Computational Geometry On The OTIS-Mesh Optoelectronic Computer *

Chih-fang Wang
Department of Computer Science
Southern Illinois University
Carbondale, IL 62901
cfw@cs.siu.edu

Sartaj Sahni
Department of Computer and
Information Science and Engineering
University of Florida
Gainesville, FL 32611
sahni@cise.ufl.edu

Abstract

We develop efficient algorithms for problems in computational geometry—convex hull, smallest enclosing box, ECDF, two-set dominance, maximal points, all-nearest neighbor, and closest-pair—on the OTIS-Mesh optoelectronic computer. We also demonstrate the algorithms for computing convex hull and prefix sum with condition on a multi-dimensional mesh, which are used to compute convex hull and ECDF respectively. We show that all these problems can be solved in $O(\sqrt{N})$ time even with N^2 inputs.

Keywords *and* **phrases:** *optoelectronic computer, OTIS-Mesh, convex hull, smallest enclosing box, ECDF, two-set dominance, maximal points, all-nearest neighbor, closest-pair, prefix sum with condition.*

1 Introduction

It is well known that optical interconnect has superior power, speed, and crosstalk properties compared to electronic interconnect when the interconnect distance is more then a few millimeters [2, 5]. With this knowledge in mind, Marsden *et al.* [8], Hendrick *et al.* [4], and Zane *et al.* [18] have proposed an architecture in which the processors are partitioned into groups. Within each group electronic interconnect is used to connect the processors. Optical interconnect is used to connect processors in different groups.

The optical transpose interconnection system (OTIS) was proposed by Marsden *et al.* [8], where the processors are partitioned into groups of the same size. Processor i of group j is connected to processor j of group i via an optical connection. Krishnamoorthy *et al.* [6] have shown that

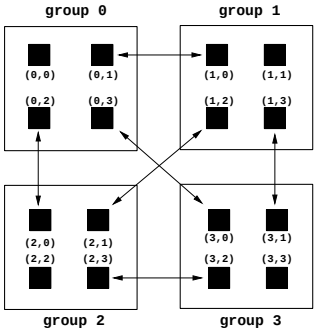

Figure 1. Example of OTIS connections with 16 processors

when the number of groups equals the number of processors within a group, the bandwidth and power efficiency are maximized, and system area and volume minimized. As a result, an N^2 processor OTIS computer is partitioned into N groups of N processors each for our study. Figure 1 shows a 16 processor OTIS architecture. The processor indices are of the form (G, P) where G is the group index and P the processor index within the group.

The OTIS-Mesh optoelectronic computer is a class of OTIS computers in which the electronic interconnect within each group follows the mesh paradigm [18, 13]. A 16 processor OTIS-Mesh computer is shown in Figure 2. The large boxes enclose groups of processors which are denoted by small boxes. The pair of numbers g, p inside a small box represents the group (g) and processor (p) indexes. Groups are numbered in row-major fashion, as they are laid out as a two-dimensional array. The pair (i, j) above a group denotes the row (i) and column (j) in which a group lies. Electronic links are indicated by arrows inside a group, while optical links are shown by arrows among groups. We shall refer to moves utilizing optical links as

*This work was supported, in part, by the National Science Foundation under grant CCR-9912395.

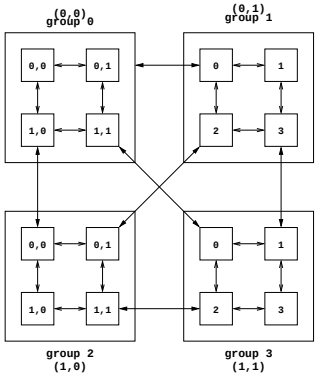

Figure 2. Example of OTIS-Mesh with 16 processors

OTIS moves.

Zane *et al.* [18] have shown that the OTIS-Mesh computer can simulate a $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ four-dimensional mesh computer. Sahni and Wang [13] have developed algorithms for data routing algorithms. Wang and Sahni have studied algorithms for basic operations [15] such as prefix sum, consecutive sum, concentrate, sort, and they have also developed algorithms for various matrix multiplication operations [17] and image processing problems [16].

In this paper, we develop algorithms for the following problems that arise in computational geometry: convex hull (CH), smallest enclosing box (SEB), ECDF, two-set dominance (2SD), maximal points (MP), all-nearest neighbor (ANN), and closest-pair for points (CPP). For each problem, we consider the two cases *(a)* An N point problem is to be solved on an N^2 processor OTIS-Mesh and *(b)* an N^2 point problem is to be solved on an OTIS-Mesh with N^2 processors. All of our algorithms run in $O(\sqrt{N})$ time.

2 Convex Hull

We wish to identify the extreme points of a set S of points in a plane. We assume that no three points are collinear.

2.1 $|S| = N$

Our algorithm employs the following result.

Theorem 1 *For any point $p_i \in S$, let $\{p_{j_0}, p_{j_2}, \ldots, p_{j_{N-2}}\} = S - \{P_i\}$ be such that the points $p_{j_0}, p_{j_2}, \ldots$ are sorted by the polar angle made by the vector $\overrightarrow{p_i p_{j_q}}$. Point p_i is an extreme point of S iff there is a k, $0 \leq k < N-1$, such that the (counterclockwise) angle between $\overrightarrow{p_i p_{j_k}}$ and $\overrightarrow{p_i p_{j_{(k+1) \bmod N-1}}}$ is more than π.*

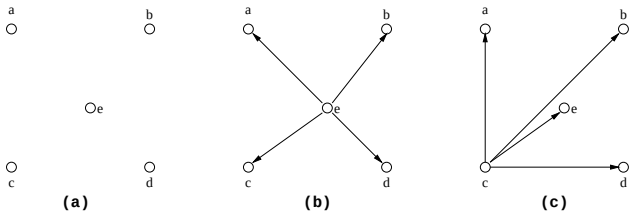

Figure 3. Example for Theorem 1:(a) original layout; (b) $p_i = e$; (c) $p_i = c$.

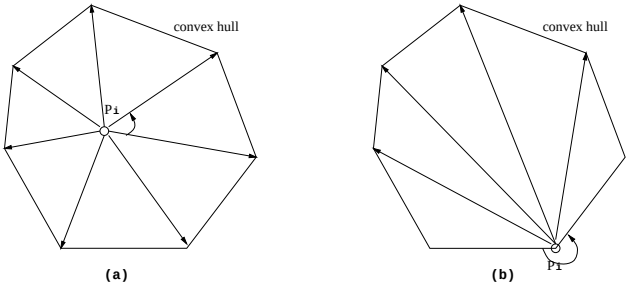

Figure 4. Relations of point p_i and the convex hull: (a) p_i is not an extreme point; (b) p_i is an extreme point.

Proof First consider the example of Figure 3(a). Figure 3(b) shows the case $p_i = e$. For this case, $S - \{p_i\} = \{b, a, c, d\}$ (in polar angle order). The counterclockwise angle between every pair of consecutive vectors is $\pi/2$. According to our theorem, point e is not an extreme point. Figure 3(c) shows the case when $p_i = c$. Now, $S - \{p_i\} = \{d, e, b, a\}$ (in polar angle order). The counterclockwise angle between the consecutive vectors $\overrightarrow{ca}$ and $\overrightarrow{cd}$ is $3\pi/2$, and according to our theorem, point c is an extreme point.

To prove the theorem, note that when p_i is not an extreme point, it must lie inside the convex hull (p_i cannot be on the hull boundary because we have assumed that no three points of S are collinear). Figure 4(a) shows the situation. Since the enclosing convex hull has at least three vertices, the angle between any two vectors originating at p_i and ending at two consecutive vertices of the hull is less than π. Therefore the angle between any two vectors that are adjacent in polar order is all less than π.

When p_i is an extreme point on the convex hull, the angle between the vectors to the hull vertices that immediately precede and follow p_i on the convex hull is more than π. ■

Theorem 1 results in the following algorithm to determine whether p_i is an extreme point of S.

Step 1: Sort the points $S - \{p_i\}$ by the polar angle of the vectors $\overrightarrow{p_i p_j}$, $p_j \in S - \{p_i\}$. Let $\{p_{j_0}, p_{j_2}, \ldots, p_{j_{N-2}}\}$

be the sorted sequence of points in $S - \{p_i\}$.

Step 2: If there is a k, $0 \leq k < N-1$, such that the counterclockwise angle between $\overrightarrow{p_i p_{j_k}}$ and $\overrightarrow{p_i p_{j_{(k+1) \bmod N-1}}}$ is more than π, then p_i is an extreme point; otherwise, p_i is not an extreme point.

On an N^2 processor OTIS-Mesh, we can perform the above two steps, concurrently, for all points $p_i \in S$. The points that are determined to be extreme points can then be sorted into polar order with respect to any point in the interior of the convex hull. The resulting sorted order of extreme points defines the convex hull of S. The algorithm is given below. The N points of S are initially distributed, one to a processor, over the processors of group 0.

Step 1: Perform an OTIS move of the points in group 0.

Step 2: Processor 0 of group i, $0 \leq i < N$, broadcasts the point it received in Step 1 to all processors in its group. All processors in a group now have the same point in their A registers.

Step 3: Perform an OTIS move on the points in the A registers. The data is received into B registers. Now, each group i processor has the point p_i in its A register and a point of $S - \{p_i\}$ in its B register.

Step 4: Each processor computes the polar angle of the vector defined by the points in its A and B registers.

Step 5: Each group sorts, into snake-like order, the angles computed by its processors.

Step 6: Each processor in a group checks the condition of Theorem 1 by computing the angle between the vectors defined by p_i, the point in its B register, and the point in the B register of the next processor in the snake like order.

Step 7: Processor 0 of each group is notified by group processors that determine a point p_i is an extreme point.

Step 8: The points that pass the test of Theorem 1 are transmitted to group 0 via an OTIS move.

Step 9: The extreme points accumulated by group 0 are sorted by polar angle order.

Step 2 takes $2(\sqrt{N}-1)$ electronic moves, Step 5 takes $(4+\epsilon)\sqrt{N}$ electronic moves [10], and Steps 6 and 7 take up to $2(\sqrt{N}-1)$ electronic moves each. For Step 9, we must first find a point that is in the interior of the convex hull. This is done by identifying any three of the hull vertices and then computing the centroid of these three vertices. The hull vertices are first moved leftwards in group 0. Then, these vertices are moved downwards to processor (0,0) until processor (0,0) has 3 vertices. The centroid is broadcast to all group 0 processors, the polar angles are computed, and then the hull vertices are sorted by polar angle. The sort takes $(4+\epsilon)\sqrt{N}$ electronic moves [10] and the remaining operations of Step 9 take $4(\sqrt{N}-1)$ electronic moves. Overall, our convex hull algorithm takes $(18+\epsilon)\sqrt{N}$ electronic and 3 OTIS moves.

2.2 $|S| = N^2$

The $|S| = N^2$ $N \times N$ mesh convex-hull-algorithm of [9] is easily generalized to run on an $N^{2/d} \times N^{2/d} \times \cdots \times N^{2/d}$ d-dimensional mesh. The run time is $O(dN^{2/d})$. We simulate the resulting 4-dimensional mesh algorithm on a $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ OTIS-Mesh using the simulation strategy described in [18]. This results in an $O(\sqrt{N})$ convex hull algorithm for our N^2 processor OTIS-Mesh. A closer analysis reveals that the resulting algorithm performs $\geq 152\sqrt{N} + o(\sqrt{N})$ electronic moves and $\geq 144\sqrt{N} + o(\sqrt{N})$ OTIS moves.

3 Smallest Enclosing Box

In the smallest enclosing box (SEB) problem, we are given a set S of coplanar points and are to find a minimum area rectangle that encloses all points in S. It is well known [3] that the SEB of S has one side that is collinear with an edge of the convex hull of S and that the remaining three sides of the SEB pass through at least one convex hull vertex each.

3.1 $|S| = N$

Step 1: Compute the convex hull as in Section 2.1.

Step 2: Broadcast the hull vertices from group 0 to all remaining groups.

Step 3: Group i determines the ith hull edge (p_l, p_r) and broadcasts this to all processors within the group.

Step 4: Each group i processor determines the distance h between its hull vertex q (if any) and the ith hull edge (p_l, p_r) as well as the distance w to the perpendicular bisector L of the ith hull edge. If q and p_l are on the same side of L, set $l = -w$ and $r = 0$; otherwise, set $l = 0$ and $r = w$.

Step 5: Processor 0 of each group i computes the maximum of the h's and r's in its group and the minimum of the l's in its group. The area of the SEB that has one side collinear with (p_l, p_r) is $A_i = h_{\max} * (r_{\max} - l_{\min})$.

Step 6: Perform an OTIS move on the A_i's. Now all A_i's are in the group 0 processors.

Step 7: Processor 0 of group 0 determines the minimum A_i.

Step 1 takes $(18+\epsilon)\sqrt{N}$ electronic and 3 OTIS moves (see Section 2.1), the remaining steps take $8\sqrt{N}$ electronic and 3 OTIS moves. Therefore, the overall complexity of the above SEB algorithm is $(26+\epsilon)\sqrt{N}$ electronic and 6 OTIS moves.

3.2 $|S| = N^2$

The steps in any SEB algorithm (including the one of Section 3.1) are:

Step 1: Compute the convex hull of S.

Step 2: Find the furthest (*i.e.*, perpendicular distance to the edge) hull vertex for each hull edge.

Step 3: Find the leftmost (*i.e.*, distance is measured from the perpendicular bisector of the hull edge) hull vertex for each hull edge.

Step 4: Find the rightmost hull vertex for each hull edge.

Step 5: Each processor computes the enclosing box defined by the edge and points from the previous steps.

Step 6: Find the box with minimum area.

When $|S|$ equals the number of processors, Step 1 can be done in $O(\sqrt{N})$ time using the algorithm of Section 2.2. Step 5 takes $O(1)$ time, and Step 6 takes $O(\sqrt{N})$ time. Steps 2, 3, and 4 can also be done in $O(\sqrt{N})$ time by adapting the technique of [9].

Let $s(\overline{xy})$ be the slope of the convex hull edge $\overline{xy}$. Let h be the convex hull vertex that is farthest from $\overline{xy}$, let p be the hull vertex that precedes h, and let q be the hull vertex that follows h. In [9] it is shown that $s(\overline{hp}) \leq s(\overline{xy}) < s(\overline{hq})$. Therefore, if the hull edges are sorted by slope, the vertex h is easily identified. The leftmost and rightmost hull vertices may be found in a similar fashion by using the slopes $s(\overline{xy}) \pm \pi/2$. The algorithm to find the hull point h that is farthest from the hull edge $\overline{xy}$ is given below. This algorithm actually finds the farthest hull vertex for all edges $\overline{xy}$ that are on the upper hull (see Step 2.1). The algorithm for all edges of the lower hull is similar. Note that for any upper hull edge, the farthest hull vertex is on the lower hull.

Step 2.1: Let l and r, respectively, be the hull vertices with the least and maximum x-coordinates. Ties are broken arbitrarily. The line $\overline{lr}$ partitions the convex hull into two parts–lower and upper.

Step 2.2: Each processor that has a hull edge computes the slope of this edge and creates a tuple $t = (s, i, u, d)$, where s is the slope of the hull edge, i is the processor index, u is true iff the hull edge is part of the upper hull, and d is empty if u is true, and d is the right end point of the hull edge when u is false.

Step 3: Sort the tuples by the slope s. Ties are broken using the u value (false $<$ true). Note that two edges on the upper (lower) hull cannot have the same slope. The tuple (if any) $t = (s, i, u, d)$ that a processor has is called its t-tuple. This t-tuple originated in processor i.

Step 2.4: Each processor that has a t-tuple with u =false creates a t'-tuple which is a copy of its t-tuple plus the processor's index. That is, $t' = (t, p)$.

Step 2.5: The t'-tuples are first ranked, then concentrated using their ranks, and finally generalized using the p values. Following the generalization, each processor has a t'-tuple.

Step 2.6: Each processor that has a t-tuple T with u =true sets the d value of this t-tuple T to the d value of its t'-tuple. This d value is the hull vertex that is farthest from the edge that resulted in the t-tuple T.

Step 2.7: Route the t-tuples back to their originating processors using the i values of the t-tuples.

Using the rank, concentrate, generalize, and sort algorithms of [15], we can complete the above seven steps in $O(\sqrt{N})$ time. The leftmost and rightmost hull points for each hull edge can similarly be found in $O(\sqrt{N})$ time. Following this, each edge can determine the area of the SEB that has one side collinear to this edge in $O(1)$ time, and the minimum of these area cab be then determined using an additional $O(\sqrt{N})$ time. Therefore, the overall SEB computation takes $O(\sqrt{N})$ time. A closer analysis reveals that the resulting algorithm performs $\geq 375\sqrt{N} + o(\sqrt{N})$ electronic moves and $\geq 144\sqrt{N} + o(\sqrt{N})$ OTIS moves.

4 ECDF

In the ECDF (*empirical cumulative distribution function*) problem, we are given a set S of distinct points in a plane. Point $p_i = (x_i, y_i) \in S$ dominates point $p_j = (x_j, y_j) \in S$ if and only if $x_i \geq x_j$ and $y_i \geq y_j$. For each point $p_i \in S$, we are to determine the number of points that it dominates. Figure 5 shows three points p_1, p_2, and p_3. p_1 and p_2 dominate p_3, p_1 does not dominate p_2, and p_2 does not dominate p_1. The numbers of points dominated by p_1, p_2, and p_3, respectively, are 1, 1, and 0.

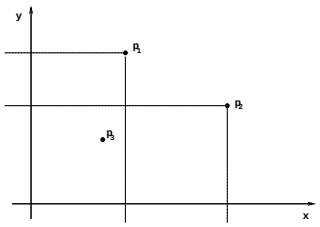

Figure 5. Example of dominating relation

4.1 $|S| = N$

We have enough processes to run the brute force algorithm–perform all N^2 pairs of point comparisons and count the number of points dominated by each point p_i—efficiently. The steps are given below. We begin with point p_i in processor i of group 0; *i.e.*, in processor $(0, i)$.

Step 1: Perform an OTIS move on the points initially in group 0. Now processor $(i, 0)$ has point p_i.

Step 2: Processor $(i, 0)$ broadcasts point p_i to the remaining processors in its group. Each processor saves its point in register A as well as register B.

Step 3: Perform an OTIS move on the register B data. Now, processor (i, j) has point p_i in register A, and point p_j in register B.

Step 4: Each processor sets its C register to 1 if its A register point dominates its B register point; the C register is set to 0 otherwise.

Step 5: Processor $(i, 0)$ computes the sum of the C values in its group.

Step 6: Perform an OTIS move on the sums computed in Step 5.

The complexity of the above algorithm is readily seen to be $4(\sqrt{N} - 1)$ electronic and 3 OTIS moves.

4.2 $|S| = N^2$

We begin with one point in each of the N^2 processors. The strategy is given below.

Step 1: Sort the N^2 points by x-coordinate, and within x-coordinate by y-coordinate.

Step 2: Each point $p_i = (x_i, y_i)$ determines the number of points to its left that have y-coordinates $\leq y_i$.

Step 3: The counts determined in Step 2 are routed back to the originating processors for the individual points.

Steps 1 is done in $O(\sqrt{N})$ time using the OTIS-Mesh sort algorithm of [15]. Step 2 is an example of the general prefix computation (GPC) operation described in [1]. We shall show below, how to do Step 2 in $O(\sqrt{N})$ time. Step 3 may be done in $O(\sqrt{N})$ time by sorting on the originating processor indexes.

Step 2 can be done in $O(\sqrt{N})$ time using a divide-and-conquer algorithm. Let $T = \{p_0, \ldots, p_{N^2-1}\}$ be the sorted point sequence. $C(p_i, T)$ be the number of points to the left of point $p_i = (x_i, y_i)$ that have y-coordinates $\leq y_i$. Let L and R, respectively, denote the left and right halves of T. Let T_y (L_y, R_y) denote the point sequence T (L, R) reordered by y-coordinate and within y-coordinate by x-coordinate. For each $p_i \in L$ (R), let $p'_i \in R_y$ (L_y) be the point with highest index of p_i. The the following equations are easily verified [1]:

$$C(p_i, S) = \begin{cases} C(p_i, L), & p_i \in L \\ C(p_i, R) + P(p'_i, L_y), & p_i \in R \end{cases}$$

$$P(p_i, S_y) = \begin{cases} P(p_i, L_y) + P(p'_i, R_y), & p_i \in L \\ P('p_i, L_y) + P(p_i, R_y), & p_i \in R \end{cases}$$

where $P(p_i, S_y)$ is one more than the number of points to the left of p_i in S_y.

We may compute C and P in $O(n)$ time on an n^d processor d-dimensional mesh by partitioning the mesh into 2^d $(n/2)^d$ processor submeshes, recursively computing the Cs and Ps of neighboring submeshes until we have the overall C and P. The combining involves sorting, ranking, concentration, and generalization, and takes $O(n)$ time. If $T(n)$ is the time required to compute the Cs and Ps on an n^d processor d-dimensional mesh, then

$$T(n) = T(n/2) + O(n) = O(n)$$

We may simulate the 4-dimensional mesh algorithm on an N^2 processor OTIS-Mesh using the techniques of [18] and compute the Cs and Ps in $O(\sqrt{N})$ time. Therefore, we can solve the N^2 point ECDF problem in $O(\sqrt{N})$ time on an N^2 processor OTIS-Mesh. A closer analysis reveals that the resulting algorithm performs $\geq 192\sqrt{N} + o(\sqrt{N})$ electronic moves and $\geq 136\sqrt{N} + o(\sqrt{N})$ OTIS moves.

5 Two-Set Dominance

In the *two-set dominance* (2SD) problem, we are given two sets S_1 and S_2 of points in a plane. For each point $p \in S_1$ (S_2) we wish to determine the number of points in S_2 (S_1) that are dominated by p. This problem is quite similar to the ECDF problem and is solved in $O(\sqrt{N})$ time by making minor modifications to the ECDF algorithm. Instead of counting all points dominated by p, we only count dominated points that are in a different set from p.

6 Maximal Points

A point $p \in S$ is maximal if and only if p is not dominated by any point in S. In the *maximal points* (MP) problem, we wish to identify all maximal points. This problem may be solved using the ECDF algorithms. Instead of counting the number of points that p dominates, we count the number of points that dominates p. Further simplification comes from realizing that instead of keeping a count of the number of points that dominate p, we can keep a boolean value that is true if and only if at least one point that dominates p has been detected. The complexity of the algorithm remains $O(\sqrt{N})$.

7 All-Nearest Neighbor

In this problem, we are given a set S of points. For each point $p \in S$, we are to find another point $q \in S, q \neq p$, such that the distance between p and q is minimum.

7.1 $|S| = N$

The algorithm for this case is similar to that given in Section 4. Steps 4-6 of that algorithm are changed to

Step 4': Each processor sets its C register to the distance between between the points in its A and B registers (if the points are the same, the C register is set to ∞).

Step 5': Processor $(i, 0)$ computes the minimum of the C values in its group and thereby identifies the nearest neighbor of the point in the group's A registers.

Step 6': Perform an OTIS move on the nearest neighbors computed in the $(i, 0)$ processors.

The overall complexity of the algorithm is the same as that for the ECDF algorithm of Section 4.

7.2 $|S| = N^2$

We use the divide-and-conquer strategy described in [12] and used in [9] to solve the problem on a 2-dimensional mesh. The steps are:

Step 1: Partition the N^2 points into d groups by x-coordinates and solve the all nearest neighbor problem in each group.

Step 2: Partition the N^2 points into d groups by y-coordinates and solve the all nearest neighbor problem in each group.

Step 3: Each point determines the closer of the two neighbors determined in Steps 1 and 2.

Step 4: The partitions of Steps 1 and 2 define d^2 rectangular regions. In each region, label the points that are closer to a corner of the region than to their nearest neighbor as determined in Step 3. There are at most 8 marked points in each region [9].

Step 5: Circulate the at most $8 \times d^2$ marked points through all N^2 processors. During this circulation process, these marked points determine their nearest neighbors.

The correctness of the algorithm has been established in [9] with $d = 5$. The algorithm is easily run on a 4-dimensional N^2 processor mesh with $d > 16$ so as to have complexity of $O(\sqrt{N})$. This results in an $O(\sqrt{N})$ OTIS-Mesh algorithm [18]. A closer analysis reveals that the resulting algorithm performs more than $8340\sqrt{N} + o(\sqrt{N})$ electronic moves and more than $8336\sqrt{N} + o(\sqrt{N})$ OTIS moves!

8 Closest-Pair of Points

The *closest-pair for points* (CPP) in S can be found by first solving the all nearest neighbor problem of Section 7 and then determining the closest pair from the nearest neighbor of each point. The additional effort needed is $2(\sqrt{N} - 1)$ electronic moves when $|S| = N$, and $4(\sqrt{N} - 1)$ electronic plus one OTIS moves when $|S| = N^2$.

9 Conclusion

We have shown that several computational geometry problem—convex hull, smallest enclosing box, ECDF, two-set dominance, maximal points, all-nearest neighbor, and closest-pair of points—can be solved in $O(\sqrt{N})$ time on an N^2 processor OTIS-Mesh. The algorithms for N points have much small constant factors than do the corresponding algorithms for N^2 points.

References

[1] S. Akl. *Parallel Computation: Models and Methods*. Prentice Hall, 1997.

[2] M. Feldman, S. Esener, C. Guest, and S. Lee. Comparison between electrical and free-space optical interconnects based on power and speed considerations. *Applied Optics*, 27(9):1742–1751, May 1988.

[3] H. Freeman and R. Shapira. Determining the minimal-area encasing rectangle for an arbitrary closed curve. *Communications of ACM*, 18:409–413, 1975.

[4] W. Hendrick, O. Kibar, P. Marchand, C. Fan, D. V. Blerkom, F. McCormick, I. Cokgor, M. Hansen, and S. Esener. Modeling and optimization of the optical transpose interconnection system. In *Optoelectronic Technology Center, Program Review*, Cornell University, Sept. 1995.

[5] F. Kiamilev, P. Marchand, A. Krishnamoorthy, S. Esener, and S. Lee. Performance comparison between optoelectronic and vlsi multistage interconnection networks. *Journal of Lightwave Technology*, 9(12):1674–1692, Dec. 1991.

[6] A. Krishnamoorthy, P. Marchand, F. Kiamilev, and S. Esener. Grain-size considerations for optoelectronic multistage interconnection networks. *Applied Optics*, 31(26):5480–5507, Sept. 1992.

[7] M. Kunde. Routing and sorting on mesh-connected arrays. In *Proceedings of the 3rd Agean Workshop on Computing: VLSI Algorithms and Architectures, Lecture Notes on Computer Science*, volume 319, pages 423–433. Springer Verlag, 1988.

[8] G. C. Marsden, P. J. Marchand, P. Harvey, and S. C. Esener. Optical transpose interconnection system architectures. *Optics Letters*, 18(13):1083–1085, July 1 1993.

[9] R. Miller and Q. F. Stout. Mesh computer algorithms for computational geometry. *IEEE Transactions on Computers*, 38(3):321–340, Mar. 1989.

[10] M. Nigam and S. Sahni. Sorting n^2 numbers on $n \times n$ meshes. In *Proceedings of the seventh International Parallel Processing Symposium (IPPS'93)*, pages 73–78, Newport Beach, California, 1993.

[11] M. Nigam and S. Sahni. Computational geometry on a reconfi gurable mesh. In *Proceedings of the International Parallel Processing Symposium*, pages 86–93, 1994.

[12] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[13] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Mesh optoelectronic computer. In *Proceedings of the fourth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'97)*, pages 130–135, Montreal, Canada, 1997.

[14] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Hypercube optoelectronic computer. *Informatica*, 22:263–269, 1998.

[15] C.-F. Wang and S. Sahni. Basic operations on the OTIS-Mesh optoelectronic computer. *IEEE Transactions on Parallel and Distributed Systems*, 9(12):1226–1236, 1998.

[16] C.-F. Wang and S. Sahni. Image processing on the OTIS-Mesh optoelectronic computer. *IEEE Transactions on Parallel and Distributed Systems*, 11(2):97–109, 2000.

[17] C.-F. Wang and S. Sahni. Matrix multiplication on the OTIS-Mesh optoelectronic computer. *IEEE Transactions on Computers*, 50(7):635–646, 2001.

[18] F. Zane, P. Marchand, R. Paturi, and S. Esener. Scalable network architectures using the optical transpose interconnection system (OTIS). In *Proceedings of the second International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96)*, pages 114–121, San Antonio, Texas, 1996.