

# Constant Time Algorithms for Computational Geometry on the Reconfigurable Mesh

Ju-wook Jang, Madhusudan Nigam,  
Viktor K. Prasanna, *Fellow, IEEE*, and Sartaj Sahni, *Fellow, IEEE*

**Abstract**—The reconfigurable mesh consists of an array of processors interconnected by a reconfigurable bus system. The bus system can be used to dynamically obtain various interconnection patterns among the processors. Recently, this model has attracted a lot of attention. In this paper, we show  $O(1)$  time solutions to the following computational geometry problems on the reconfigurable mesh: all-pairs nearest neighbors, convex hull, triangulation, two-dimensional maxima, two-set dominance counting, and smallest enclosing box. All these solutions accept  $N$  planar points as input and employ an  $N \times N$  reconfigurable mesh. The basic scheme employed in our implementations is to recursively find an  $O(1)$  time solution. The number of recursion levels and the size of the subproblems at each level of recursion are optimized such that the problem decomposition and the solution to the problem can be obtained in constant time. As a result, we have developed some efficient merge techniques to combine the solutions for subproblems on the reconfigurable mesh. These techniques exploit reconfigurability in nontrivial ways leading to constant time solutions using optimal size of the mesh.

## 1 INTRODUCTION

THE reconfigurable mesh [24] is a variant of the mesh connected computer model in which the shape of the buses can be altered to suit the need of the programs in execution. It shares some basic features with the CHiP computer [41], mesh connected computers augmented with broadcast buses [35], the bus automaton [39], the polymorphic-torus network [19], and the coterie network in the latest version of the Content Addressable Array Parallel Processor (CAAPP) [50]. A reconfigurable mesh has been built by NEC which has 512 PEs [18]. Researchers at IBM have implemented a reconfigurable mesh called polymorphic-torus [19]. An optical implementation of the reconfigurable mesh is suggested by Ben-Asher and Schuster [2]. This implementation employs Electrically controlled Directional Coupler (EDC) as a switch and optical fibers as buses.

On the theoretical side, many efficient parallel algorithms have been developed for the reconfigurable mesh. These include algorithms for fundamental data movement operations [24], [27], sorting [10], [1], [29], [22], [46], selection [7], [9], multiplication [13], division [34], histogramming [12], [17], [32], and graph problems [27], [47].

In this paper, we develop constant time algorithms for computational geometry problems including convex hull,  $k$ -dimensional maxima, two-set dominance counting, smallest

enclosing box, all-pairs nearest neighbor, and triangulation, all on a reconfigurable mesh of size  $N \times N$ . Preliminary versions of this paper have appeared in [11], [30], [31]. Previously, there has been a constant time algorithm for convex hull on a reconfigurable mesh [38]. The algorithm is flawed. We show a counter example and present a corrected and more general solution in Section 3.1.

The minimal size of the reconfigurable mesh needed to solve the computational geometry problems we consider in this paper in constant time is  $\Omega(N \times N)$ , where  $N$  is the number of input points. To achieve these time and processor bounds, we use the following strategy in our algorithms:

- 1) Divide a given problem of size  $N$  into subproblems of size  $N^{1-\epsilon}$ , where  $0 < \epsilon < 1$ . Solve each subproblem in constant time, using a mesh of size at most  $N \times N^{1-\epsilon}$ .
- 2) Merge the solutions to the subproblems in constant time using an  $N \times N$  mesh.

Our main contributions are in identifying the subproblems and in devising techniques on the reconfigurable mesh to solve the subproblems in constant time and to perform the merge operations in constant time. The rest of the paper is organized as follows. In Section 2, we describe the reconfigurable mesh model used in this paper and several variants of it. Also, prior work in performing several basic operations on the reconfigurable mesh is identified. Constant time algorithms for computational geometry are developed in Section 3 and Section 4 concludes the paper.

## 2 RECONFIGURABLE MESH MODEL

For the sake of completeness, we briefly define the reconfigurable mesh model and some variants of it.

### 2.1 Reconfigurable Mesh

The reconfigurable mesh architecture used in this paper is

- J. Jang is with the Department of Electronic Engineering, Sogang University, Seoul, Korea 121-742.
- M. Nigam and S. Sahni are with the Department of Computer and Information Science and Engineering, CSE 301, P.O. Box 116120, University of Florida, Gainesville, FL 32611.  
E-mail: sahani@cis.ufl.edu.
- V.K. Prasanna is with the Department of EE-Systems, EEB-244, University of Southern California, Los Angeles, CA 90089-2562.  
E-mail: prasanna@halycon.usc.edu.

Manuscript received Jan. 27, 1995.

For information on obtaining reprints of this article, please send e-mail to: [transpds@computer.org](mailto:transpds@computer.org), and reference IEEECS Log Number D95254.

based on the architecture defined in [24]. An  $N \times N$  reconfigurable mesh consists of an  $N \times N$  array of PEs (processors) connected to a grid-shaped reconfigurable broadcast bus. A  $4 \times 4$  reconfigurable mesh is shown in Fig. 1. Each PE has locally controllable bus switches. Internal connection among the four ports (N, E, W, and S) of a PE can be configured during the execution of algorithms. Note that there are 15 allowed connection patterns (see Fig. 2). For example, {SW, EN} represents the configuration in which S (South) port is connected to W (West) port while N (North) port is connected to E (East) port. Each bit of the bus can carry one of *1-signal* or *0-signal* at any time. The switches allow the broadcast bus to be divided into *sub-buses*, providing smaller reconfigurable meshes. For a given set of switch settings, a *subbus* is a maximal connected subset of PEs. Other than the buses and the switches, the reconfigurable mesh is similar to the standard two-dimensional Mesh Connected Computer (2-MCC). In this paper, we use the exclusive write model which allows only one PE to broadcast to a *subbus* shared by multiple PEs at any given time. We assume that the value broadcast consists of  $O(\log N)$  bits and a broadcast takes  $\Theta(1)$  time. Also, we assume that each PE can perform an arithmetic and logic operation on  $O(1)$  words in unit time. The size of the local storage in each PE is  $O(1)$  words, where each word is  $\Theta(\log N)$  bits.

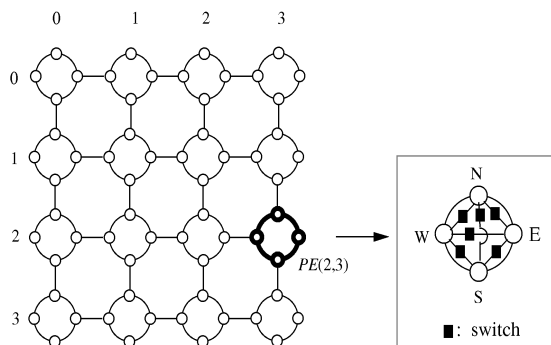


Fig. 1.  $4 \times 4$  reconfigurable mesh.

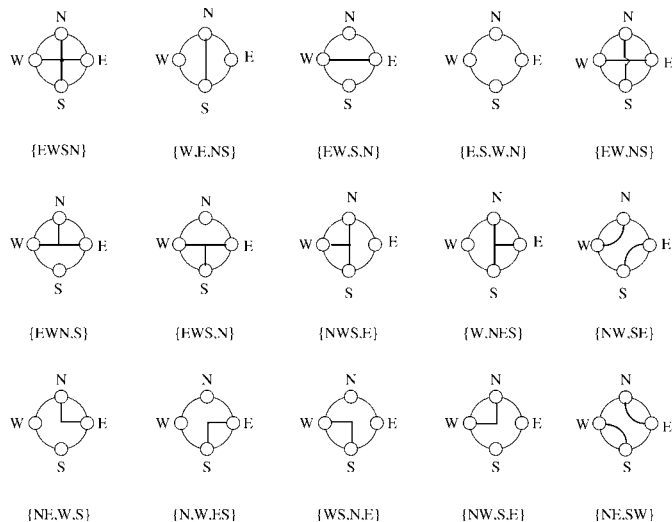


Fig. 2. Patterns of internal connections between the four I/O ports of a PE.

## 2.2 Related Models

After the definition of the reconfigurable mesh in [24], other models have been defined [1], [29], [46]. These models restrict the allowed connection patterns. The most general and powerful model among these is the PARBS model [46] which allows any combination of four-port connections in each PE. Notice that the PARBS model is the same as our model of Section 2.1.

The CAAPP [50] and RMESH [24] architectures appear to be quite similar. So, we shall describe the RMESH only. In this model, we have a bus grid with an  $N \times N$  arrangement of processors at the grid points (see Fig. 3 for a  $4 \times 4$  RMESH). Each grid segment has a switch on it which enables one to break the bus, if desired, at that point. When all the switches are closed, all  $N^2$  processors are connected by the grid bus. The switches around a processor can be set by using local information. If all processors disconnect the switch on their north, then we obtain row buses. Column buses are obtained by having each processor disconnect the switch on its east. The set of connection patterns supported by this model is shown in Fig. 4. Notice that in the RMESH model it is not possible to simultaneously have  $N$  disjoint row buses and  $N$  disjoint column buses that, respectively, span the width and height of the RMESH. It is assumed that processors on the same bus can communicate in  $O(1)$  time. RMESH algorithms for fundamental data movement operations and image processing problems can be found in [7], [15], [17], [24], [25], [27], [29], [38].

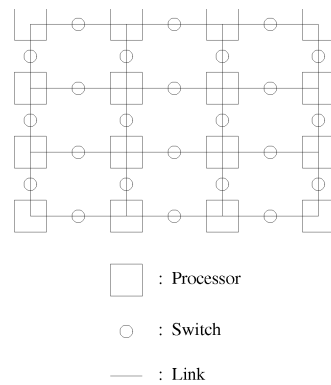


Fig. 3.  $4 \times 4$  RMESH.

The polymorphic torus architecture [19] is identical to the PARBS except that the rows and columns of the underlying mesh wrap around (Fig. 5). A variant of the reconfigurable mesh model has been studied in [20], [21] for arithmetic operations. The reconfigurable multiple bus machine (RMBM) has been studied in [43], [44], [45]. In this model, the reconfiguration hardware is separated from the processing elements.

In [1], the Reconfigurable Network (RN) model is introduced and several algorithms for this model are derived under the mesh restriction. This model has been denoted as MRN in [29] and LRN in [3]. In the MRN/LRN model, the number of possible connection patterns in each PE is 10. Fig. 6 shows the allowed connection patterns. A bit model of the reconfigurable mesh has been defined in [14].

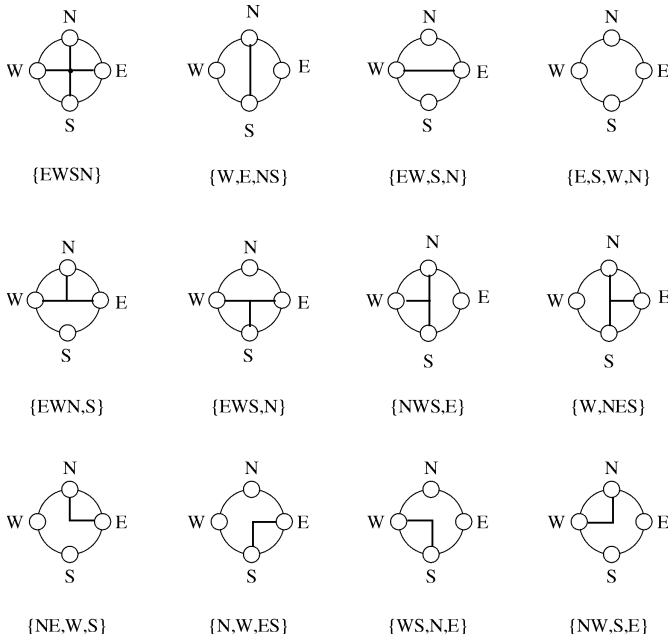


Fig. 4. Connection patterns allowed in RMESH.

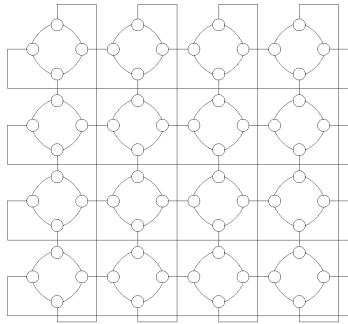


Fig. 5. 4 × 4 polymorphic torus.

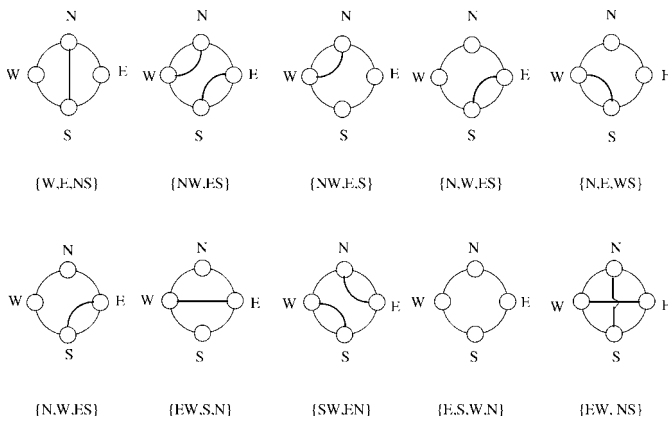


Fig. 6. Connection patterns allowed in MRN.

Throughout this paper, the term reconfigurable mesh refers to the model defined in Section 2.1.

### 2.3 Some Basic Operations

Several operations can be performed quickly on the reconfigurable mesh [1], [7], [9], [10], [12], [13], [15], [16], [17],

[28], [32], [47]. We briefly outline the results used in our algorithms in Section 3.

Given a 0/1 sequence,  $b_j$ ,  $0 \leq j < N$ , the *prefix-and computation* is to compute, for each  $i$ ,  $0 \leq i < N$ ,  $b_0 \wedge b_1 \wedge \dots \wedge b_i$ . Adapting the techniques in [27], it is easy to show:

LEMMA 1. *Given a 0/1 sequence of length  $N$  in the row of an  $1 \times N$  reconfigurable mesh, the prefix-and of the elements in the sequence can be computed in  $O(1)$  time.*

LEMMA 2 [27]. *Given a quadratonic sequence of length  $N$  in the row of an  $1 \times N$  reconfigurable mesh, the maximum and minimum of the elements in the sequence can be computed in  $O(1)$  time.*

The next operation considers computing the maximum of  $N \log N$ -bit numbers.

LEMMA 3 [27]. *Given a set of  $N \log N$ -bit numbers in a row of an  $N \times N$  reconfigurable mesh, the maximum of the elements in the set can be computed in  $O(1)$  time.*

Given a 0/1 sequence,  $b_j$ ,  $0 \leq j < N$ , the *prefix modular  $k$  computation* is to compute, for each  $j$ ,  $(\sum_{w=0}^j b_w) \bmod k$ .

LEMMA 2 [12]. *Given a 0/1 sequence of length  $N$  in a row, the prefix modular  $k$  computation can be performed in  $O(1)$  time on a  $(k + 1) \times 2N$  reconfigurable mesh.*

Note that the above result is not true for the RMESH model.

Given  $N$   $k$ -bit binary numbers,  $1 \leq k \leq N$ , the *addition problem* is to add these numbers into a  $(k + \log N)$ -bit binary number.

LEMMA 5 [10]. *Given  $N$   $k$ -bit binary numbers,  $1 \leq k \leq N$ , these numbers can be added in  $O(1)$  time on an  $N \times Nk$  reconfigurable mesh.*

Given  $N \log N$ -bit numbers, the problem of sorting these numbers has been considered by several authors.

LEMMA 6 [10], [22], [29], [1]. *Given  $N$  numbers in a row, these numbers can be sorted in  $O(1)$  time using an  $N \times N$  reconfigurable mesh.*

## 3 CONSTANT TIME GEOMETRY ALGORITHMS

In this section, we develop  $O(1)$  time solutions to several problems on  $N$  planar points using an  $N \times N$  reconfigurable mesh. We begin with computing the convex hull of  $N$  planar points.

### 3.1 Convex Hull

Given a set  $S$  of  $N$  planar points, the *convex hull* problem is to find the smallest convex polygon containing all the  $N$  points of  $S$ . Let  $CH(S)$  denote the convex hull of  $S$ . We solve the general problem of *prefix computation of convex hulls* of disjoint subsets of  $S$ .  $S$  is partitioned into  $\sqrt{N}$  disjoint subsets,  $S_j$ ,  $0 \leq j < \sqrt{N}$  of size  $\sqrt{N}$  using the  $x$  coordinates of the points as keys. The problem is to compute the convex hull of the union of the first  $j$  subsets, for each  $j$ ,  $0 \leq j < \sqrt{N}$ . Clearly, the solution to this problem includes the solution to the *convex hull* problem on  $S$ . Let  $E(S_j)$  denote the set of extreme

points of the convex hull of  $S_j$ . Thus, we compute  $E(\cup_{j=0}^j S_j)$  for each  $j$ ,  $0 \leq j < \sqrt{N}$ . The coordinates of  $N$  planar points are given as input and the outputs are the coordinates of the points in  $E(\cup_{j=0}^j S_j)$ , for  $0 \leq j < \sqrt{N}$ .

**THEOREM 1.** *Given  $N$  points stored in a row of the reconfigurable mesh, the prefix computation of convex hulls can be performed in  $O(1)$  time on an  $N \times N$  reconfigurable mesh.*

**PROOF.** First, the  $N$  points are sorted using their  $x$  coordinates as keys. The sorted list is partitioned into  $\sqrt{N}$  disjoint subsets of size  $\sqrt{N}$  each,  $S_j$ ,  $0 \leq j < \sqrt{N}$ .

Here, the  $x$  coordinate of any point in  $S_j$  is no greater than that of any point in  $S_{j+1}$ , for  $0 \leq j < \sqrt{N} - 1$ . The rest of computation is performed in two phases:

**Phase 1:** Compute the convex hulls of the subsets  $S_j$ ,  $0 \leq j < \sqrt{N}$  in parallel. Each convex hull is computed in  $O(1)$  time using a submesh of size  $N \times \sqrt{N}$ . Using Lemma 6, move the extreme points of each convex hull in  $O(1)$  time to a row of the submesh such that the points are sorted in angular order with respect to a point inside the convex hull.

**Phase 2:** Merge the convex hulls of  $S_0, S_1, \dots, S_j$  in parallel for each  $j$ ,  $0 \leq j < \sqrt{N}$ .

An algorithm to obtain the extreme points of  $S_q$ ,  $0 \leq q < \sqrt{N}$  is shown in the following. Initially, the  $\sqrt{N}$  points of  $S_q$  are stored in the top row of a submesh of size  $N \times \sqrt{N}$ . The basic idea is to consider all the lines arising from each pair of distinct points. For each line, check if all the other points lie on one side of the line. If so, declare the line joining the points as an edge of the convex hull. The set of endpoints of the detected edges is the set of extreme points. The steps are given below for each  $N \times \sqrt{N}$  submesh.

- 1) Use column buses to broadcast the points of  $S_q$  from row 0 to all other rows.
- 2) In row  $i = j\sqrt{N} + k$ ,  $0 \leq i < N$ , points  $j$  and  $k$  are broadcast to all processors (in row  $i$ ) using a row bus.
- 3) Using the points  $j$  and  $k$  received in Step 2, each processor computes  $a$ ,  $b$ , and  $c$  such that  $ax + by + c = 0$  defines the straight line through points  $j$  and  $k$ .
- 4) Let  $(u, v)$  be the coordinates of the point of  $S_q$  assigned to  $PE(e, f)$  in Step 1.  $PE(e, f)$  sets its *flag* to 1 if  $au + bv + c > 0$ , -1 if  $au + bv + c < 0$ , 0 if  $au + bv + c = 0$  and  $(u, v)$  is on the line segment connecting points  $j$  and  $k$  of Step 3 (this includes the cases when  $(u, v)$  is point  $j$  or  $k$ ), 2 otherwise.
- 5) Using row buses and row bus splitting,  $PE(i, 0)$ ,  $i = j\sqrt{N} + k$ ,  $0 \leq i < N$ , determines if there is a *flag* = 2 on row  $i$ . If so, the line segment connecting point  $j$  and point  $k$  is not an edge of the convex hull. If not, it determines if there is a 1 and later if there is a -1. If both a 1 and -1 are present, the line segment between point  $j$  and point  $k$  is not an edge of the convex hull. Otherwise it is.

- 6) If  $PE(i, 0)$ ,  $i = j\sqrt{N} + k$ ,  $0 \leq i < N$ ,  $j > k$  detects that the line segment between point  $j$  and point  $k$  is an edge of the convex hull, then using a row bus it broadcasts a 1 to  $PE(i, j)$  and  $PE(i, k)$ .
- 7) PEs that receive a 1 in Step 6 broadcast this to the PEs in row 0 of the same column (note 0 or 4 PEs in each column receive a 1 in Step 6; using column bus splitting, all but one of these may be eliminated to avoid concurrent writes to a column bus).
- 8) Now, PEs in row 0 mark the points of  $S_q$  they contain as being in or out of  $E(S_q)$  (note: a point is in  $E(S_q)$  if and only if it receives a 1 value in Step 7).
- 9) Using row bus splitting in row 0, any three extreme points are accumulated in  $PE(0, 0)$ . In case  $|E(S_q)| = 2$ , the remainder of this step is omitted. The centroid of these three points is computed. Since no three points of  $E(S_q)$  can be colinear, the centroid is an interior point of the convex hull.
- 10) The centroid computed in Step 9 is broadcast to all points of  $E(S_q)$  using a row bus. Each of these points computes the polar angle made<sup>1</sup> by the point using the centroid as the origin.
- 11) The points of  $E(S_q)$  are sorted by polar angle using Lemma 6.

Let  $BM(i, j)$  denote the  $(i, j)$ th block when the mesh is partitioned into  $N$  blocks of size  $\sqrt{N} \times \sqrt{N}$ ,  $0 \leq i, j < N$ . At the end of Phase 1,  $E(S_j)$  is routed to  $BM(j, j)$ , for  $0 \leq j < \sqrt{N}$ . In Phase 2, merge of the convex hulls is performed. Our merge technique involves masking vectors. The basic idea is to associate  $\sqrt{N}$  masking vectors for each set of extreme points and perform logical operations on the masking vectors instead of directly updating the extreme points in each step. The actual updating of the extreme points will be performed at the end of Phase 2 using the final masking vector for the corresponding set. Let  $M_{i,j}$  be the masking vector associated with  $BM(i, j)$  and  $M_{i,j}(r)$  be the  $r$ th element (a bit) of  $M_{i,j}$ .  $\sqrt{N}$  masking vectors,  $M_{0,j}, M_{1,j}, \dots, M_{\sqrt{N}-1,j}$  will be used for parallel updating of the extreme points of  $E(S_j)$ . Details will follow. At the end of Phase 2,  $BM(j, j)$  will delete the  $r$ th point of the  $E(S_j)$  if  $M_{j,j}(r)$  is 0,  $0 \leq r < \sqrt{N}$  and will output the updated  $E(S_j)$ .

#### Phase 2:

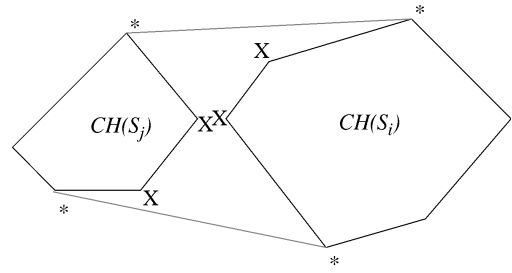
- 1) Initialize  $M_{i,j}(r) = 1$  if the  $r$ th point of  $S_j$  is in  $E(S_j)$ .
- 2) Refer to Fig. 7 for this step. For all  $i, j$ ,  $BM(i, j)$ ,  $i \neq j$ , identifies the upper and the lower tangent lines between  $CH(S_i)$  and  $CH(S_j)$ . First, we will show how to identify the upper tangent line. Broadcast  $E(S_i)$  and  $E(S_j)$  within  $BM(i, j)$  such that all the PEs in row(column)  $u(v)$  of  $BM(i, j)$  have copies of the  $u(v)$ th point of  $E(S_i)(E(S_j))$ , for  $0 \leq u, v < \sqrt{N}$ . For all  $u, v$ , the  $(u, v)$ th PE of  $BM(i, j)$  computes the

1. An alternative to using the polar angle is discussed on page 100 of [36].

slope of the line connecting the  $u$ th point of  $E(S_i)$  and the  $v$ th point of  $E(S_j)$ . Since the points of  $E(S_i)$  in row  $u$  is in angular order, the slopes in row  $u$  connecting the points of  $E(S_j)$  to the  $u$ th point of  $E(S_i)$  forms a tritonic (increase, decrease, and increase or decrease, increase, and decrease) sequence. So the maximum can be identified in  $O(1)$  time using Lemma 2. This can be simultaneously performed for all  $u$ ,  $0 \leq u < \sqrt{N}$ . As a result, we have up to  $\sqrt{N}$  maximums in  $BM(i, j)$ . The minimum of the  $\sqrt{N}$  maximums can be obtained in  $O(1)$  time using Lemma 3. The line associated with the minimum is the upper tangent line for  $CH(S_i)$  and  $CH(S_j)$ . The lower tangent line can be similarly obtained by identifying the maximum of minimums. Now set  $M_{ij}(r)$  to 0 if the  $r$ th point of  $E(S_j)$  is inside the polygon formed by the (up to four) tangential points (marked by '\*' in Fig. 7), for  $0 \leq r < \sqrt{N}$ . Create a set,  $T(i, j)$ , to hold the tangent lines. Now  $M_{ij}$  represents the extreme points of  $CH(S_j)$  which would survive the merge with  $CH(S_i)$ . The  $r$ th point of  $E(S_j)$  belongs to  $E(S_i \cup S_j)$  if  $M_{ij}(r) = 1$ .

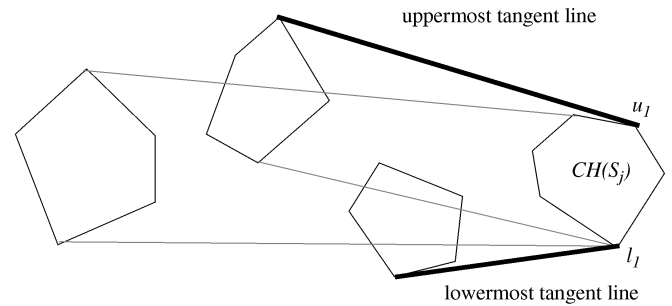
- 3) For each  $j$ ,  $0 \leq j < \sqrt{N}$ , combine  $\sqrt{N}$  blocks,  $BM(i, j)$ ,  $0 \leq j < \sqrt{N}$  into an  $N \times \sqrt{N}$  submesh. In this submesh, set  $M_{ij}(r) = M_{0j}(r) \wedge M_{1j}(r) \wedge \dots \wedge M_{ij}(r)$  for each  $r$ ,  $0 \leq r < \sqrt{N}$ . This can be performed in  $O(1)$  time using Lemma 1.
- 4) Note that, if  $j < i$ , it is still possible that there exist (up to two) points of  $E(S_j)$  that do not belong to  $E(S_0 \cup S_1 \cup \dots \cup S_i \cup S_j)$ , but not eliminated so far. One example is shown as  $u_1 (= u_2)$  in Fig. 8b. Broadcast  $T(i, j)$ ,  $0 \leq i, j < \sqrt{N}$  such that  $BM(i, j)$  has copies of  $T(i', j)$ ,  $0 \leq i' \leq i$ . For  $T(i, j)$ , refer to Step 2 of Phase 2. Identify the uppermost tangent line and the lowermost tangent line in  $T(i', j)$ ,  $0 \leq i' < j$  as illustrated in Fig. 8a. Let  $u_1(l_1)$  be the resulting upper(lower) tangent point on  $E(S_j)$ . This requires us to compute the maximum(minimum) of (up to  $\sqrt{N}$ ) slopes. This can be performed in  $O(1)$  time using  $BM(i, j)$  by Lemma 3. Similarly identify the uppermost(lowermost) tangent line in  $T(i', j)$ ,  $j < i' \leq i$  and let  $u_2(l_2)$  be the upper(lower) tangent point on  $E(S_j)$ . Compare  $u_1(l_1)$  against  $u_2(l_2)$ . If  $u_1 = u_2$  as in Fig. 8b, compute the minimum enclosing angle from  $u_1$ . The enclosing angle is defined to enclose all the tangent points associated the tangent lines in  $T(i', j)$ ,  $0 \leq i' \leq i$ . If the angle is greater than 180 degrees, set  $M_{ij}(r)$  to 0 when  $u_1$  is the  $r$ th point in  $E(S_j)$ . If  $u_1 \neq u_2$ , do nothing. Do similarly for  $l_1$  and  $l_2$ . At this time,  $M_{ij}(r)$  is equal to 1 if the  $r$ th point of  $E(S_j)$  belongs to  $E(S_0 \cup S_1 \cup \dots \cup S_i \cup S_j)$ .

- 5) (Updating the extreme points using the final masking vectors)  $BM(j, j)$ ,  $0 \leq j < \sqrt{N}$ , has set aside an original copy of  $E(S_j)$ . The points of  $E(S_j)$  are broadcast via column buses to  $BM(i, j)$ ,  $0 \leq i < \sqrt{N}$ . For all  $r$ ,  $0 \leq r < \sqrt{N}$ , for all  $j$ ,  $0 \leq i, j < \sqrt{N}$ ,  $BM(i, j)$  eliminates the  $r$ th point from  $E(S_j)$  if  $M_{ij}(r) = 0$ . Consider the submesh consisting of  $BM(j, i)$ ,  $0 \leq i < j$ . The set of points in this submesh that have survived is equal to  $E(\bigcup_{j'=0}^j S_{j'})$ .  $\square$

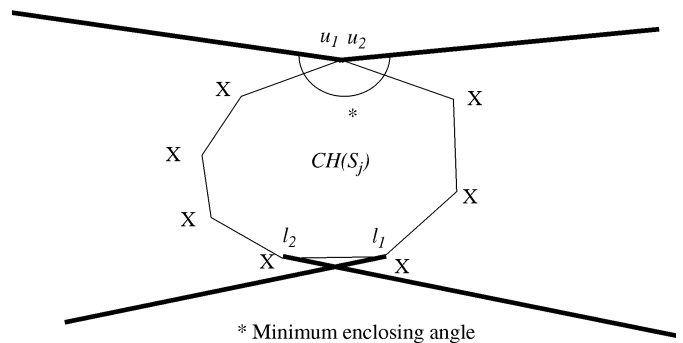


Tangents = broken lines  
 \* = Tangent points of  $CH(S_i)$  and  $CH(S_j)$   
 X = points to be eliminated

Fig. 7. Illustration of Step 3 of Phase 2.



(a) Computation of uppermost and lowermost tangent lines



(b) Elimination of non extreme tangent points

Fig. 8. Illustration of Step 4 of Phase 2.

As for the  $O(1)$  time convex hull algorithm in [38], we have found it has a flaw. The algorithm regards the union of the sets of nonextreme points resulting from the merge of all pairs of convex hulls (of disjoint subsets) as the nonextreme points of the convex hull enclosing all the convex hulls (of subsets). Step 5 in Phase 2 of our algorithm is missing in [38]. This would result in incomplete elimination of nonextreme points when more than two convex hulls are merged. Consider Fig. 8b as a counterexample. The nonextreme points marked "X" are eliminated in Step 3 of Phase 2 as well as in the convex hull algorithm in [38]. However, a point such as  $u_1 (= u_2)$  is not eliminated. Such points will be eliminated in Step 5 of Phase 2 in our algorithm. Further, our algorithm differs from [38] in that it provides prefix computation of convex hulls of  $\sqrt{N}$  subsets while the algorithm in [38] computes the convex hull of the union of the  $\sqrt{N}$  subsets. Theorem 1 implies:

**COROLLARY 1.** *Given  $N$  points stored in a row of the reconfigurable mesh, the convex hull of these points can be computed in  $O(1)$  time on an  $N \times N$  reconfigurable mesh. Further, the extreme points can be stored in a row in angular order with respect to a point inside the convex hull.*

### 3.2 Smallest Enclosing Box

It is well known [8] that the smallest enclosing rectangle of a set of  $N$  planar points has at least one side that is an extension of an edge of the convex hull. Hence, the smallest enclosing rectangle may be found by first computing the convex hull; then determining for each convex hull edge, the smallest enclosing rectangle that has one side which is an extension of this edge; and finally determining the smallest of these rectangles. The algorithm is shown in Fig. 9.

### 3.3 Triangulation

Given a set  $S$  having  $N$  planar points, the *triangulation problem* is to join them by nonintersecting straight-line segments (called *triangulating edges*) so that every region internal to the convex hull of  $S$  is a triangle. The input is  $N$  points and the output is the coordinates of the endpoints of the *triangulating edges*. Note that the number of triangulating edges is  $O(N)$ . Without loss of generality, assume that each point has distinct  $x$  and  $y$  coordinates and no four points are colinear.

Our algorithm runs in the following two phases:

**Phase 1** (Solve subproblems) Sort the  $N$  points in  $S$  by their  $x$  coordinates. Partition them into  $\sqrt{N}$  subsets of  $\sqrt{N}$  points each,  $S_i$ ,  $0 \leq i < \sqrt{N}$  such that, the  $x$  coordinate of any point in  $S_i$  is greater than the  $x$  coordinate of any point in  $S_{i-1}$ ,  $1 \leq i < \sqrt{N}$ . Each subset is assigned to a submesh of size  $N \times \sqrt{N}$ . Triangulate all the subsets in parallel.

**Phase 2** (Merge) Triangulate the region(s) formed between the convex hull of  $S$  and the convex hulls of  $S_i$ ,  $0 \leq i < \sqrt{N}$ .

In Phase 1, note that the the number of inputs is  $\sqrt{N}$  and the size of the available mesh is  $N \times \sqrt{N}$ . This enables us to develop a simple  $O(1)$  time algorithm for each subproblem. Then we develop a merge technique which results in an

Step 1: Find the convex hull of the  $N$  points.

Step 2: [Construct convex hull edges]

- (a) The convex hull points are broadcast to all rows using column buses. These are saved in variable  $R$  of each processor.
- (b) PE[ $i$ ,  $i$ ] broadcasts its  $R$  value using a row bus to the  $S$  variable of all processors on row  $i$ ,  $1 \leq i \leq p$ ,  $p =$  number of convex hull points.
- (c) PE[ $i$ ,  $i + 1$ ] broadcasts its  $R$  value using a row bus to the  $T$  variable of all processors on row  $i$ ,  $1 \leq i \leq p - 1$ . For  $i = p$ , PE[ $p$ , 1] does this. {Note: Now each PE in row  $i$  contains the same edge of the convex hull in its  $S$  and  $T$  variables.}

Step 3: [Determine area of minimum rectangle for each edge]

- (a) Using its  $R$ ,  $S$ , and  $T$  values, each PE computes the perpendicular distance  $D$  between point  $R$  and the straight line defined by the points  $S$  and  $T$ . Since, the  $R$ s are in convex hull order, the  $D$  values in a row form a tritonic sequence whose minimum,  $h$ , can be found in  $O(1)$  time using row bus splitting.  $h$  is the minimum height of the rectangle for the row edge.
- (b) Let  $P$  be the perpendicular through the middle of the edge defined by points  $S$  and  $T$ . Each processor computes the perpendicular distance of its point  $R$  from the infinite line  $P$  (use negative distances for points on one side of  $P$  and positive distances for points on the other side). These distances form a quadratonic sequence and its maximum,  $d_{max}$  and minimum,  $d_{min}$  can be found in  $O(1)$  time using row bus splitting.
- (c) The minimum area,  $A$ , of the rectangle for row  $i$  is  $h^2(d_{max} - d_{min})$ . Let this be stored in the  $A$  value of PE[ $i$ , 1].

Step 4: [Determine overall minimum rectangle]

Compute the minimum of the  $A$ s of PEs [1, 1],  $1 \leq i \leq p$ . This is done by forming the cross product of the  $A$ 's in a  $p \times p$  sub array and comparing the two  $A$ s in each PE.

Fig. 9. Minimum enclosing rectangle.

$O(1)$  time algorithm to triangulate  $N$  points using an  $N \times N$  mesh.

#### 3.3.1 $O(1)$ Time Solution Using $N^2 \times N$ Mesh

Let  $S$  denote the given set of  $N$  planar points. The basic scheme is to divide the convex hull of  $S$  into  $N - 2$  special polygons (*spolygons*) and triangulate the polygons in parallel. The *spolygons* are suitable for  $O(1)$  time triangulation using the broadcast feature of reconfigurable mesh. The *spolygon* is a special type of polygon formed between a convex hull and a point,  $p_0$ , outside the convex hull as illustrated (by the shaded region) in Fig. 10.  $\overline{p_0 q_0}$  represents the upper tangent line from  $p_0$  to the convex hull, while  $\overline{p_0 q_4}$  represents the lower tangent line.

**LEMMA 7.** *Given  $N$  planar points stored in a row of the reconfigurable mesh, the triangulation problem can be solved in  $O(1)$  time on an  $N^2 \times N$  reconfigurable mesh.*

**PROOF.** Initially, the  $N$  points are assumed to be stored in PE(0,  $i$ ),  $0 \leq i < N$ . Using Lemma 6, sort  $S$  into ascending order using the  $x$  coordinates as keys. Let  $p_i$  be the  $i$ th point in the sequence and let  $x_i$  ( $y_i$ ) be the  $x$  ( $y$ ) coordinate of  $p_i$ . Let  $CH_i$ ,  $1 \leq i \leq N - 3$  denote the convex hull of the points  $p_i$ ,  $i \leq i < N$ . Let  $EX_i$  denote the set of extreme points of  $CH_i$ ,  $1 \leq i \leq N - 3$ . Then,

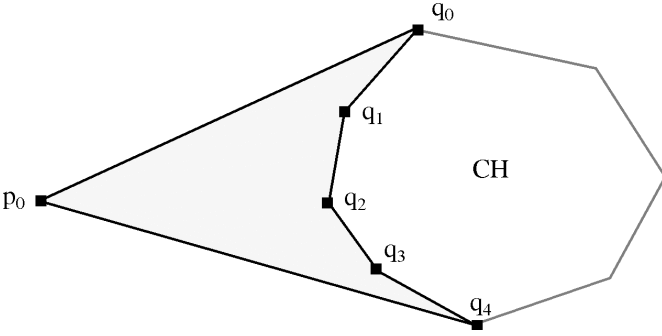


Fig. 10. A spolygon.

the *triangulation problem* for  $S$  can be reduced to triangulation of all the *spolygons* formed between  $p_i$  and  $CH_{i+1}$  for  $0 \leq i \leq N - 4$ . Each *spolygon* is determined and triangulated by a submesh of size  $N \times N$ .

Divide the reconfigurable mesh into  $RM(i)$ ,  $0 \leq i < N$  such that  $RM(i)$  consists of  $PE(iN + j, k)$ ,  $0 \leq j, k < N$ . Broadcast the  $N$  sorted points in the top row to  $RM(i)$ ,  $0 \leq i < N$  such that the top row of  $RM(i)$  receives  $(x_k, y_k)$ ,  $i \leq k < N$ .  $RM(i)$  computes  $CH_{i+1}$ ,  $0 \leq i \leq N - 4$ . Using Corollary 1, this is completed in  $O(1)$  time.  $EX_{i+1}$  is stored in the leftmost column of  $RM(i)$ . Using  $RM(i)$ , sort  $EX_{i+1}$  in angular order with respect to a point inside  $CH_{i+1}$ . This is done in parallel for  $0 \leq i \leq N - 4$ . Using Lemma 6, this can be completed in  $O(1)$  time.

The *spolygon* formed by  $(x_i, y_i)$  and  $CH_{i+1}$  is determined by  $RM(i)$  by computing the upper and lower tangents from  $(x_i, y_i)$  to  $CH_{i+1}$ . This can be completed in  $O(1)$  time as in the proof of Theorem 1. Mark all the lines joining  $(x_i, y_i)$  and the extreme points between the upper and lower tangents as *triangulating edges*. Note that  $RM(N - 3)$  has only three points. The three edges connecting the points are marked as *triangulating edges*. Now there are at most  $N - 1 - i$  *triangulating edges* in  $RM(i)$ . The *triangulating edges* from  $RM(i)$ ,  $0 \leq i \leq N - 3$  can be collected into the top row of the entire mesh in  $O(1)$  time. Let  $NT(i)$  denote the number of *triangulating edges* in  $RM(i)$ .  $NT(i)$  can be computed in  $O(1)$  time using Lemma 5. Note that,

$$\sum_{i=0}^{i=N-3} NT(i) = 2N - 3,$$

for  $N \geq 4$ . To assign the columns to be used for sending edges perform prefix summation on  $NT(i)$ ,  $0 \leq i \leq N - 3$ . This can be performed in  $O(1)$  time using Lemma 5.  $RM(i)$  uses columns

$$\left( \sum_{k=0}^{k=i-1} NT(k) \right) \pmod{N}$$

through

$$\left( \sum_{k=0}^{k=i} NT(k) - 1 \right) \pmod{N}$$

to send its edges to the top row.  $\square$

### 3.3.2 $O(1)$ Time Merge

Let  $S$  be the given set of  $N$  points. Assume that  $S$  is divided into  $S_i$ ,  $0 \leq i < \sqrt{N}$  such that, the  $x$  coordinate of any point in  $S_i$  is greater than the  $x$  coordinate of any point in  $S_{i-1}$ , for  $1 \leq i < \sqrt{N}$ . Using Lemma 7, the triangulation of each  $S_i$ ,  $0 \leq i < \sqrt{N}$  can be performed in  $O(1)$  time simultaneously for all  $i$  on an  $N \times N$  reconfigurable mesh. To complete the triangulation of  $S$ , we only need show how to triangulate the region(s) formed between the convex hull of  $S$  and the convex hulls of  $S_i$ ,  $0 \leq i < \sqrt{N}$ . Now we regard the convex hull of a subset of points as a *superpoint* and employ a similar scheme as in Section 3.3.1. One difference is that the polygons formed between a *superpoint* and the convex hull of a set of *superpoints* need not be a *spolygon*. For this we partition the region(s) into  $\sqrt{N} - 1$  disjoint polygons of a special type called *dpolygons*, and triangulate them in parallel. The *dpolygon* is a polygon formed between two disjoint convex hulls as shown in Fig. 11a.  $\overline{v_0 w_0}$  represents the upper common tangent while  $\overline{v_6 w_5}$  represents the lower common tangent. The  $i$ th *dpolygon* is defined as the polygon formed between the convex hull of  $S_i$  and the convex hull of  $\bigcup_{k=i+1}^{\sqrt{N}-1} S_k$ ,  $0 \leq i \leq \sqrt{N} - 2$ .

**LEMMA 8.** *The dpolygon formed by two disjoint convex hulls having  $N_1$  and  $N_2$  extreme points can be triangulated in  $O(1)$  time using an  $N_1 \times N_2$  reconfigurable mesh.*

**PROOF.** We use the technique by Wang and Tsin [49] on the CREW PRAM. Their technique results in an  $O(\log N)$  time solution while the reconfigurable bus system can be exploited in nontrivial ways to achieve  $O(1)$  time. Without loss of generality, assume  $N_1 \geq N_2$  and  $v_i$ ,  $0 \leq i < N_1$  represent the extreme points (sorted in angular order) of the left convex hull and  $w_j$ ,  $0 \leq j < N_2$  represent those of the right convex hull. Initially,  $v_i$  is assumed to be in  $PE(i, 0)$  for  $0 \leq i < N_1$  and  $w_j$  is assumed to be in  $PE(0, j)$  for  $0 \leq j < N_2$ . Let  $x(p)$  be the  $x$  coordinate of a point  $p$ . Identify  $v_r$  and  $w_l$  such that  $x(v_r) \geq x(v_j)$ ,  $0 \leq i < N_1$  and  $x(w_l) \leq x(w_j)$ ,  $0 \leq j < N_2$ . In the example shown in Fig. 11a,  $v_r = v_3$  and  $w_l = w_4$ . Since both  $x(v_j)$ ,  $0 \leq i < N_1$  and  $x(w_j)$ ,  $0 \leq j < N_2$  form bitonic sequences,  $v_r$ ,  $w_l$  can be identified in  $O(1)$  time using Lemma 2. The end points of the upper and lower tangents ( $v_0, w_0$  and  $v_6, w_5$  in Fig. 11a) can be similarly identified. We will show how to triangulate the upper half of the *dpolygon*, formed by  $v_i$ ,  $0 \leq i \leq r$  and  $w_j$ ,  $0 \leq j \leq l$ . The lower half formed by  $v_i$ ,  $r \leq i < N_1$  and  $w_j$ ,  $l \leq j < N_2$  can be triangulated in the same way.

- 1) For each  $w_j$ ,  $0 \leq j \leq l$ , find a  $v_k$  such that the angle  $\angle v_r w_j v_k$  is the maximum over  $0 \leq k \leq r - 1$ . Create a pair of points  $(v_k, w_j)$  as a *triangulating edge*.  $l + 1$  pairs of points will be created as illustrated in Fig. 11b.
- 2) For  $0 \leq j \leq l$ , create  $(v_k, w_{j-1})$  if  $(v_k, w_j)$  and  $(v_k, w_{j-1})$  were created in Step 1 and  $k \neq k'$ . This can be performed in  $O(1)$  time (See Fig. 11c).
- 3) For  $0 \leq i \leq r$ , if  $v_i$  does not belong to any pairs created in Steps 1 or 2 and  $v_i$  is located between  $v_r$  and

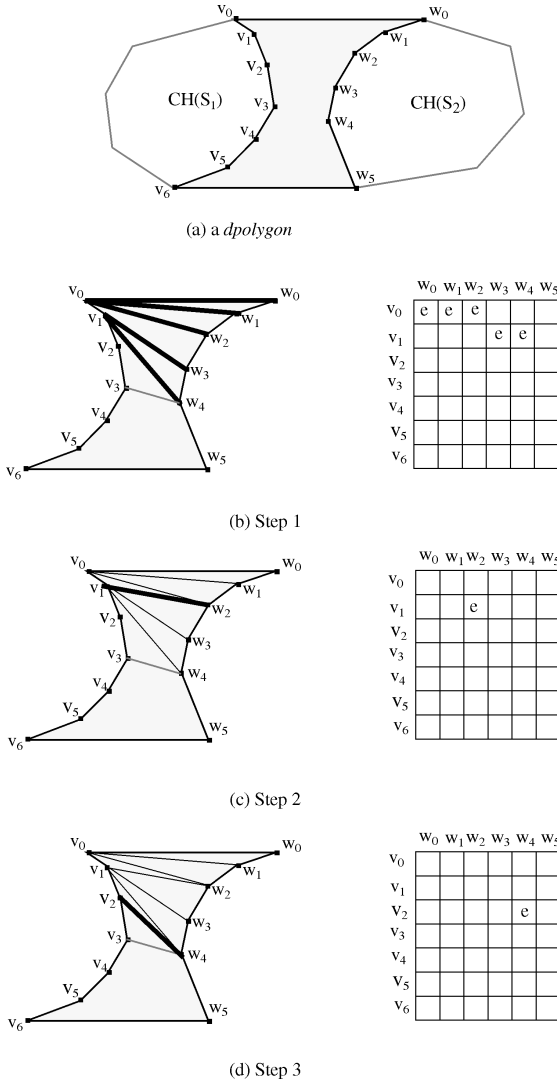


Fig. 11. Triangulation of a *dpolygon* (e: triangulating edge).

$v_{f'}$  where  $(v_f, w_j)$  and  $(v_{f'}, w_j)$  belong to the pairs created in Steps 1 or 2, then create a pair  $(v_{f'}, w_j)$  (See Fig. 11d).

Since all the pairs created are in either distinct rows or in distinct columns, they can all be collected into the leftmost column in  $O(1)$  time without any conflict on the bus.  $\square$

### 3.3.3 $O(1)$ Time Solution Using $N \times N$ Mesh

Lemmas 6, 7, 8, and Theorem 1 can be combined to result in an  $O(1)$  time algorithm to triangulate  $N$  points on an  $N \times N$  reconfigurable mesh.

**THEOREM 2.** *Given  $N$  points stored in a row of the reconfigurable mesh, these can be triangulated in  $O(1)$  time on an  $N \times N$  reconfigurable mesh.*

**PROOF.** Let  $S$  be the given set of  $N$  points. Using Lemma 6, partition  $S$  into  $S_i$ ,  $0 \leq i < \sqrt{N}$  such that  $|S_i| = \sqrt{N}$  and the  $x$  coordinates of all the points in  $S_i$  are greater than those in  $S_{i-1}$  for  $1 \leq i < \sqrt{N}$ . Let  $CH(S_i)$  and  $E(S_i)$  de-

note the convex hull and the extreme points of  $S_i$  respectively. Then, triangulating  $S$  can be decomposed into the following two steps.

- 1) Triangulate  $S_i$ ,  $0 \leq i < \sqrt{N}$ .
- 2) Triangulate the *dpolygons* formed between  $CH(S_i)$  and  $CH(\cup_{k=i+1}^{k=\sqrt{N}-1} S_k)$ , for  $0 \leq i \leq \sqrt{N} - 2$ .

From Theorem 1,  $CH(\cup_{k=i+1}^{k=\sqrt{N}-1} S_k)$ ,  $0 \leq i \leq \sqrt{N} - 2$  can be computed in  $O(1)$  time on an  $N \times N$  reconfigurable mesh. Let  $CM(i)$  denote the submesh consisting of  $PE(u, i\sqrt{N} + w)$ ,  $0 \leq u < N$ ,  $0 \leq w < \sqrt{N}$ .  $E(\cup_{k=i+1}^{k=\sqrt{N}-1} S_k)$  is stored in  $CM(i)$ . In  $O(1)$  time,  $CM(i)$  identifies the *dpolygon* formed between  $CH(S_i)$  and  $CH(\cup_{k=i+1}^{k=\sqrt{N}-1} S_k)$ ,  $0 \leq i < \sqrt{N}$ .

Using Lemma 8, each *dpolygon* can be triangulated in  $O(1)$  time by a submesh  $CM(i)$ . The triangulating edges created in each  $CM(i)$ ,  $0 \leq i \leq \sqrt{N} - 2$  by Step 3 in the proof of Lemma 8 are in distinct rows of the entire mesh. Note also that the edges created in Steps 1 and 2 in the proof of Lemma 8 are in distinct columns of the mesh. Thus, the triangulating edges from  $CM(i)$ ,  $0 \leq i < \sqrt{N}$  can be collected into the leftmost column of the mesh in  $O(1)$  time.  $\square$

### 3.4 All-Pairs Nearest Neighbor

Given a set  $S$  of  $N$  planar points, the *all-pairs nearest neighbor problem* can be defined as follows: For each  $p \in S$ , find a  $q \in S$ , such that  $q \neq p$  and  $q$  is nearest to  $p$  among all points in  $S$ . Miller and Stout [25] solve the problem in  $O(\sqrt{N})$  time using  $\log N$  levels of recursion on a 2-MCC of size  $\sqrt{N} \times \sqrt{N}$ . In [23], Mackenzie and Stout solve it using  $\log \log N$  levels of recursion on a hypercube of size  $N$  to run in  $O(\log N (\log \log N)^3)$  time. We develop an  $O(1)$  time algorithm for a reconfigurable mesh of size  $N \times N$ . In the following, any distance measure that can be computed within a PE in  $O(1)$  time given a pair of arguments can be used.

**THEOREM 3.** *Given  $N$  points stored in a row of the reconfigurable mesh, the all-pairs nearest neighbor problem can be solved in  $O(1)$  time on an  $N \times N$  reconfigurable mesh.*

**PROOF.** Let  $S$  be the input set of  $N$  planar points. Without loss of generality, assume that no two points have either the same  $x$  coordinate or the same  $y$  coordinate.

- 1) Using Lemma 6, sort the points by their  $x$  coordinates. Partition the sorted sequence of points into  $X^b$ ,  $0 \leq b < N^{1/4}$ , such that  $|X^b| = N^{3/4}$  and the  $x$  coordinates of the points in  $X^b$  is greater than the  $x$  coordinates of the points in  $X^{b-1}$ .
- 2) Solve the *all-pairs nearest neighbor problem* within each  $X^b$  on a submesh of size  $N^{3/4} \times N$ , simultaneously for  $0 \leq b < N^{1/4}$ .
- 3) Repeat Step 1 using the  $y$  coordinates as keys. Let  $Y^a$ ,  $0 \leq a < N^{1/4}$ , be the resulting subsets.
- 4) Solve the *all-pairs nearest neighbor problem* within each  $Y^a$  on a submesh of size  $N^{3/4} \times N$ , simultaneously for  $0 \leq a < N^{1/4}$ .

- 5) For each  $p \in Y^a \cap X^b$ ,  $0 \leq a, b < N^{1/4}$ , find its nearest neighbor point within  $Y^a \cup X^b$  by comparing its nearest neighbor within  $Y^a$  and that within  $X^b$ .
- 6) Route  $Y^a \cap X^b$  to the  $(a, b)$ th submesh of size  $N^{3/4} \times N^{3/4}$ ,  $0 \leq a, b < N^{1/4}$ . Note that  $|Y^a \cap X^b| \leq N^{3/4}$ .
- 7) Let  $x_b$  ( $y_a$ ) be the smallest  $x$  ( $y$ ) coordinate of the points in  $X^b$  ( $Y^a$ ). Broadcast  $x_b, x_{b+1}, y_a, y_{a+1}$  to all the points in  $Y^a \cap X^b$ . For each point, compare its current nearest neighbor with the four grid points,  $(y_a, x_b)$ ,  $(y_{a+1}, x_b)$ ,  $(y_a, x_{b+1})$ , and  $(y_{a+1}, x_{b+1})$ . Mark the point if there is any grid point which is closer than its current nearest neighbor. It is known [25], [23] that after all the points in  $Y^a \cap X^b$  have found their nearest neighbors within  $Y^a \cup X^b$ , at most eight points from  $Y^a \cap X^b$  have not yet found their nearest neighbors in  $S$ . Note that the marked points are the ones that have not yet found their nearest neighbors in  $S$ .
- 8) Route the marked points (at most eight) from  $Y^a \cap X^b$ ,  $0 \leq a, b < N^{1/4}$  to the  $(a, b)$ th submesh of size  $\sqrt{N} \times N$ ,  $0 \leq a, b < N^{1/4}$ . Call these points  $R(a, b)$ . Copy the set of all marked points to each submesh. Note that there are at most  $8\sqrt{N}$  marked points in the entire mesh.
- 9) In the  $(a, b)$ th submesh, update the current nearest neighbor of each point in  $R(a, b)$  by computing the distances between the point and the other  $N - 1$  points of  $S$ . This can be performed by recursively using Lemma 3. The minimum of the  $N - 1$  distances can be obtained in  $O(1)$  time using a submesh of size  $\sqrt{N} \times N$ . First, the  $N - 1$  distances are divided into  $\sqrt{N}$  subsets consisting of either  $\sqrt{N}$  or  $\sqrt{N} - 1$  points. The minimum of each subset can be identified in  $O(1)$  time using a block of size  $\sqrt{N} \times \sqrt{N}$ . Then the minimum of the resulting  $\sqrt{N}$  minimums can be obtained in  $O(1)$  time to find the nearest point. Since there are only eight points in  $R(a, b)$ , the points in  $R(a, b)$  can update their nearest neighbors in  $O(1)$  time, for all  $0 \leq a, b < N^{1/4}$ .

Let  $T(N)$  be the time needed to perform the above steps on a set  $S$  of size  $N$  on an  $N \times N$  reconfigurable mesh. Then, Step 2 and Step 4 each take  $T(N^{3/4})$  time. Since other steps can be performed in  $O(1)$  time, we have  $T(N) = 2T(N^{3/4}) + O(1)$ . We have partitioned the input to  $N^{1/4}$  subsets of size  $N^{3/4}$ . This results in at most  $8\sqrt{N}$  points which have not found their nearest points before the start of Step 9. If we had partitioned the input in Step 1 (and in Step 3) into  $N^{1/2}$  subsets of size  $N^{1/2}$ , there would be up to  $8N$  points which have not found their nearest neighbors at the start of Step 9.

The remaining problem is how to perform Step 2 and Step 4 in  $O(1)$  time. This problem can be solved using one level of recursion in Step 2 and Step 4. Then, we have  $T(N) = 2[2(T(N^{27/64}) + O(1)) + O(1)] + O(1)$ . Clearly, the *all-pairs nearest neighbor* problem on

$N^{27/64}$  points can be solved in  $O(1)$  time using an  $N \times N^{27/64}$  reconfigurable mesh. Alternatively the  $N^{3/4} \times N^{3/4}$  submesh may be partitioned into  $N^{3/4} \times N^{3/4} \times N^{1/4}$  submeshes and each of these finds the nearest neighbor for one point. For this, in each  $N^{3/4} \times N^{1/4}$  submesh, the distance to each of the remaining  $N^{3/4} - 1$  points is found in the leftmost column. Next, we need to find the minimum of these distances. This is done in three passes. In the first pass, each  $N^{1/4} \times N^{1/4}$  submesh finds the minimum of  $N^{1/4}$  values by comparing all pairs of distances. This leaves us with  $N^{1/2}$  candidates. In the second pass, these candidates are divided into  $N^{1/4}$  groups and submeshes of size  $N^{1/2} \times N^{1/4}$  are used to find the minimum in each group. Following this,  $N^{1/4}$  candidates remain. Their minimum is now found using the entire  $N^{3/4} \times N^{1/4}$  submesh. Each pass takes  $O(1)$  time. Thus,  $T(N) = O(1)$ .  $\square$

### 3.5 Two-Set Dominance Counting

Given two points  $p$  and  $q$  in  $d$ -dimensional space,  $p$  is said to *dominate*  $q$  if each coordinate of  $p$  is larger than the corresponding coordinate of  $q$ . Given a set  $S$  of  $m$  points and a set  $T$  of  $k$  points, the *two-set dominance counting problem* between  $S$  and  $T$  is, for each point  $p \in S$ , compute the number of points in  $T$  dominated by  $p$ , and also for each point  $q \in T$ , find the number of points in  $S$  dominated by  $q$ . Let  $N = m + k$ . In the following, we develop a constant time algorithm for  $d = 2$ .

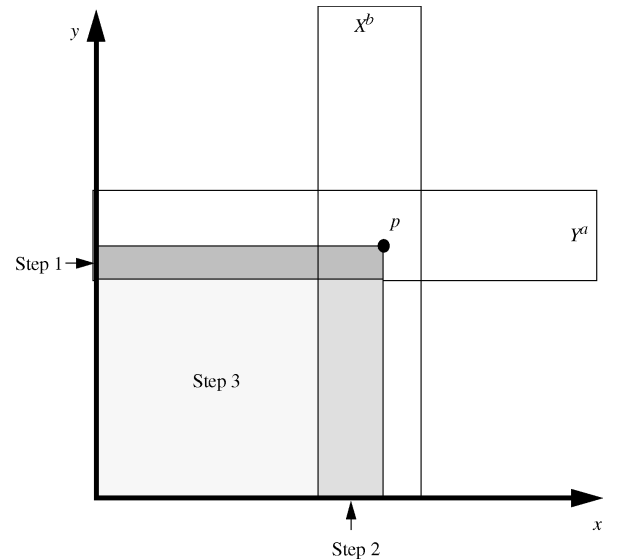


Fig. 12. Computing the number of points dominated by a point  $p$ .

**THEOREM 4.** *Given  $N$  points stored in a row of the reconfigurable mesh, the two-set dominance counting problem can be solved in  $O(1)$  time on an  $N \times N$  reconfigurable mesh.*

**PROOF.** We will show, for each point  $p \in S$ , how to find the number of points in  $T$  dominated by  $p$ . Partition  $S \cup T$  into  $\sqrt{N}$  subsets,  $X^b$ , such that the  $x$  coordinates of

- 1: Broadcast the  $N$  points on row 0 to all rows using column buses.
- 2: [Use the  $i$ th row to determine if the  $i$ th point of  $S$  is a non-dominated point]
  - PE( $i, i$ ) broadcasts its point to all processors in its row using a row bus,  $0 \leq i < N$
  - PE( $i, j$ ),  $0 \leq i, j < N$  compares the coordinates of the point it received in Step 1 to those of the points it received in Step 2. If each coordinate of the Step 1 point is larger than the corresponding coordinate of its Step 2 point, then PE( $i, j$ ) sets its  $T$  variable to 0. Otherwise,  $T$  is set to 1.
  - Using row bus splitting, PE( $i, 1$ ) determines if there is a 0 in row  $i$ ,  $0 \leq i < N$ . If so, it sets its  $U$  variable to 0 (the  $i$ th point is dominated). Otherwise,  $U$  is set to 1 (the  $i$ th point is not dominated).
- 3: [Route the results back to row 0]
  - Using row buses PE( $i, 0$ ) sends its  $U$  value to PE( $i, i$ ). Using column buses, PE( $i, i$ ) sends the  $U$  value just received to PE( $0, i$ ),  $0 \leq i < N$ .

Fig. 13. Simple constant time algorithm for three-dimensional maxima.

all the points in  $X^b$  are greater than those in  $X^{b-1}$ ,  $0 \leq b < \sqrt{N}$ . Similarly, create  $Y^a$ ,  $0 \leq a < \sqrt{N}$  using the  $y$  coordinates as keys. For each point  $p \in S \cap Y^a \cap X^b$ ,  $0 \leq a, b < \sqrt{N}$ , the number of points in  $T$  dominated by  $p$  can be computed by the following four steps (See Fig. 12):

- 1) Find the number of points which belong to  $T \cap Y^a$ ,  $0 \leq a < \sqrt{N}$  and are dominated by  $p$ , simultaneously for all  $p \in S \cap Y^a \cap X^b$ .
- 2) Find the number of points which belong to  $(\bigcup_{0 \leq i < a} Y^i) \cap X^b \cap T$  and are dominated by  $p$ , simultaneously for all  $p \in S \cap Y^a \cap X^b$ .
- 3) Compute  $\sum_{i=0}^{a-1} \sum_{j=0}^{b-1} |Y^i \cap X^j \cap T|$ .
- 4) Sum up the above three numbers.

We will show how to implement Step 1 in  $O(1)$  time in parallel for  $0 \leq a, b < \sqrt{N}$ . In the  $a$ th  $N \times \sqrt{N}$  submesh, the two-set dominance counting problem for  $S \cap Y^a$  and  $T \cap Y^a$  is solved. Note that since  $|S \cap Y^a| \leq \sqrt{N}$  and  $|T \cap Y^a| \leq \sqrt{N}$ , each  $p \in S \cap Y^a$  can use a  $\sqrt{N} \times \sqrt{N}$  block of the  $a$ th  $N \times \sqrt{N}$  submesh for this purpose. The number of points in  $T \cap Y^a$  dominated by  $p$  is now trivially obtained. Since each  $p$  knows which  $X^b$  it belongs to, it can easily determine whether or not the count just obtained contributes to Step 1.

To implement Step 3,  $|Y^a \cap X^b \cap T|$  is computed in a submesh of size  $\sqrt{N} \times \sqrt{N}$ ,  $0 \leq a, b < \sqrt{N}$ . Note that,  $|Y^a \cap X^b \cap T| \leq \sqrt{N}$ . Prefix summation of  $|Y^a \cap X^b \cap T|$  over  $0 \leq b < \sqrt{N}$  is performed in the  $a$ th submesh of size  $N \times \sqrt{N}$ . This is performed by prefix modular computation using  $k = \sqrt{N} + 1$ . Using Lemma 4, this can be completed in  $O(1)$  time. Each  $|Y^a \cap X^b \cap T|$  is represented as a 0/1 vector of length

$\sqrt{N}$  having  $|Y^a \cap X^b \cap T|$  1s. Another prefix summation on the prefix sums over  $0 \leq a < \sqrt{N}$  can be performed using Lemma 5. As a result, we have

$$\sum_{i=0}^{a-1} \sum_{j=0}^{b-1} |Y^i \cap X^j \cap T|$$

for all  $a, b$ ,  $0 \leq a, b < \sqrt{N}$ . Step 4 computes  $N$  sums of triples computed in Steps 1, 2, and 3. This can be completed in  $O(1)$  time. Thus, all the steps can be performed in  $O(1)$  time.  $\square$

### 3.6 Three-Dimensional Maxima

Given a set  $S$  of  $N$  points in three-dimensional space, the *three-dimensional maxima problem* is to find all points  $p \in S$  such that no other point in  $S$  dominates  $p$ .

**THEOREM 5.** *Given  $N$  points in three dimensions stored in a row of the mesh, the three-dimensional maxima problem can be solved in  $O(1)$  time on an  $N \times N$  reconfigurable mesh.*

**PROOF.** The problem is solved in constant time using the algorithm of Fig. 13. The input points are in row 0 of the mesh.  $\square$

## 4 CONCLUSION

We have shown  $O(1)$  time solutions for the convex hull, triangulation, all-pairs nearest neighbors, the smallest enclosing rectangle, two-set dominance counting, and three-dimensional maxima problems on the reconfigurable mesh. All the solutions accept  $N$  points as input and solve the problem using  $N \times N$  reconfigurable mesh. These solutions either improve on known results or present parallel solutions to problems never examined on the reconfigurable mesh model. One major contribution of the paper is in optimizing the divide-and-conquer parameters such as the number of levels and the breadth of each level of the recursion to obtain constant time solutions using  $N \times N$  reconfigurable mesh. For this, we have developed nontrivial techniques to exploit the reconfiguration feature.

## ACKNOWLEDGMENTS

This research was supported in part by the U.S. National Science Foundation under grants IRI-9217528 and MIP-9103379 and by DARPA and AFOSR contracts F-49260-89-C-0126 and F-49620-90-C-0078 and by KOSEF under contract 961-0909-052-1.

## REFERENCES

- [1] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," *J. Parallel and Distributed Computing*, vol. 13, no. 2, pp. 139-153, 1991.
- [2] Y. Ben-Asher and A. Schuster, "Optical Splitting Graphs," *Proc. Int'l Topical Meeting on Optical Computing*, Kobe, Japan, 1990.
- [3] Y. Ben-Asher, D. Gordon, and A. Schuster, "Optimal Simulations in Reconfigurable Arrays," Technical Report #716, Computer Science Dept., Technion—Israel Inst. of Technology, Feb. 1992.
- [4] V. Bokka, H. Gurla, S. Olariu, and J.L. Schwing, "Constant-Time Convexity Problems on Reconfigurable Meshes," *J. Parallel and Distributed Computing*, vol. 27, no. 1, pp. 86-99, May 1995.

- [5] V. Bokka, H. Gurla, S. Olariu, and J.L. Schwing, "Constant-Time Triangulation Problems on Reconfigurable Meshes," *Proc. Application Specific Array Processor*, pp. 357-368, San Francisco, Aug. 1994.
- [6] R. Cole and U. Vishkin, "Approximate Coin Tossing with Applications to List, Tree and Graph problems," *Proc. IEEE Symp. Foundations of Computer Science*, pp. 478-491, 1986.
- [7] H. Elgindy and P. Wegrowicz, "Selection on the Reconfigurable Mesh," *Proc. Int'l Conf. Parallel Processing*, pp. III.26-III.33, Aug. 1991.
- [8] H. Freeman and R. Shapira, "Determining the Minimal-Area Encasing Rectangle for an Arbitrary Closed Curve," *Comm. ACM*, vol. 18, pp. 409-413, 1975.
- [9] E. Hao, P.D. MacKenzie, and Q.F. Stout, "Selection on the Reconfigurable Mesh," *Proc. Frontiers of Massively Parallel Computation*, pp. 38-45, Oct. 1992.
- [10] J. Jang and V.K. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Mesh," *J. Parallel and Distributed Computing*, vol. 25, pp. 31-41, Feb., 1995. Also appears as Technical Report IRIS #277, Dept. of EE-Systems, Univ. of Southern California, Aug. 1991.
- [11] J. Jang and V.K. Prasanna, "Efficient Parallel Algorithms for Some Geometric Problems on Reconfigurable Mesh," *Proc. Int'l Conf. Parallel Processing*, pp. III.127-III.130, Aug. 1992.
- [12] J. Jang, H. Park, and V.K. Prasanna, "A Fast Algorithm for Computing a Histogram on Reconfigurable Mesh," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 97-106, Feb. 1995.
- [13] J. Jang, H. Park, and V.K. Prasanna, "An Optimal Multiplication Algorithm on Reconfigurable Mesh," *Proc. Symp. Parallel and Distributed Processing*, pp. 384-391, Dec. 1992.
- [14] J. Jang, V.K. Prasanna, and H. Park, "A Bit Model of Reconfigurable Mesh," *Proc. Reconfigurable Architectures Workshop, IPPS '94*, Cancun, Mexico, Apr. 1994.
- [15] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for the Area and Perimeter of Image Components," *Proc. Int'l Conf. Parallel Processing*, pp. 280-281, Aug. 1991.
- [16] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching," *Proc. Int'l Parallel Processing Symp.*, pp. 208-215, Apr. 1991.
- [17] J. Jenq and S. Sahni, "Histogramming on a Reconfigurable Mesh Computer," *Proc. Int'l Parallel Processing Symp.*, pp. 425-432, Mar. 1992.
- [18] J. Levinson, I. Kuroda, and T. Nishitani, "A Reconfigurable Processor Array with Routing LSIs and General Purpose DSPs," *Proc. Int'l Conf. Application Specific Array Processors*, Oct. 1992.
- [19] H. Li and M. Maresca, "Polymorphic-Torus Network," *IEEE Trans. Computers*, vol. 38, no. 9, pp. 1,345-1,351, Sept. 1989.
- [20] R. Lin, "Shift Switching and Novel Arithmetic Schemes," *Proc. 29th Asilomar Conf. Signals, Systems, and Computers*, pp. 580-585, Pacific Grove, Calif., Nov. 1995.
- [21] R. Lin and S. Olariu, "Reconfigurable Buses with Shift Switching: Concept and Applications," *IEEE Trans. Parallel And Distributed Systems*, vol. 6, no 1, pp. 93-102, Jan. 1995.
- [22] R. Lin, S. Olariu, J. Schwing, and J. Zhang, "A VLSI-Optimal Constant Time Sorting on Reconfigurable Mesh," *Proc. Ninth European Workshop Parallel Computing*, pp. 1-16, Spain, 1992.
- [23] P.D. Mackenzie and Q.F. Stout, "Asymptotically Efficient Hypercube Algorithms for Computational Geometry," *Proc. Frontiers of Massively Parallel Computation*, pp. 8-11, Oct. 1990.
- [24] R. Miller, V.K. Prasanna Kumar, D.I. Reisis, and Q.F. Stout, "Meshes with Reconfigurable Buses," *Proc. MIT Conf. Advanced Research in VLSI*, pp. 163-178, Apr. 1988.
- [25] R. Miller and Q.F. Stout, "Mesh Computer Algorithms for Computational Geometry," *IEEE Trans. Computers*, vol. 38, no. 3, pp. 321-340, Mar. 1989.
- [26] R. Miller, V.K. Prasanna Kumar, D. Reisis, and Q.F. Stout, "Image Computations on Reconfigurable Mesh," *Proc. IEEE Conf. Computer Vision and Pattern Recognition(CVPR)*, pp. 925-930, 1988.
- [27] R. Miller, V.K. Prasanna Kumar, D. Reisis, and Q.F. Stout, "Parallel Computations on Reconfigurable Meshes," *IEEE Trans. Computers*, vol. 42, no. 6, pp. 678-692, June 1993.
- [28] K. Nakano, T. Msuzawa, and N. Tokura, "A Sub-Logarithmic Time Sorting Algorithm on a Reconfigurable Array," *Inst. of Electronics, Information, and Communication Engineers*, vol. E-74, no. 11, pp. 3,894-3,901, Nov. 1991.
- [29] M. Nigam and S. Sahni, "Sorting  $n$  Numbers on  $n \times n$  Reconfigurable Meshes with Buses," *Proc. Int'l Parallel Processing Symp.*, pp. 174-181, Apr. 1993.
- [30] M. Nigam and S. Sahni, "Computational Geometry on a Reconfigurable Mesh," *Proc. Int'l Parallel Processing Symp.*, pp. 86-93, 1994.
- [31] M. Nigam and S. Sahni, "Triangulation on a Reconfigurable Mesh with Buses," *Proc. Int'l Conf. Parallel Processing*, pp. 251-257, 1994.
- [32] S. Olariu, J.L. Schwing, and J. Zhang, "Fast Computer Vision Algorithms for Reconfigurable Meshes," *Image and Vision Computing*, pp. 610-616, 1992.
- [33] S. Olariu, J.L. Schwing, and J. Zhang, "Time-Optimal Convex Hull Algorithms on Enhanced Meshes," *BIT*, vol. 33, pp. 396-410, 1993.
- [34] H. Park, V.K. Prasanna, and J. Jang, "Fast Arithmetic on Reconfigurable Meshes," *Proc. Int'l Conf. Parallel Processing*, Aug. 1993.
- [35] V.K. Prasanna Kumar and C.S. Raghavendra, "Array Processor with Multiple Broadcasting," *J. Parallel and Distributed Computing*, vol. 4, pp. 173-190, 1987.
- [36] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*. New York: Springer-Verlag, 1985.
- [37] D.I. Reisis, "Parallel Computations on Meshes with Static and Reconfigurable Buses," PhD Thesis, Dept. of EE-Systems, Univ. of Southern California, May 1989.
- [38] D.I. Reisis, "An Efficient Convex Hull Computation on the Reconfigurable Mesh," *Proc. Int'l Parallel Processing Symp.*, pp. 142-145, Mar. 1992.
- [39] J. Rothstein, "Bus Automata, Brains, and Mental Models," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 18, no. 4, pp. 522-531, Apr. 1988.
- [40] D.B. Shu, L.W. Chow, J.G. Nash, and C.C. Weems, "A Content Addressable Array Parallel Processor," *Proc. IEEE Workshop VLSI Signal Processing III*, R.W. Brodersen and H.S. Moscovitz, eds., pp. 120-128. New York: IEEE CS Press, 1988.
- [41] L. Snyder, "Introduction to the Configurable Highly Parallel Computer," *Computer*, vol. 15, no. 1, pp. 47-56, Jan. 1982.
- [42] Q.F. Stout, "Meshes with Multiple Buses," *Proc. IEEE Conf. Foundations of Computer Science*, pp. 264-272, Oct. 1986.
- [43] C. Subbaraman, J.L. Trahan, and R. Vaidyanathan, "List Ranking and Graph Algorithms on the Reconfigurable Multiple Bus Machine," *Proc. Int'l Conf. Parallel Processing*, vol. III, pp. 244-247, St. Charles, Ill., Aug. 1993.
- [44] R.K. Thiruchelvan, J.L. Trahan, and R. Vaidyanathan, "On the Power of Segmenting and Fusing Buses," *Proc. Seventh Int'l Parallel Processing Symp.*, pp. 79-83, Newport Beach, Calif., Apr. 1993.
- [45] J.L. Trahan, R. Vaidyanathan, and C.P. Subbaraman, "Constant Time Graph and Poset Algorithms on the Reconfigurable Multiple Bus Machine," *Proc. Int'l Conf. Parallel Processing*, vol. III, pp. 214-217, St. Charles, Ill., Aug. 1994.
- [46] B.F. Wang, G.H. Chen, and F.C. Lin, "Constant Time Sorting on a Processor Array with a Reconfigurable Bus Systems," *Information Processing Letters*, pp. 187-192, 1990.
- [47] B.F. Wang and G.H. Chen, "Constant Time Algorithms for the Transitive Closure Problem and Some Related Graph Problems on Processor Arrays with Reconfigurable Bus Systems," *IEEE Trans. Parallel and Distributed Systems*, pp. 500-507, 1991.
- [48] B.F. Wang, G.H. Chen, and H. Li, "Configurational Computation: A New Computation Method on Processor Arrays with Reconfigurable Bus Systems," *Proc. Int'l Conf. Parallel Processing*, pp. III. 42-49, Aug. 1991.
- [49] C.A. Wang and Y.H. Tsin, "An  $O(\log n)$  Time Parallel Algorithm for Triangulating a Set of Points in the Plane," *Information Processing Letters*, vol. 25, pp. 55-60, Apr. 1987.
- [50] C.C. Weems and J.H. Burrill, "The Image Understanding Architecture and Its Programming Environment," *Parallel Architectures and Algorithms for Image Understanding*, V.K. Prasanna Kumar, ed. Academic Press, 1991.



**Ju-wook Jang** received the BS degree in electronic engineering from Seoul National University in 1983, the MS degree in electrical engineering from the Korea Advanced Institute of Science and Technology in 1985, and the PhD degree in electrical engineering from the University of Southern California in 1993. Since 1995, he has been a member of the faculty of the Department of Electronic Engineering, Sogang University. From 1985 to 1988 and from 1993 to 1995, he worked with Samsung Electronics in the fields of communication and computer development. His research interests include parallel architectures, parallel algorithms, and multimedia.

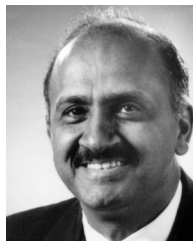
**Madhusudan Nigam** received a BTech in mechanical engineering from the Indian Institute of Technology, Kanpur, India. He has a master's degree in engineering, computer science, and business administration from the University of Minnesota and a PhD in computer science from the University of Florida. He has worked in industry in the area of software design and development, A.I., and neural network applications. His current research interests include parallel and distributed processing, networks, image processing, computational geometry, and A.I. and neural network applications. He is also interested in financial markets and investment management.



**Viktor K. Prasanna** (V.K. Prasanna Kumar) received his BS in electronics engineering from Bangalore University and his MS from the School of Automation, Indian Institute of Science. He obtained his PhD in computer science from the Pennsylvania State University in 1983. He is currently a professor in the Department of Electrical Engineering-Systems, University of Southern California, Los Angeles, and serves as the director of the Computer Engineering Division. His research interests include parallel

computation, computer architecture, VLSI computations, and high performance computing for signal and image processing, and vision.

Dr. Prasanna has published extensively and consulted for industries in the areas of his research interests. He served on the organizing committees of several international meetings in VLSI computations, parallel computation, and high performance computing. He is the general co-chair of the IEEE International Parallel Processing Symposium, 1997, and has been the key person in developing the meeting into a premier international meeting in parallel computing. He also serves on the editorial boards of the *Journal of Parallel and Distributed Computing* and *IEEE Transactions on Computers*. He was the founding chair of the IEEE Computer Society Technical Committee on Parallel Processing. He is a fellow of the IEEE.



**Sartaj Sahni** received his BTech (electrical engineering) degree from the Indian Institute of Technology, Kanpur, and the MS and PhD degrees in computer science from Cornell University. He is a professor in the Department of Computer and Information Sciences and Engineering at the University of Florida.

Dr. Sahni's research publications are on the design and analysis of efficient algorithms, parallel computing, interconnection networks, and design automation, in which areas he has published more than 150 research papers and authored or coauthored several texts including: *Fundamentals of Data Structures* (coauthor), *Fundamentals of Data Structures in Pascal* (coauthor), *Fundamentals of Data Structures in C* (coauthor), *Fundamentals of Data Structures in C++* (coauthor), *Fundamentals of Computer Algorithms* (coauthor), *Hypercube Algorithms: With Applications to Image Processing and Pattern Recognition* (coauthor), *Concepts in Discrete Mathematics* (author), and *Software Development in Pascal* (author). He also edited the proceedings of the 1987 International Conference on Parallel Processing.

Dr. Sahni is a co-editor of the *Journal of Parallel and Distributed Computing* and he is a member of the editorial boards of *IEEE Parallel and Distributed Technology* and *Computer Systems: Science and Engineering*. He has served as program committee chair and general chair of and has been a keynote speaker for many conferences. He is a fellow of the IEEE, ACM, AAAS, and Minnesota Supercomputer Institute.