ and $max_x(P)$ denote the *minimum* and *maximum* x-coordinates in the set respectively. Let two more functions $min_y(P)$ and $max_y(P)$ be defined similarly w.r.t. the y-coordinate.

Lemma 2 *Given K sets S_0, S_1, $\ldots S_{K-1}$ of planar points such that*

$$max_x(S_t) \leq min_x(S_{t+1}), \quad t = 0, \ldots, K-2$$

then, for every $p \in m(S_i)$, $i = 0, \ldots, K-1$,

$$p \in m\left(\bigcup_{t=0}^{K-1} S_t\right)$$

if and only if

$$y(p) > max_y(m(S_j)), \text{ for all } j = i+1, \ldots, K-1.$$

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

The m-contour problem is also known as the problem of finding the maxima of a set of vectors and has been extensively explored for serial computers in [3, 4, 14]. 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 [4]. This lower boundary can be concluded from the fact that the problem of sorting can be easily transformed into an m-contour problem of same size. The AT^2 [12] lower bound of m-contour problem of size n is thus $\Omega(n^2)$. Dehne [1] 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. Murshed & Brent [7, 8] have presented three constant time m-contour algorithms on RM of various dimensions. Using the result of optimal simulation of a multidimensional RM by a 2-dimensional RM in [13], it can easily be shown that all the three algorithms in [7, 8] are AT^2 optimal. An adaptive optimal m-contour algorithm on The reconfigurable mesh has been developed by Murshed & Brent [9]. Murshed & Brent [10] have further developed optimal m-contour algorithms on constrained reconfigurable meshes.

2.3 Sorting on Meshes

The following lemmas have been frequently referred to in the complexity analysis of the algorithms discussed in this paper.

Lemma 3 *Sorting of n items on a linear array of n processors can be done in $n-1$ steps which is the lowest possible.*

Proof. See the odd-even transposition sort in [5, Section 1.6.1]. □

Snake-like row(column)-major order is a special type of row(column)-major order where consecutive numbers in the ordering are adjacent in the mesh as shown in Figure 4.

Lemma 4 *Sorting of n items in snake-like row(column)-major order on an $\sqrt{n} \times \sqrt{n}$ mesh can be done in $3\sqrt{n}$ steps. This is also the lower bound.*

Proof. See [11]. □

Lemma 5 *Sorting of rc items in snake-like row-major order on a mesh of size $r \times c$, $c \leq r^2$ can be done in $r + 2c$ steps which is the lowest possible.*

Proof. See [11]. □

2.4 Transposing on a Specific Rectangular Mesh

Consider a mesh of size $\sqrt{n} \times \frac{\sqrt{n}}{2}$ where $\frac{n}{2}$ items are stored, one item per processor, in snake-like column major order. Let the problem of permutation, where the items will be rearranged in snake-like row-major order, be called *transpose*. The rearrangement sequence in transposing can start either from the top-left processor or from the bottom-right processor. Let these be denoted as *normal* and *inverted* transpose respectively as shown in Figure 4.

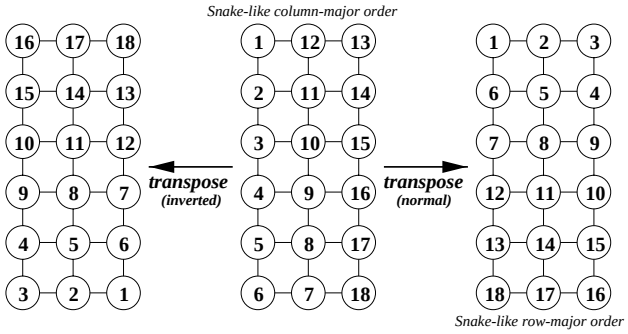

Figure 4: The transpose permutation.

Transpose can also be defined, in a similar fashion, for square meshes. Throughout this paper, if not stated otherwise, transposing will always be assumed to be normal on a rectangular mesh where the height is twice the width.

The general problem of permutation, which includes transpose as a simple case, can be solved in $\frac{3}{2}\sqrt{n}$ steps (the fewest possible) using a greedy algorithm where at each step, each item that still needs to move does so first along the column then along the row provided there is no contention for the same edge and the contention is resolved by the *farthest-first* protocol [5, Section 1.7.1]. The only problem with such greedy algorithm is the development of queues in the processors while resolving edge contentions. For solving the general problem of permutation, the queue size can become as large as $\frac{1}{3}\sqrt{n} - 3$ [5, Section 1.7.1]. The maximum queue size in transposing can be computed as follows:

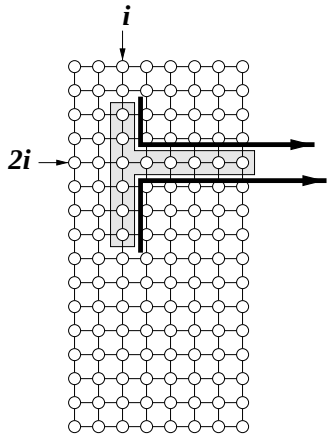

Figure 5: Development of queues in greedy transposing.

Consider the column i where items are stored from top to bottom sequence and also consider that after transposing the first n items will be stored from left to right in the row $2i$ (Figure 5). Now, the items in processors $PE_{i,i}$, $PE_{i+1,i}$, ..., $PE_{\frac{\sqrt{n}}{2}-1,i}$ will all be contending for the east-bound link of the

processor $PE_{2i,i}$ and thus will form a queue of size $\min\left(\frac{\sqrt{n}}{2} - 2i, i - 1\right)$ in the processor $PE_{2i,i}$. The maximum queue size in transposing then will be

$$\max_{i=0}^{\frac{\sqrt{n}}{2}-1} \min\left(\frac{\sqrt{n}}{2} - 2i, i - 1\right) = \frac{\sqrt{n}}{6} - 1 \ .$$

Lemma 6 *A $\sqrt{n} \times \frac{\sqrt{n}}{2}$ mesh, where the items are arranged in snake-like column major order can be transposed into snake-like row major order in $\frac{3}{2}\sqrt{n}$ steps with maximum queue size $\frac{\sqrt{n}}{6} - 1$.* □

2.4.1 Transposing Without Queue

The general problem of permutation can be transformed into the problem of sorting where keys will be the rank of the destination processor in some specific order. Considering Lemma 5, it is thus obvious that the general problem of permutation can be realized on a $r \times c$ mesh in $\max(r, c) + 2\min(r, c)$ routing steps [11]. The following lemma can then be easily concluded.

Lemma 7 *Transposing a mesh of size $\sqrt{n} \times \frac{\sqrt{n}}{2}$ can be done in $2\sqrt{n}$ steps without forming any queue.*

Transposing a mesh by means of sorting is not likely to be very efficient in practice, since for moderate values of n, the lower order term in the complexity of sorting (which is at least $5.35n^{3/8}$ [11]) is significant. An efficient queue-free transposing algorithm is thus presented below.

Algorithm $\mathcal{T}$
Pre: Each processor contains exactly one item and the items are in snake-like column-major order.
Post: Each processor contains exactly one item and the items are in snake-like row-major order.

1. Transpose the upper $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$ square submesh normally and the lower $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$ square submesh invertedly in parallel using the greedy algorithm.

2. Sort all the columns in parallel.

Correctness of this algorithm can be established easily using the fact that every column is transposed into two successive rows which are arranged in mutually inverted order.

To show that Algorithm $\mathcal{T}$ is queue-free, it is sufficient to prove that the greedy algorithm of transposing square meshes never forms any queue. Consider the column i which will be transposed into the row i. Now every item in the column i will be forwarded to the processor $PE_{i,i}$ but these will never contend for the same edge as the items above the processor $PE_{i,i}$ will be taking the west-bound link and the items below the processor $PE_{i,i}$ will be taking the east-bound link as shown in Figure 6.

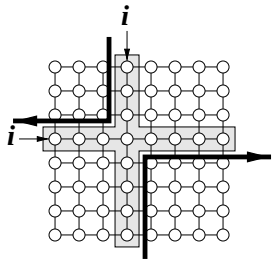

Figure 6: Greedy transposing of square meshes are queue free.

Lemma 8 *Algorithm $\mathcal{T}$ transposes a mesh of size $\sqrt{n} \times \frac{\sqrt{n}}{2}$ in exactly $2\sqrt{n}-1$ steps without forming any queue.*

Proof. It is obvious that the greedy algorithm, where at each step every item that still needs to move does so first along the column then along the row, takes at most $\sqrt{n}$ steps to transpose, normally or invertedly, a square mesh of size $\frac{\sqrt{n}}{2} \times \frac{\sqrt{n}}{2}$. By Lemma 3, step 2 of Algorithm $\mathcal{T}$ takes $\sqrt{n} - 1$ steps. □

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Shuffle row-major

0	2	8	10
1	3	9	11
4	6	12	14
5	7	13	15

Shuffle column-major

Figure 7: Shuffle row(column)-major ordering.

2.5 Shuffle Row(Column)-Major Sorting Order

There is no single natural ordering of a two-dimensional mesh, so many orderings viz, row(column)-major order, snake-like order etc. have been used. In [6] *shuffled row(column)-major ordering*, has been introduced which has, in excess of row(column)-major order, the property that the first quarter of the processors form one quadrant, the next quarter form another quadrant, etc., with this property holding recursively within each quadrant as shown in Figure 7. This shuffled property has been found very useful in developing parallel divide-and-conquer algorithms.

Lemma 9 *Sorting of n items in shuffle row(column)-major order on an $\sqrt{n} \times \sqrt{n}$ mesh can be done in $6\sqrt{n}$ steps.*

Proof. First the items are sorted in snake-like order in $3\sqrt{n}$ steps using Lemma 4 and then each processor computes the rank of the item it contains from it's address. Now, in each processor, the destination processor in shuffle order for each item is computed from it's rank in $O(\log n)$ time by the binary search technique. The snake-like order rank of the destination processor of

each item is then computed in constant time and the items are then sorted again in snake-like order using this rank as the key in $3\sqrt{n}$ steps. □

An algorithm that directly sorts items in shuffle row(column)-major order may take fewer steps and to our knowledge, it still remains an open problem.

2.6 Broadcasting and Finding Maxima on Meshes

The following lower bounds have been used in analysing the complexity of the algorithms presented in this paper.

Lemma 10 *Broadcasting the value of any processor $PE_{i,j}$ on a $r \times c$ mesh can be done in $r + c - 2$ steps.*

Proof. First broadcast along row i and then broadcast along all the columns in parallel. □

Lemma 11 *The maximum item on a $r \times c$ mesh can be accumulated into the processor $PE_{0,0}$ in $r + c - 2$ steps.*

Proof. Accumulate the maximum of each column into the processors at row 0 in $c - 1$ steps (Lemma 3) by sorting each column in parallel. Then accumulate the maximum of these maximums into the processor $PE_{0,0}$ in $r - 1$ steps (Lemma 3) by sorting row 0. □

3 Dehne's Algorithm

In [1] Dehne develops an optimal algorithm to compute the m-contour of n planar points on an $\sqrt{n} \times \sqrt{n}$ mesh.

Algorithm $\mathcal{D}$
Pre: Each processor contains exactly one point of S.
Post: The first $|m(S)|$ processors contain $\tilde{m}(S)$ in snake-like order.

1. Sort S in snake-like order w.r.t. the x-coordinate of the points.
2. Divide S into two disjoint subsets L and R of equal size with $x(l) \leq x(r)$ for all $l \in L$ and $r \in R$.
3. Shift all points of L and R, respectively, to the left and right half of the mesh.
4. Recursively compute $m(L)$ and $m(R)$ in parallel.
5. Set $m(S) \leftarrow m(R)$.
6. Get $max_y(m(R))$ in the right half of the mesh.

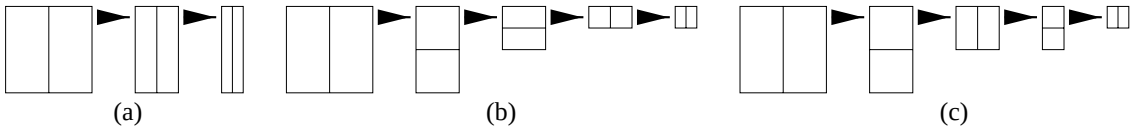

Figure 8: Recursive division of mesh into halves.

7. Broadcast $max_y(m(R))$ to all the processors in the left half of the mesh.
8. For all $l \in m(L)$: if $y(l) \geq max_y(m(R))$ then set $m(S) \leftarrow m(S) \cup \{l\}$.
9. Only at the first level of recursion, sort $m(S)$ in snake-like order w.r.t. the x-coordinate of the points to get $\tilde{m}(S)$.

In [1] Dehne claims that the complexity of Algorithm $\mathcal{D}$ satisfies

$$T(n) = T\left(\frac{n}{2}\right) + c\sqrt{n}$$

and thus Dehne concludes $T(n) = O(\sqrt{n})$.

In Section 4 we have developed a new optimal algorithm to compute the m-contour of n planar points on an $\sqrt{n} \times \sqrt{n}$ mesh. We can compare asymptotically optimal order algorithms by evaluating the exact constant factor in the highest order term of the complexity which the Big-Oh expressions hide.

If the recursive division of the mesh, into halves, is made only along rows(columns) as shown in Figure 8(a), steps 6 & 7 require $\sqrt{n}+o(\sqrt{n})$ steps (Lemma 11) as long as a half contains a full row(column) and the optimality will be lost. Hence, the recursive division of the mesh into halves must be made as Figure 8(b) or 8(c). Now, if the sorting order in step 1 is carefully chosen between snake-like row-major and snake-like column-major orders then steps 2 & 3 can be done in constant time. Step 8 always takes constant time. So, the following recurrence equation can be written for Algorithm $\mathcal{D}$ using Lemmas 4 & 5.

$$\left.\begin{aligned} T(n) &= \overbrace{3\sqrt{n}}^{\text{step 9}} + T'(n) \\ T'(n) &= \underbrace{\overbrace{3\sqrt{n}}^{\text{step 1}} + 2\overbrace{\left(\sqrt{n} + \frac{\sqrt{n}}{2}\right)}^{\text{steps 6 \& 7}}}_{\text{level 1}} + \\ &\quad \underbrace{\overbrace{2\sqrt{n}}^{\text{step 1}} + 2\overbrace{\left(\frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{2}\right)}^{\text{steps 6 \& 7}}}_{\text{level 2}} + T'(\tfrac{n}{4}) \end{aligned}\right\} \quad (1)$$

Solving (1) we get that Algorithm $\mathcal{D}$ requires $23\sqrt{n}$ steps.

Now, a careful observation will reveal that if the mesh is sorted before applying Algorithm $\mathcal{D}$ and the mesh is recursively divided as Figure 8(b) then step 1 is not necessary in all the odd levels of the recursion. Hence,

$$\left.\begin{array}{rcl} T(n) & = & 6\sqrt{n} + T'(n) \\ T'(n) & = & 7\sqrt{n} + T'(\frac{n}{4}) \end{array}\right\} \tag{2}$$

Solving (2) we get $20\sqrt{n}$ steps.

Again, if the mesh is sorted before applying Algorithm $\mathcal{D}$ then the purpose of step 1 in all the even levels of the recursion can also be achieved through transposing using the same number of steps by Lemma 7. By Lemma 8, replacing sorting in step 1 with transposing is more practical for moderate values of n. If a queue of size $\frac{\sqrt{n}}{6}$ is allowed in each processor then we can further reduce the steps required by Algorithm $\mathcal{D}$ to $19\sqrt{n}$ by using Lemma 6.

Interestingly, Algorithm $\mathcal{D}$ can also be executed in $19\sqrt{n}$ steps without forming any queue. The remedy is to sort the mesh in shuffle column-major order before applying Algorithm $\mathcal{D}$ and to follow Figure 8(c) in dividing the mesh. In this case step 1 is not at all necessary. Hence,

$$\left.\begin{array}{rcl} T(n) & = & 9\sqrt{n} + T'(n) \\ T'(n) & = & 5\sqrt{n} + T'(\frac{n}{4}) \end{array}\right\} \tag{3}$$

Theorem 1 *Any implementation of Algorithm $\mathcal{D}$ requires at least $15\sqrt{n}$ steps.* □

Proof. Steps 1–3 can be discarded if the items are pre-sorted in shuffle column-major order. Lower bounds of performing steps 6 & 7 are stated in Lemmas 10 & 11. As mentioned in Section 2.5, sorting n items directly in shuffle row(column)-major order on a $\sqrt{n} \times \sqrt{n}$ mesh may take $< 6\sqrt{n}$ steps. As the communication diameter of a $\sqrt{n} \times \sqrt{n}$ mesh is $2\sqrt{n} - 1$, sorting n items in shuffle column-major order on a $\sqrt{n} \times \sqrt{n}$ mesh can at best be done in $2\sqrt{n}$ steps. By Lemma 4, the lower bound of performing step 9 is $3\sqrt{n}$. Hence, the following recurrence equation holds.

$$\begin{array}{rcl} T(n) & = & \overbrace{2\sqrt{n}}^{\text{pre-sort}} + \overbrace{3\sqrt{n}}^{\text{step 9}} + T'(n) \\ T'(n) & = & \overbrace{\underbrace{2\left(\sqrt{n} + \frac{\sqrt{n}}{2}\right)}_{\text{level 1}} + \underbrace{2\left(\frac{\sqrt{n}}{2} + \frac{\sqrt{n}}{2}\right)}_{\text{level 2}}}^{\text{steps 6 \& 7}} + T'(\tfrac{n}{4}) \ . \end{array}$$

□

4 A New Optimal Algorithm

Dehne's Algorithm $\mathcal{D}$ is a binary divide-and-conquer algorithm based on Lemma 2 where K is assumed to be 2. Lemma 2 also suggests that it is possible to develop a divide-and-conquer m-contour algorithm where the problem will be divided recursively into more than two halves and the optimality of such an algorithm depends entirely on the complexity of merging the solutions of the divided subproblems. Moreover, recursion disappears when the problem is divided into $\sqrt{n}$ subproblems.

Algorithm $\mathcal{M}$
Pre: Each processor contains exactly one point of S.
Post: The first $|m(S)|$ processors contain $\tilde{m}(S)$ in snake-like order.

1. Sort S in snake-like column-major order w.r.t. the x-coordinate of the points.
2. Let the points in column c be denoted by the set S_c for all $0 \leq c < \sqrt{n}$. Compute $m(S_c)$, $0 \leq c < \sqrt{n}$, in parallel.
3. For all $0 \leq c < \sqrt{n}$: compute $max_y(m(S_c))$ in parallel.
4. Broadcast $max_y(m(S_c))$, $1 \leq c < \sqrt{n}$, to all columns i, $0 \leq i < c$.
5. For all $0 \leq c < \sqrt{n}$: for all $p \in m(S_c)$: if $y(p) > max_y(m(S_i))$, for all $i > c$ then set $m(S) \leftarrow m(S) \cup \{p\}$.
6. Sort $m(S)$ in snake-like order w.r.t. the x-coordinate of the points to get $\tilde{m}(S)$.

Theorem 2 *Algorithm $\mathcal{M}$ requires at most $10\sqrt{n}$ steps.*

Proof. By Lemma 4 steps 1 & 6 take $3\sqrt{n}$ steps each. Step 2 can be done in $\sqrt{n}$ steps in the following way:

After step 1 each column is sorted in either ascending or descending order. Every point in a column, say c, is systolically shifted to the descending direction for $\sqrt{n}-1$ times and each processor in column c then try to discard the point it contains, from the $m(S_c)$ by comparing it with the shifted points. The correctness of this procedure to compute $m(S_c)$ can easily be established using Lemma 2.

By Lemma 11 step 3 takes $\sqrt{n}-1$ steps. Step 4 can be done in $2\sqrt{n}$ steps as follows:

Every $max_y(m(S_c))$ is broadcast to all the processors in column c, for all $0 \leq c < \sqrt{n}$, in parallel taking $\sqrt{n}-1$ steps. Now, these maximum values are systolically shifted to the left in parallel for $\sqrt{n}-1$ times.

Step 5 takes constant time. □

5 Conclusion

Dehne [1] presented an optimal $O(\sqrt{n})$ order algorithm to compute the contour of the maximal elements of n planar points on a $\sqrt{n} \times \sqrt{n}$ mesh. We have calculated that Dehne's algorithm requires $23\sqrt{n}$ steps and we have been able to reduce this number to $19\sqrt{n}$ through pre-sorting and using an efficient strategy in dividing the mesh into halves. It has also been establishes that any implementation of Dehne's algorithm requires at least $15\sqrt{n}$ steps. We have further developed a new optimal algorithm which requires only $10\sqrt{n}$ steps.

References

[1] 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.

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

[3] 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.

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

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

[6] Russ Miller and Quentin F. Stout. Mesh computer algorithms for computational geometry. *IEEE Transactions on Computers*, 38:321–340, 1989.

[7] 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.

[8] 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.

[9] M. Manzur Murshed and Richard P. Brent. Adaptive AT^2 optimal algorithms on reconfigurable meshes. In *Proceedings of the 10th International Conference on Parallel and Distributed Computing and System*, Las Vegas, U.S.A., October 1998.

[10] M. Manzur Murshed and Richard P. Brent. Optimal computation of the contour of maximal elements on constrained reconfigurable meshes. Technical Report TR-CS-98-05, Joint Computer Science Tech. Report Series, The Australian National University, May 1998.

[11] C. P. Schnorr and A. Shamir. An optimal sorting algorithm for mesh connected computers. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 255–263, May 1986.

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

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

[14] 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.