

COMPUTATIONAL GEOMETRY ON A HYPERCUBE

Ivan Stojmenović

Department of Mathematics and Computer Science  
University of Miami, P.O.Box 249085  
Coral Gables, FL 33124, USA

**Abstract.** This research focuses on implementing algorithms to solve basic geometric problems on hypercube computers which are recently introduced on the commercial market. Two solutions for the planar convex hull problem are presented, both in  $O(\log^2 n)$  time which is the best one can expect with existing sorting algorithms. An  $O(\log^3 n)$  Voronoi diagram algorithm is given.  $O(\log n)$  solutions for detecting and finding intersection of two convex polygons, computing minimal distance between two convex polygons and finding critical support lines of two convex polygons and  $O(\log^2 n)$  solutions to the diameter, smallest enclosing box, width, minimax linear fit, vector sum of two convex polygons, ECDF searching, 2-D and 3-D maximal elements, 2-set dominance counting and closest points problems are described. Several data communications techniques used to solve geometric problems are also presented.

Introduction

A  $d$ -dimensional hypercube computer consists of  $n = 2^d$  synchronized processing elements (or nodes), linked together in a  $d$ -dimensional binary cube network. Each node has associated a constant size memory. Each node  $m$  is given a unique  $d$ -bit identification number  $(m_{d-1}, \dots, m_1, m_0)$  (henceforth referred as the node i.d.). In the lexicographic order of nodes, node and its i.d. number are related by  $m = 2^{d-1}m_{d-1} + \dots + 2m_1 + m_0$ . Two nodes in a hypercube are said to be neighboring if they share a communication link, i.e. iff their corresponding i.d.'s differ in exactly one bit position. The notation  $\otimes_k(m)$  will be used to denote the node  $m$  with  $k$ -th bit flipped (for example,  $\otimes_3(01001) = 00001$ ). The neighbors of a node  $m$  are exactly  $\otimes_0(m), \otimes_1(m), \dots, \otimes_{d-1}(m)$ . The communication diameter of hypercube networks is logarithmic.

We use a model of hypercubes in which communication time is assumed to predominate. We ignore the time for start-up and termination and the transfer rate when sending messages. Thus, we assume that, in unit time, each processor may send at most one message to one of its neighbors or perform at most one operation (processor-bound model). Using the model described, we solve some geometric problems, assuming that we are given one element (point, edge, ...) per processor (we suppose that input/output procedures are done in constant time via a processor with large memory connected with all other processors). The next section describes data communication techniques used in our solutions.

Data communication on hypercubes

**Broadcasting.** One node has to send the same message to all the other nodes in the hypercube. A  $O(\log n)$  solution is presented in [20].

**Parallel prefix.** Given an array  $b_0, \dots, b_{n-1}$ , one element per processor, compute  $b_0 * b_1 * \dots * b_i$  for  $1 \leq i \leq n-1$ , where '\*' is arbitrary binary associative operation. We implement the standard parallel prefix algorithm (cf. [7]) on a hypercube, to run in  $O(\log n)$  time. Each node  $m$  of hypercube stores three data:  $t, r$  and  $c$ , where  $t$  is initially equal to  $b_m$  and finally to  $b_0 * b_1 * \dots * b_m$ . We use ' $a \leftarrow b$ ' ( $b \rightarrow a$ )

to denote sending data  $b$  from  $\otimes_i(m)$  ( $m$ , resp.) to node  $m$  ( $\otimes_i(m)$ , resp.) which receives it and stores as data  $a$ . ' $:=$ ' is used for assignments made in node  $\otimes_i(m)$ . The algorithm runs for each node  $x$  in parallel.

```
FOR i = 0 TO d - 1 DO IF x + 1 = 0(mod 2i+1) THEN
  BEGIN r ← t; t := r * t END;
FOR i = d - 2 DOWNTWO 0 DO
  IF x + 1 = 0(mod 2i+1) and x ≠ 2i+1 - 1 THEN
    BEGIN c ← t; r → t; t := r * t; r := r * c END
```

**Maximum.** For '\*' being max the first step of our algorithm will report (in node  $n-1$ ) the maximal element.

**Ranking.** Some nodes are selected. The rank of a node is the number of selected nodes with a smaller index. A  $O(\log n)$  ranking algorithm has been presented in [15]. Our parallel prefix algorithm solves also the ranking problem (for '\*' being '+' and  $b_m$  being 0 or 1) with less number of data movement operations than in [15].

**Sorting.** Given an element per processor, the sorting can be done in  $O(\log^2 n)$  time [4,22]. After sorting the elements are kept in nodes in the lexicographic order.

**Merging.** Given two sorted arrays  $A$  and  $B$  each stored in a hypercube of size  $n/2$ , their merging can be done in  $O(\log n)$  time [22]. We present an iterative and simple code of merging procedure from [22].

```
FOR i = 0 TO d - 2 DO IF x < n/2 THEN x → \otimes_i(x)
FOR i = d - 1 DOWNTWO 0 DO
  IF x_i = 0 THEN order(x, \otimes_i(x))
```

If the data in  $x$  is less than the data in  $\otimes_i(x)$  then  $x$  and  $\otimes_i(x)$  will exchange data as the effect of function  $order(x, \otimes_i(x))$ . The symbol ' $\rightarrow$ ' is used for passing data from one node to the other.

The cousins of  $a \in A$  in  $B$  are two consecutive elements in  $B$  so that  $a$  is between them in sorted list  $A \cup B$ . The cousins in  $B$  of each element in  $A$  can be determined in  $O(\log n)$  time on a hypercube by merging and interval broadcasting operations. The ranks of two cousins  $b_1$  and  $b_2$  from  $B$  for an element  $a \in A$  are determined by  $r(b_1, B) = r(a, A \cup B) - r(a, A)$  and  $r(b_2, B) = r(b_1, B) - 1$ , where  $r(e, X)$  denote the rank of an element  $e$  in the sorted set  $X$  (the rank of the first element being 0).

**Reversing.** The effect of the first step in the merging procedure is to reverse data in nodes  $0, \dots, n/2 - 1$ . It means that a list of data can be reversed in  $O(\log n)$  time.

**Distribution.** We assume that some nodes  $m$  of the hypercube store a record  $r_m$  and a node destination address  $h_m$  such that if  $i < j$  then  $h_i < h_j$ . The distribution operation consists of routing, for each  $m$  the record  $r_m$  to the node  $h_m$ . It can be performed in  $O(\log n)$  time [15].

**Translation.** Node  $x$  has to send a message to the node  $x + s \pmod n$  concurrently for several nodes  $x$ . This can be done in  $O(\log n)$  time by two distributions (ones for nodes with  $x + s < n$  and ones for the remaining nodes). For  $s = 1$  it gives an access for each node to the data in its successor in the lexicographic order.

**Compression.** Some nodes of hypercube contain "active" elements while others do not. Compress the active elements, i.e. store them in nodes  $0, 1, 2, \dots, s-1$  where  $s$  is

the number of active elements. After ranking active elements compression became inverse distribution operation and a solution is presented in [15].

**Unmerging.** Given a sorted list of elements so that half of elements belong to a set  $A$  (thus the remaining belong to  $\bar{A}$ , the complement of  $A$ ) and each element knows the corresponding rank in  $A$  or  $\bar{A}$ , permute the list to return each  $A$  and  $\bar{A}$  to a hypercube of size  $n/2$ . The problem can be solved by running the merging algorithm in reverse order, or by two compressions and a translation.

**Interval broadcasting.** Certain of nodes  $0, 1, \dots, n-1$  are leaders; they possess data that they must share with all the higher numbered nodes, up to but not including the next leader (the interval of nodes between two leaders). Interval broadcasting can be done in  $O(\log n)$  time on a hypercube ([22, Theorem 6.9], and [15, Theorem 1]).

**Many-one routing.** Both origin and destination nodes have keys, with keys of origin nodes being different between each other. Each destination node should receive data from the origin with the same key. The problem can be solved in  $O(\log^2 n)$  time on a hypercube [22], by applying sorting and interval broadcasting techniques.

**Pairing elements.** Given two sets  $A$  and  $B$  each containing  $\sqrt{n}$  data distributed one per node of a hypercube of size  $n$ , broadcast these data in such a way that each node of the hypercube contains exactly one pair of data (taken one from each  $A$  and  $B$ ) and all pairs are distributed. First we compress data from  $A$ . These data will be stored in a sub-hypercube  $A'$  having nodes  $(0, \dots, 0, x_{d/2-1}, \dots, x_0)$ . Also we compress data from  $B$  and translate them to the sub-hypercube  $B'$  having nodes  $(x_{d-1}, \dots, x_{d/2}, 0, \dots, 0)$ . Now, we broadcast data from each node of  $A'$  and  $B'$  to all nodes of a hypercube of size  $\sqrt{n}$ . As a result, each node  $(x_{d-1}, \dots, x_0)$  of hypercube receives a pair of data by broadcasting from nodes  $(0, \dots, 0, x_{d/2-1}, \dots, x_0)$  and  $(x_{d-1}, \dots, x_{d/2}, 0, \dots, 0)$ .

#### Planar convex hull algorithms

We present two  $O(\log^2 n)$  solutions for planar convex hull problem on a hypercube model of computation. One uses merging slopes technique (independently used in [10, 14, 18] for solving several problems on mesh computers and in [18] for solving all problems mentioned in the section on CREW PRAM; the corresponding sequential technique is presented in [21, 5]) while the other is based on CREW PRAM algorithm of [2].

Divide-and-conquer is a common strategy to find the convex hull  $H(S)$  of a set of points  $S$  sorted by  $x$ -coordinate: Partition the points of  $S$  into two separated sets  $P$  and  $Q$  of half the size, each stored in a hypercube of size  $n/2$ , recursively compute  $H(P)$  and  $H(Q)$  and merge  $H(P)$  and  $H(Q)$  to form  $H(S)$  by computing common tangents of  $H(P)$  and  $H(Q)$ . Two proposed solutions differ in the way to merge  $H(P)$  and  $H(Q)$ .

**Merging slopes technique.**  $\alpha$ -distance of a point to an oriented edge  $p$  is its distance to the an edge  $p'$  obtained by rotating  $p$  for the angle  $\alpha$  (with distances of points to the left (right) of  $p'$  being positive (negative, resp.)).

Let  $A$  and  $B$  be two convex polygons in the plane, each containing  $O(n)$  edges given in counterclockwise order. Given an angle  $\alpha$ , consider the following problem (we call it the extremal search problem  $ES(A, B, \alpha)$ ): For each edge  $p \in A$  find a vertex  $P \in B$  with the smallest  $\alpha$ -distance to  $p$  among vertices from  $B$  ( $P$  is associated point of  $p$  in direction  $\alpha$ ). It is easy to see that for  $\alpha=0$  ( $\alpha=\pi$ )

$P$  is the vertex with the smallest (greatest, resp.) distance from  $p$  among vertices of  $B$ . For  $\alpha = \pi/2$  ( $\alpha = 3\pi/2$ )  $P$  is the easternmost (westernmost, resp.) point of  $p$ .

To describe the procedure  $ES(A, B, \alpha)$ , we first increase slopes of edges of  $A$  by  $\alpha$ . The edges with minimal slopes in  $A$  and  $B$  are recognized and by some translations they are moved to first nodes of corresponding hypercubes. Since slopes of edges of both polygons are then given in increasing order, the sets  $A$  and  $B$  can be merged (by their slopes) in  $O(\log n)$  time. Now sets  $A$ ,  $B$  and  $A \cup B$  are sorted and each edge  $e$  of  $A$  can find its cousins in  $B$ , the common elements of which is associated point of  $e$ . We use unmerge technique to return all edges to initial positions.

In order to merge  $H(P)$  and  $H(Q)$ , we decide for each their edge whether it is an external or internal edge, i.e. if it is convex hull edge of  $H(S)$  as well. To judge if an edge is external, we need to test if  $H(P)$  and  $H(Q)$  are in the same half-plane bounded by the edge. However, instead of testing all the vertices of  $H(Q)$  with an edge  $e$  of  $H(P)$ , we only test two representatives (associated points of  $e$ ) such that if they are in the same half-plane bounded by  $e$  as  $H(P)$ , every point in  $H(Q)$  is.

These two representatives for  $e$  in  $H(P)$  ( $H(Q)$ ) are nearest and furthest extreme points from  $H(Q)$  ( $H(P)$ , resp.) and are obtained by calling procedures  $ES(H(P), H(Q), 0)$ ,  $ES(H(P), H(Q), \pi)$ ,  $ES(H(Q), H(P), 0)$  and  $ES(H(Q), H(P), \pi)$ . Now each edge can decide in constant time if it is external or not. Then each extreme point of  $H(P)$  or  $H(Q)$  can learn if it is an extreme point of  $H(S)$  (translation by 1 can be used to find the necessary data). Two of them in both  $H(P)$  and  $H(Q)$  share an external and an internal edge. These four points determine two common tangents of  $H(P)$  and  $H(Q)$ . Then the computation of the circular edge list of  $H(S)$  can be done in  $O(\log n)$  time by some translations.

The time complexity of all procedures in merge step is  $O(\log n)$ . Because of  $O(\log n)$  recursive calls, the overall time complexity of presented algorithm is  $O(\log^2 n)$ .

Using the merging slopes technique the diameter, smallest enclosing box, width and minimax linear fit of a set of  $n$  points and vector sum and critical support lines of two convex polygons (see [21, 5, 16, 18, 19] for definitions) can be found on a hypercube. The details of these solutions are presented in a full version of the article [19].

**Another convex hull algorithm** on a hypercube can be derived by using Atallah and Goodrich [2] CREW PRAM solution. The main point in the algorithm [2] is to divide  $P$  and  $Q$  into  $\sqrt{n}$  equal portions by considering  $\sqrt{n}$  vertices and, by examining each pair of considered vertices, to find common tangent of polygons of size  $\sqrt{n}$  obtained in this way. Then one of polygons  $P$  or  $Q$  can be reduced to size  $\sqrt{n}$  and, in one more iteration, the common tangent of  $P$  and  $Q$  will be constructed [2]. The pairing elements and the broadcasting (to construct the tangent passing through a point) techniques are applied. A similar approach was used in [6] to solve the convex hull problem in  $O(\log^2 n)$  time on CREW PRAM and cybe connected cycle models. The later one is directly implementable on hypercube in  $O(\log^2 n)$  time.

The problem of computing minimal distance between two convex polygons involves similar techniques, and is solved in [2] for CREW PRAM model of computation. Using pairing elements and other techniques a  $O(\log n)$  hypercube solution can be obtained.

#### Planar point location and Voronoi diagram

In order to locate  $O(n)$  points into the planar subdivision defined by  $O(n)$  edges we use the chain method described by Lee and Preparata [11], a parallelization of which for mesh-connected computers is given in [12]. We slightly modify both methods in order to get an  $O(\log^2 n)$  planar point location algorithm on a hypercube.

First we sort regions by  $z$ -coordinate of selected interior points (called centers). Then a monotone complete set of chains is defined as in [11,16,12]. These chains are nodes of a balanced binary tree the leaves of which correspond to regions of subdivision. Each chain has its level and index (the rank of the chain in the chains of given level). Chains may share common edges. If an edge  $e$  belongs to more than one chain it belongs to all members of a set (an interval) of consecutive chains. We assign  $e$  to hierarchically the highest chain to which  $e$  belongs. The level and index of the chain is determined in constant time by the rule described in [12]. Now we sort all edges by their level as the primary key, their index as the secondary and the  $y$ -coordinate of the endpoint of edge as the ternary key (endpoint with less  $y$ -coordinate among two endpoints of an edge is chosen). Also, we sort all query points by their  $y$ -coordinates. Initially, all query points are assigned highest level  $\lceil \log n \rceil$  and index 0. Then, for each level  $i$ , from  $i = \lceil \log n \rceil$  to  $i = 0$  do the following:

- (i) Merge the set of edges and query points (note that all query points have the same level, equal to  $i$ ),
- (ii) Perform interval broadcasting to find, for each query point  $Z$ , the corresponding edge  $e$  the query point should be discriminated against. If the  $y$ -coordinate of  $Z$  is not between  $y$ -coordinates of endpoints of  $e$  then  $Z$  has been discriminated at level before. Depending on which side of  $e$  the query point  $Z$  is,  $Z$  calculates the index of chain at the next level it should be discriminated,
- (iii) Unmerge edges and query points (using former indices of query points),
- (iv) Re-sort query points by new indices, by compressing query points with answer "left" of corresponding edge in Step (ii) (query points with answer "right" will be also compressed) and (since both subsets of query points are sorted by new indices after compressing) merging "left" and "right" query points by their new indices. Give next level to all query points.

All query points will be located in the Step (ii) when  $i = 0$ . Since all steps (i)-(iv) take  $O(\log n)$  time, the time complexity of planar point location algorithm is  $O(\log^2 n)$ .

An  $O(\log^3 n)$  algorithm to construct Voronoi diagram of a set  $S$  of  $n$  planar points on a hypercube with  $n$  processors can be obtained by using Jeong and Lee [10] algorithm to solve the problem on mesh-connected computers, planar point location technique and presented data communication techniques. In [19] it is shown that all operations in the merge step of the algorithm can be implemented in  $O(\log^2 n)$  time.

#### Finding intersection of two convex polygons

We give a parallel algorithm for finding the intersection of two convex polygons  $P$  and  $Q$  with  $O(n)$  vertices all together by modifying the sequential method of [17].

By drawing a vertical line through each vertex of  $P$  and  $Q$  we divide  $P$  and  $Q$  into slabs. The leftmost and the rightmost vertices of  $P$  and  $Q$  (they can be found in  $O(\log n)$  time) divide both  $P$  and  $Q$  into two chains: the upper and the lower chain of vertices (denote them  $u_P, l_P, u_Q$  and  $l_Q$  respectively). The intersections  $A_1, A_2$  and  $A_3$  of a vertical line passing through a vertex  $A$  of a

chain with remaining three chains can be obtained in parallel (one processor per each vertex  $A$ ) by computing nearest points  $A'$  and  $A''$  of  $A$  to the left and to the right respectively in the considered chain and finding the intersection of  $A'A''$  with vertical line through  $A$ . Clearly  $A'$  and  $A''$  are the cousins of  $A$  in the chain. Upper and lower chains of  $P$  and  $Q$  can be formed by some translations and reversions from  $P$  and  $Q$ . Then desired intersections can be obtained by merging  $u_P \cup u_Q, u_P \cup l_Q, u_P \cup l_P, l_P \cup u_Q, l_P \cup l_Q, u_Q \cup l_Q$  and unmerging between two steps. Let  $P \cup Q$  denote the list of vertices of  $P$  and  $Q$  sorted together by  $z$ -coordinate (it can be constructed in  $O(\log n)$  time by merging upper and lower chains of  $P$  and  $Q$ ). Consider each slab defined by two neighboring points  $A$  and  $B$ . On the basis of the coordinates of points  $A, A_1, A_2, A_3, B, B_1, B_2$  and  $B_3$  (translation by 1 can be used to exchange the data) one can decide in  $O(1)$  time in parallel whether  $P$  and  $Q$  intersect within the slab and determine (at most two) points of  $P \cap Q$  which are located in the slab (these are either intersections of edges of  $P$  and  $Q$  or vertices of  $P$  or  $Q$  which are located inside the other polygon). So far we have detected all vertices of intersection of  $P$  and  $Q$  in  $O(\log n)$  time. However, we should order them to obtain their convex hull. For each vertex of intersection we decide whether it is an vertex of upper or lower chain of  $P \cap Q$  (this can be done in constant time). Also, we assign the vertex to left vertex of corresponding slab defined by points of  $P \cup Q$ . Thus each vertex of  $P \cap Q$  from upper convex hull chain (and similarly for lower convex hull chain). Now, upper chain of  $P \cap Q$  can be obtained by simply compressing points of  $P \cup Q$ , assuming that active points of  $P \cup Q$  are those having assigned a vertex of  $P \cap Q$ . Similarly we find lower convex hull chain and finally order vertices of  $P \cap Q$  by some translations and reverse steps.

This algorithm solves also the problem of detecting intersection of two convex polygons in parallel (linear separability). Clearly they intersect if at least one vertex of  $P \cap Q$  is found. The time complexity is still  $O(\log n)$ .

The described algorithm can be also implemented in optimal time on a mesh computer and in  $O(\log n)$  time on a CREW PRAM.

#### ECDF searching problem

Given a set  $S = \{p_1, \dots, p_n\}$  of  $n$  points in 2-dimensional space. A point  $p_i$  dominates a point  $p_j$  ( $p_i > p_j$ ) iff  $p_i[k] > p_j[k]$  for  $k = 1, 2$ , where  $p[k]$  denotes the  $k$ -th coordinate of a point  $p$ . The 2-dimensional ECDF searching problem consists of computing for each  $p \in S$  the number  $D(p, S)$  of points of  $S$  dominated by  $p$ .

Let the rank  $B(p, S)$  of a point  $p$  in the set  $S$  containing  $n$  points be the position of  $p$  in the set  $S$  sorted according to the  $y$ -coordinate of points, the rank of bottommost and uppermost points being 0 and  $n - 1$ , resp.

As a preprocessing step of ECDF searching algorithm, we sort points by  $z$ -coordinate. The rest of algorithm is best described recursively. Suppose  $S$  is divided into two subsets  $L$  and  $R$  of equal size with  $l[1] \leq r[1]$  for all  $l \in L$  and  $r \in R$ , both sorted by  $y$ -coordinate. After the recursive calls for  $L$  and  $R$  in parallel we will have  $D(l, L), D(r, R), B(l, L)$  and  $B(r, R)$  for all  $l \in L$  and  $r \in R$ . The main point is that the number of points from  $L$  which are below  $r \in R$  is  $\max\{B(l, L) \mid l[2] \leq r[2]\} = B(r, S) - B(r, R)$ . Therefore the final result will be obtained directly from the relations:

$$D(l, S) := D(l, L) \text{ for all } l \in L,$$

$$D(r, S) := D(r, R) + B(r, S) - B(r, R) \text{ for all } r \in R.$$

Unfolding recursion yields the iterative solution. Initially  $B = D = P = 0$  for each node  $x$  of hypercube.

```
FOR i = 0 TO d - 2 DO
  BEGIN
    MERGE consecutive blocks of size  $2^i$  in pairs;
    IF  $x_i = 1$  THEN  $D := D + P - B$ ;
     $B := P$ 
  END
```

In the merge procedure values  $B, D$  and  $P$  are exchanged whenever data are exchanged between nodes. The ranks of elements after merging are denoted by  $P$  and are easily obtained as the relative node's i.d. in the corresponding block of size  $2^{i+1}$ .

The running time of merging step is  $O(\log n)$  which give a total  $O(\log^2 n)$  time for ECDF searching problem.

The same algorithm solves also the maximal elements problem, i.e. the problem of determining points which are dominated by no other point. We replace the sign  $>$  by  $<$  in the definition of domination and look for points  $p$  with  $D(p, S) = 0$ . Maximal elements can also be determined directly by sorting and parallel prefix (with  $*$  = max) operation, as suggested in [3] for CREW PRAM model.

The 2-set dominance counting problem (computing for each point from  $A$  the number of points from  $B$  dominated by the point) and the maximal element problem for point sets in three-dimensional space can be solved on hypercube in  $O(\log^2 n)$  time by a similar iterative algorithm, using labeled functions from [3].

#### Closest points problem

A  $O(\log^2 n)$  hypercube solution to the problem of computing two points with the smallest distance among  $n$  given points based on a sequential method presented in [16] will be described. We again apply iterative approach rather than recursive one.

First we sort  $n$  points from given set  $S$  of points by  $x$ -coordinate. At a stage  $i$  (where  $i$  ranges from 0 to  $d-1$ ), let  $L$  and  $R$  be left and right halves of points in a given block of size  $2^i$ , respectively. Suppose  $L$  and  $R$  are both sorted by  $y$ -coordinate, and  $\delta_1(\delta_2)$  (the smallest distance between points in  $L(R, \text{resp.})$ ) are found. Let active elements of  $S$  be those with distance from a line separating  $L$  and  $R$  less than  $\delta = \min(\delta_1, \delta_2)$ . Compress a copy of active elements in both  $L$  and  $R$  and merge them to form list  $S'$  of active elements. Each active element from  $L(R)$  should calculate its distance to constant number of active elements in  $R(L)$  (at most six, as shown in [16]). By repeating interval broadcasting technique constant number of times (six) we inform each active element about neighboring elements in other set. Then active elements choose the nearest element from other set and minimum over obtained distances is found and compared to  $\delta$ . Now broadcast new value of  $\delta$  and merge  $L$  and  $R$  for the next stage.

#### Acknowledgements

The author would like to thank Frank Dehne (Carleton University) and the referees for valuable comments.

#### References

- [1] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C. Yap, "Parallel computational geometry," *IEEE Symp. Found. Comp. Sci.* (1985), pp. 468-477.
- [2] M.J. Atallah and M.T. Goodrich, "Parallel algorithms for some functions of two convex polygons," *Algorithmica*, to appear.
- [3] M.J. Atallah and M.T. Goodrich, "Efficient plane sweeping in parallel," *ACM Symp. Comp. Geom.* (1986), pp. 216-225.
- [4] K.E. Batcher, "Sorting networks and their applications," *Proc. Spring Joint Computer Conf.* (1968), pp. 307-314.
- [5] K.Q. Brown, Geometric transforms for fast geometric algorithms, Dept. of Computer Science, Carnegie-Mellon Univ. (1979).
- [6] A.L. Chow, "A parallel algorithm for determining convex hulls of sets of points in two dimensions," *Proc. Allerton Conf. Comm. Control and Comp.* (1981), pp. 214-223.
- [7] R. Cole and U. Vishkin, Faster optimal parallel prefix sums and list ranking, *Ultracomputer Note* 117, *Comp. Sci. TR 277*, (Feb. 1987).
- [8] F. Dehne, " $O(\sqrt{n})$  algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer," *Inform. Process. Lett.*, 22(1986), pp. 303-306.
- [9] F. Dehne and I. Stojmenović, "An  $O(\sqrt{n})$  algorithm for the ECDF searching problem for arbitrary dimensions on a mesh of processors," *Inform. Process. Lett.*, to appear.
- [10] C.S. Jeong and D.T. Lee, Parallel geometric algorithms on mesh-connected computers, Dept. of Electr. Eng. and Comp. Sci., Northwestern Univ., TR 87-02-FC-01, (1987).
- [11] D.T. Lee and F.P. Preparata, "Location of points in a planar subdivision and its applications," *SIAM J. Comp.*, 6,3 (1977), pp. 594-606.
- [12] M. Lu, "Constructing the Voronoi diagram on a mesh-connected computer," *IEEE Conf. Par. Proc.* (1986), pp. 806-811.
- [13] R. Miller and S.E. Miller, "Using hypercube multi-processors to determine geometric properties of digitized pictures," *Proc. IEEE Conf. Par. Proc.* (1987), pp. 638-640.
- [14] R. Miller and Q.F. Stout, Mesh computer algorithms for computational geometry (revised), Dept. Comp. Sci., Univ. Buffalo, State Univ. of New York, TR 86-18, (1987).
- [15] D. Nassimi and S. Sahni, "Data broadcasting in SIMD computers," *IEEE Trans. on Comp.* C-30, 2 (Feb. 1981), pp. 101-106.
- [16] F.P. Preparata and M.I. Shamos, Computational Geometry, An Introduction, Springer-Verlag, N.Y., (1985).
- [17] M.I. Shamos and D. Hoey, "Geometric intersection problems," *IEEE Symp. Found. Comp. Sci.* (1976), pp. 208-215.
- [18] I. Stojmenović, Parallel computational geometry, Washington State Univ., Pullman, CS-87-176, (1987).
- [19] I. Stojmenović, Computational geometry on a hypercube, Washington State Univ., Pullman, CS-87-180, (1987).
- [20] Q.F. Stout and B. Wager, Intensive hypercube communication I: Prearranged communication in link-bound machines, *Comp. Res. Lab. Univ. Michigan, CRL-TR-9-87*, (1987).
- [21] G.T. Toussaint, "Solving geometric problems with the rotating calipers," *Proc. IEEE MELECON '83*, Athens, Greece, (1983).
- [22] J.D. Ullman, Computational aspects of VLSI, *Comp. Sci. Press, Potomac, MD* (1984).