A Survey of Parallel Computational Geometry Algorithms[1]

by

Frank Dehne and **Jörg-Rüdiger Sack**
School of Computer Science
Carleton University
Ottawa, Ontario
Canada K1S 5B6

Abstract

We survey computational geometry algorithms developed for various models of parallel computation including the PRAM, hypercube, mesh-of-processors, linear processor array, mesh of trees, and pyramid.

1. Introduction

In this paper we survey a number of results in a recent and fast growing field of research: the design of parallel algorithms for computational geometry problems.

During recent years, computational geometry has emerged as a field of its own and generated a large number of results dealing with the computational complexity of geometric problems. Its recent offspring, parallel computational geometry, is concerned with the computational complexity of geometric problems under parallel models of computation.

There are mainly two reasons why parallel algorithms for geometric problems have become of special interest:

- A steadily increasing number of parallel machines has become commercially available.
- Geometric algorithms are mainly used for on-line applications where short response times are a necessity. However, these geometric applications often require large amounts of data to be processed which makes it hard to obtain reasonable response times on standard sequential computers, even if optimal algorithms can be applied.

In contrast to sequential computational geometry, there exists a variety of models which are considered for designing parallel geometry algorithms. The following are some of the most commonly used architectures which will be used throughout this paper:

(a) The parallel random access machine, PRAM (of size n): a set of n processors which are all connected to a global shared memory (Figure 1a). In the CREW PRAM model, processors are allowed to read concurrently from one memory location but may not write concurrently into it; in the CRCW PRAM model, both operations can be executed concurrently.

(b) The hypercube (of size $n = 2^d$): a set of n processors $P_0,\ldots,P_{n-1}$ with O(1) memory space, each,

[1] Research supported by the Natural Sciences and Engineering Research Council of Canada.

F. Dehne and J.-R. Sack, "A survey of parallel computational geometry algorithms," invited paper, in Proc. *International Workshop on Parallel Processing by Cellular Automata and Arrays (PARCELLA)*, East-Berlin (GDR), 1988, Springer Verlag, Lecture Notes in Computer Science, Vol. 342, pp. 73-88.

where processors P_i and P_j are connected by a communication link if the binary representations of i and j differ in exactly one bit (Figure 1b).

(c) The mesh-of-processors (of size $n = m^2$): a set of n processors, with O(1) memory space, each, arranged in a square grid where each processor is connected to its direct neighbors (Figure 1c).

(d) The linear processor array (of size n): a set of n processors, with O(1) memory space, each, arranged in linear order where each processor is connected to its (at most two) direct neighbors (Figure 1d).

(e) The mesh of trees (of size n): a mesh-of-processors of size n and in addition, for every row and column, a tree of processor whose leaves are the processors of that row or column, respectively.

(f) the pyramid (of size n): a sequence of $k=\log_4 n$ meshes-of-processors $M_1,...,M_k$ of size n, $\frac{n}{4}, \frac{n}{16}, \ldots,$ 1 where every processor in M_{i+1} is connected to a quadruple of processors in M_i (Figure 1f).

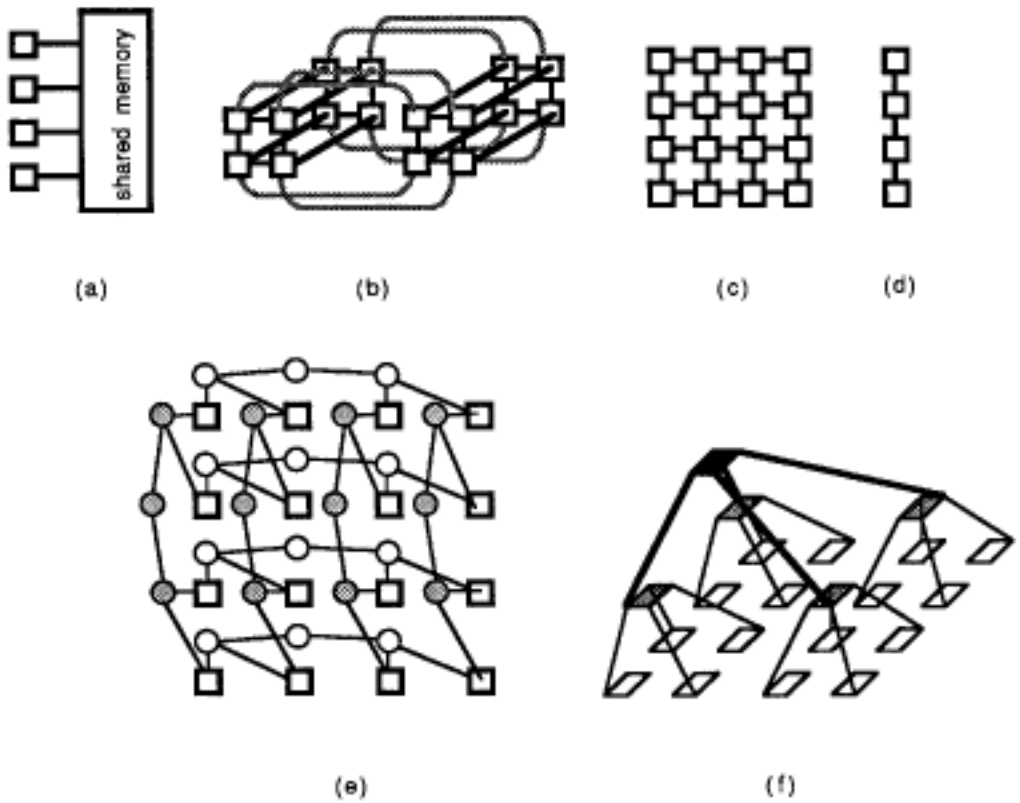

Figure 1: (a) PRAM, (b) Hypercube, (c) Mesh-of-Processors
(d) Linear Processor Array, (e) Mesh of Trees, (f) Pyramid

Since the parallel complexity of a geometric problem is best studied using the PRAM model, a large number of geometric algorithms has been designed for this model. In fact, Yap [Y87] has shown by that "most" computational geometry problems are in NC*; this result is derived by a reduction to the cell decomposition problem which is known to be in NC [KY85]. Although this theoretical result is very important, in practice it is still necessary to design parallel PRAM algorithms for the individual problem

because, in general, the reduction technique does not yield optimal solutions. A number of these results will be presented in the forthcoming sections.

The major disadvantage of PRAM algorithms is, however, that even a CREW PRAM can not actually be built because it requires each individual memory cell and any combination of them to be concurrently accessible by an unbounded number of processors in constant time. In practice, a multi-processor must either be a set of processors connected by a bus system (which limits the number of processors which may be connected to the bus without degenerating performance), or a bounded degree network of processors connected by point to point communication links, or a combination of both.

This motivated the studies of parallel algorithms for other parallel models: the hypercube (this is not a bounded degree network but it can be efficiently simulated on one [PV79]), mesh-of-processors, linear processor array, mesh of trees, and the pyramid. These model resemble architectures which have actually been built, at least as a prototype. As it turns out, algorithm design on these models is more complicated than on the PRAM because a variety of data routing problems have to be solved which do not occur in the PRAM. Because of its global shared memory, the PRAM allows e.g. instant communication between any pair of processors.

Since the major reason for studying parallel architectures is a decrease of the execution time, the obvious question is, how much of a speed-up can we expect.
Clearly, a lower bound of $T(n)$ for the sequential time complexity of a particular problem implies that, on any parallel machine with p processors, an $O(\frac{T(n)}{p})$ time algorithm is optimal. In general, poly-logarithmic time algorithms are considered efficient for parallel machines with a linear number of processors [Y87]. For processor networks, however, the lower bound for the majority of problems is the diameter D of the network. For comparing two data items stored in two processors, D time steps may be necessary to route the data from one processor to the other.
Hence, for most problems $O(\sqrt{n})$ and $O(n)$ time solutions are optimal for the mesh-of-processors and linear processor array, respectively, and $O(\log n)$ time algorithms are optimal for the hypercube, mesh of trees, and pyramid. In this sense, the latter ones appear to be superior to the former ones. However, the mesh-of-processors and linear processor array have the major advantages that they are easy to scale and that the length of all communication links is the same and constant. For the hypercube, mesh of trees, and pyramid this is not the case; in fact, whatever geometric layout is used for building these networks, the maximum wire length increases with the number of processors. In the above computational models, however, communication time is considered to be constant between any two adjacent processors. In this sense, these models are not realistic and one can also consider other complexity models which take wire length into account [CM81], [MC80]. However, the majority of authors considers communication between adjacent processors a constant time operation which will also be done for the remainder of this paper.

In the following we will survey parallel algorithms for geometric problems which have been proposed for the parallel models listed above. The paper is sorted by geometric problems which are

grouped similar to [LP84]. For each group of geometric problems, a survey of parallel algorithms is given. Section 2 surveys convex hull and related problems, Section 3 intersection problems, Section 4 proximity and geometric searching problems, Section 5 visibility problems, and Section 6 decomposition problems. It is, however, in the nature of such a survey that it is not complete. An omission of a paper is not meant to imply a judgement on the part of the authors.

2. Convex Hull

Among all problems studied in computational geometry, the determination of the convex hull has probably received the most attention in the sequential or parallel model of computation. This is due to the variety of applications this problem arises in (e.g., in robotics to approximate complex objects), as well as due to its occurrence as intermediate step when solving other computational geometry problems (e.g., in the calculation of the diameter of a set of points).

Convex hull problems have been studied for sets of points, polygons, and polyhedra, in the plane, 3-space and d-dimensional space. Variants of the problem are:

Identify Convex Hull

determine the vertices of the convex hull

Determine Convex Hull Polygon

compute the convex hull polygon for a set of points, or for a polygon, in the plane (the output is an ordered list of the convex hull vertices),

Determine Convex Hull Polyhedron

compute the convex polyhedron for a set of points, or for a polyhedron, in space.

A related problem is that of finding the *maxima* of a set in 2 (or 3) dimensions where a point p is a maximum if there is no other point in the set which has larger x, y (and z) coordinate than p. For a digitized object the set of maxima (sorted by x-coordinate) is called the 1^{st} *contour*. The k^{th} contour of an object S is defined recursively as:

1^{st} contour(S) := sorted maxima of S

$k+1$ contour(S) := 1^{st} contour(S - (1^{st} contour(S) $\cup$ ... $\cup$ k^{th} contour(S)),

where the operator "-" between sets denotes the set difference.

Digitized objects are frequently described by rectilinear (orthogonal) polygons. For these objects, the notion of rectilinear convexity has been studied. A polygon is *rectilinearly convex* if any horizontal or vertical line intersects the polygon in at most one connected region. The rectilinear hull of a rectilinear polygon P is the smallest rectilinearly convex polygon containing P. We define the k^{th} rectilinear hull analogously to contours. The following problems have been studied:

Determine Rectilinear Hull

Determine the rectilinear hull for a given object

Determine All k^{th} Rectilinear Hulls

Determine all k^{th} rectilinear hulls for a given object

On the 1-dimensional array of processors, Chazelle [C84] gives simple O(n) algorithms for both, Identify Convex Hull and Determine Convex Hull Polygon problem for sets of points in the plane. Each processor containing a point p can determine whether it is a convex hull vertex or not by computing the maximal wedge centered at p with respect to all other points. If this wedge is convex and contains all other points of the set then p is a convex hull vertex; otherwise, p is interior to the convex hull. This operation is done by propagating the point p through the array, thereby visiting each other element of the point set once and updating the wedge, if necessary, at each processor visited. All points can perform this task in parallel by a "fold-over" operation defined by Chazelle which can be seen as a row-rotation. This yields the O(n) solution to the Identify Convex Hull problem. Having solved the Identify Convex Hull problem, the Determine Convex Hull Polygon problem can be solved by sorting each of the convex hull points by polar angle around any interior point. Since sorting on a 1-dimensional array is performed in O(n) time, an O(n) solution to the Determine Convex Hull Polygon problem is derived. (For the latter problem, Chazelle describes a different approach.)
A variant of the 1-dimensional processor array, allowing cyclic shift operations among the array elements, has been introduced in [CCL87]; the convex hull algorithms derived for this model are similar to ones just discussed.

For the mesh-of-processor of size n, Miller and Stout [MS86] give $O(\sqrt{n})$ time algorithms for the convex hull problems in two dimensions. Their approach is based on the divide and conquer paradigm. They first partition the data into four roughly equal parts by sorting the points by x-coordinates and, recursively, solve the problems in each of the four regions. The remaining problem consists of merging the four hulls. To merge two disjoint convex hulls, two bridges connecting the hulls need be found. A bridge has the property that its endpoints belong to different convex hulls and that all other points of both hulls lie on the same side of the line defined by the bridge. By using a binary search on the sorted sub-list of vertices defining the two convex hulls a bridge can be found in O(log n) steps. Each step reduces the number of points to be considered by a constant factor. By using data compression after each step, the total time complexity of the algorithm becomes $O(\sqrt{n})$. An alternate $O(\sqrt{n})$ convex hull algorithm was presented in [S87].

For the mesh-of-processor architecture Dehne, Sack and Stojmenovic´ have shown that the 3-dimensional convex hull of a set of n points in 3-space can be found in $O(\sqrt{n}\ \log(n))$ time [DSS88]. Their approach is using in addition to a divide and conquer and data compression, an efficient techniques for finding supporting planes in 3 dimensions due to [DK87].

They also noted a space/time trade-off by showing that an $O(\sqrt{n})$ algorithm can be designed for a mesh in which each processor has O(log n) extra local memory. It is open whether an $O(\sqrt{n})$ algorithm can be found for the usual mesh-of-processor architecture (i.e., each processor has only O(1) memory).

For digitized pictures, convexity is defined as follows: a digitized point set is convex iff the corresponding set of integer lattice points is convex. The following results have been obtained [MS84, KE86, DSS87].

The convex hull of a digitized point set is determined in $O(\sqrt{n})$ time on a mesh-of-processors of size n. Since for certain applications the input point set may belong to differently labeled sets, the convex hull points for each of these sets may have to be computed. This problem can also be solved in $O(\sqrt{n})$ time using a variation of the above algorithm [MS84].

Kumar and Eshaghian studied computational geometry algorithms on a mesh-of-trees os size n [KE86]. They showed that in O(log n) time the convex hull vertices of a digitized picture can be identified and enumerated. For several figures stored simultaneously on the mesh-of-trees, the convex hulls of all figures can be determined in $O(\log^4 n)$ time.

Dehne, Sack, Santoro [DSS87] have given optimal algorithms for the following problems related to convex hulls: determine all k^{th} contours and all rectilinear k-hulls of a digitized set of points. Furthermore, their results yield a parallel solution to the problem of finding all longest common subsequences of two strings. The latter result improves on the algorithm by [RT85] in that it computes all such subsequences and all processing elements are of the same type.

For the hypercube architecture few results are known, as yet:
On the Intel IPSC Miller and Miller discuss design and implementation issues for a convex hull algorithm based on the gift-wrapping principle [MM87].
In [S87] Stojmenovic' showed that the 2-dimensional convex hull can be determined in $O(\log^2 n)$ time by adapting the PRAM algorithm due to [AG85]. The trivial lower bound is O(log n), thus this result is not optimal. However, solving the convex hull problem takes at least as long as sorting n numbers which currently requires $O(\log^2 n)$ time. Thus, better convex hull algorithms can not be found before a better sorting algorithm has been discovered.

Many results have been obtained on the PRAM model(s). We commence our discussion with an argument due to Akl [A82] who remarked that if one allows the number of processors to be polynomial in the number of input points then a constant time algorithm can be designed to identify the convex hull vertices. The reader will probably see that $O(n^4)$ processors suffice to obtain a constant time algorithm. This is using the simple geometric argument that a point is not a convex hull vertex if it is enclosed in a triangle formed by three other points in the set. Akl reduces this bound to $O(n^3)$ [A82]. Akl also presented

an optimal algorithm for computing the convex hull of n points in $O(n^{\varepsilon} \log h)$ using $O(n^{1-\varepsilon})$ processors, where h denotes the number of edges on the convex hull [A84].

It is easy to see that, using n processors, an $O(\log^2 n)$ time PRAM-algorithm can be designed by implementing the sequential divide-and-conquer algorithm in parallel (see [C80 , NMB81]).
Atallah and Goodrich presented an optimal O(log n) divide-and-conquer algorithm for determining the convex hull (polygon) of n points in the plane for the CREW-PRAM of size n [AG85]. A similar strategy was employed by [ACGDY85]. The idea is to split the point-set into $\sqrt{n}$ equal-sized subproblems and subsequently solve the subproblems recursively in parallel. The merging step consist of creating the bridges and is done by a reduction to sorting. Wagener independently solved the problem in the same time using also the divide-and-conquer paradigm [W85]; his approach is to half the problem rather than splitting it into $\sqrt{n}$ subproblems.

Wang and Tsin presented another optimal algorithm for finding the convex hull of a set of n points in 2 dimensions [WT87]. They give an algorithm for triangulating the set of points which explicitly builds the (upper) convex hull of the set. Their strategy is to partition the problem into $\sqrt{n}$ subproblems of size $\sqrt{n}$ and solve the resulting problems recursively. Each of the subproblems is triangulated and its (upper) convex hull is found. The bridges for each pair of adjacent subproblems is computed via a binary search. Each polygon formed by a bridge edge and its two chains is of a simple nature (one concave, one convex chain) which is easily triangulated.

Recently, using O(n/log n) processors, Goodrich presented an (optimal) O(log n) algorithm for determining the convex hull of n points which are sorted along some axis [G87].
In 3 dimensions, the convex hull of a set of points can be determined in $O(\log^3 n)$ time as was demonstrated in [C80, ACGDY85]. Furthermore, all 3-dimensional maxima can be determined in O(log n loglog n) time [AG86a]. The latter problem can be solved with high probability in time O(log n) [RS87]; i.e., the probability that the time is O(log n) approaches 1, for n approaching infinity.

As in the sequential model of computation several problems can be solved efficiently in parallel once the convex hull is available. Such problems include: deciding whether a point set is convex, testing whether two sets are linearly separable, finding the smallest enclosing box for a set of points, determining the diameter, supporting etc. We refer the reader to the literature discussed above.

3. Intersection Problems

Another class of geometric problems which has been thoroughly studied under the sequential model of computation deals with intersection problems for geometric objects (e.g., points, lines, rectangles, polygons). There exist two major classes of intersection problems: *intersection detection problems* are

concerned with detecting whether there exists an intersection between any two elements of a given set of objects (yes/no answer), *intersection determination problems* consist of computing the intersection (e.g., reporting all intersection points between any two of several polygons).

For parallel models of computation, nearly all of the proposed algorithms solve intersection detection problems only; the major obstacle to solving intersection determination problems on parallel computers is the potential size of the output which may cause bottlenecks in storing and/or reporting the result. (Thus current parallel algorithms for intersection determination problems typically report only the size of the result.)

The intersection problems studied for parallel machine models are the following:

Line Segment Intersection Detection

detect whether any two of n given line segments intersect

Polygon Intersection Detection

detect whether two polygons (with a total of n edges) intersect

Area of Intersection for Iso-oriented Rectangles

determine the area of intersection of n iso-oriented rectangles

Iso-oriented Line Segments Intersection Counting

count the number of intersections between n horizontal or vertical line segments

For the mesh-of-processor architecture, independently, Miller and Stout [MS86], [MS87], and Jeong and Lee [JL87a], [JL87b] studied the problem of detecting whether any two of n given line segments (in the plane) intersect. The solutions are quite similar and obtain an optimal (for the mesh) $O(\sqrt{n})$ time complexity (using n processors). To achieve this bound, the set of line segments is split, by vertical lines, into a constant number of subsets of equal size for which the problem is recursively solved in parallel. The problem arising is that portions of line segments may be contained in several subsets which may increase the problem size during the recursive calls (exponentially, in the worst case). To solve this, they observe that the portions of these line segments contained in one subset can be ordered (i.e., sorted by y-coordinate). Detecting intersection among these as well as with the remaining line segments, is then easily achieved; in case no intersection occurs, these line segments are eliminated from any further consideration.

Atallah and Goodrich [AG86a] studied the line segment intersection detection problem for the PRAM model. Their solutions are based on a parallel implementation of the well known (sequential) plane sweep technique [LP84]. For this intersection problem, they obtain $O(\log^2 n)$ and O(log n log log n) time algorithms for the CREW (of size n) and CRCW PRAM (of size n log n), respectively.

The above algorithms lead to a variety of algorithms of the same time complexity, in particular for detecting whether any of several given polygons with a total number of n edges intersect. They also solve the problem of counting the number of intersections between n horizontal or vertical line segments in

O(log n loglog n) time [AG86a]. Other problems solved using this technique are discussed in the respective sections.

The line intersection detection problem was also studied by Chazelle [C84] for the 1-dimensional array of processors. His algorithm has a time complexity of O(n) for n processors and is based on the same "fold-over" strategy used for convex hull determination (see Section 2).

Lu, Varman [LV86] studied the problem of determining the area of intersection of n iso-oriented rectangles. For the mesh-of-processors of size n they obtain an $O(\sqrt{n})$ time solution which is based on splitting the set of rectangles into vertical slabs such that each slab contains one vertical edge of a rectangle. Within each slab the problem is solved directly (in parallel) and then the results are merged.

4 Proximity and Geometric Searching Problems

Proximity problems arise in a wide field of application such as spatial data analysis, image processing, CAD, etc. Let $S=\{s_1, ...,s_n\}$ be a set of n objects and let $d(s_i,s_j)$ denote a distance metric between objects s_i and s_j. Some typical distance-related problems which have been extensively studied for the sequential model of computation are:

Nearest Neighbor Problem for a given object s (not necessarily contained in S) find its nearest neighbor in S,

All Nearest Neighbor Problem for each $s_i \varepsilon S$ find its closest neighbor $s_j \varepsilon S-\{s_i\}$, and

Closest Pair Problem find the pair $(s_i,s_j) \varepsilon S$ which minimizes $d(s_i,s_j)$ over all index pairs i,j

Another related problem, referred to as the *minimum distance problem*, is to find for two given sets S_1 and S_2 of objects (e.g., arbitrary or convex point sets) the pair $(s_i,s_j) \varepsilon S_1 x S_2$ which minimizes $d(s_i,s_j)$.

In [MS84] and [MS86], Miller and Stout study the all nearest neighbor problem for sets of n points in the plane and derive an $O(\sqrt{n})$ upper bound for the mesh-of-processors. The main technique applied in this solution is to split the problem by four vertical lines into five subsets of equal size and recursively solve each subproblem in parallel. Then the same procedure is again executed, this time, however, using four horizontal lines. Now, every point has assigned to it the nearest neighbor with respect to its region in the 5x5 rectangular grid determined by the horizontal and vertical lines. It is easy to prove that for every region there are at most 8 points whose nearest neighbor is outside the regions and that these points are closer to one of the corner points of the region than to the nearest neighbor determined so far. These at most 25*8 points are then simply broadcast to all points.

For the 1-dimensional array of processors, Chazelle [C84] described an O(n) time algorithm for computing all nearest neighbors; it is based on the same "fold-over" strategy used for convex hull determination (see Section 2) and, essentially, compares all pairs of points.

Atallah and Goodrich [AG85] study the closest pair problem for the PRAM. They split the point set into $\sqrt{n}$ sets, of size $\sqrt{n}$, each, using vertical cut lines. For each set, the closest pair is recursively computed in parallel. Let D denote the distance between the closest of these $\sqrt{n}$ pairs. The problem is to reduce the number of vertical cut lines with distance closer than D. For this, adjacent slabs which are too small are merged first, using a similar divide-and-conquer technique (this time using horizontal cut-lines). Subsequently, these solutions are combined. The total time complexity of their algorithm, using a PRAM of size O(n), is O(log n log log n).

For the related minimum distance problem for two convex sets whose boundaries are convex polygons (the sequential time complexity of this problem is O(log n)), Atallah and Goodrich described an $O(k^{1+\varepsilon})$ time algorithm for a PRAM with $O(n^{1/k})$ processors (with arbitrary, selectable, k and arbitrarily small ε). For k=1, this yields a constant time algorithm on a linear size PRAM.
In [S87b], Stojmenovic' describes an O(log n) time hypercube solution for the same problem.

An interesting generalization of the above problems was considered by Boxer and Miller [BM87] who studied dynamic versions, where n points are moving in Euclidean space. Each coordinate of a moving point is given by a time-dependent polynomial, say of degree at most k. For such a system of moving points, they solve the dynamic one-to-all nearest neighbor problem; i.e., they compute for the time interval from zero to infinity the sequence of nearest neighbors of one of the points. The time complexity of their solution is $O(\log^2 n)$ on a PRAM consisting of $\lambda(n-1,2k)$ processors where $\lambda(n,k)$ is the number of pieces of the minimum of n polynomials of degree k (lower envelop); $\lambda(n,k)=\Omega(n)$ and $\lambda(n,k)=O(n \log^* n)$. The algorithm is based on an $O(\log^2 n)$ time parallel algorithm to compute the description of the minimum of n polynomials of degree k (i.e., the pieces of the minimum in sorted order).

The closest pair and minimum distance problems have also been studied for pixels of a digitized image of size $\sqrt{n}x\sqrt{n}$. The mesh-of-processor architecture of size n is optimally suited to such image processing tasks. For this model Miller and Stout [MS85] presented $O(\sqrt{n})$ time algorithms for (a) computing the minimum distance between sets of black pixels and (b) the maximum distance between two black pixels in each set. For the pyramid computer $O(\sqrt{n})$ and O(log n) time solutions to the closest pair problem for the set of black pixels have been developed by [D80] and [S87], respectively. [KE] presents O(log n) time *mesh-of-tree algorithms for the all nearest neighbor and minimum distance problem.*

Under the sequential model of computation, the nearest neighbor problem for sets of points in the plane can be solved in time O(log n) per query (with O(n log n) time preprocessing). Hence, interesting parallel solutions are hard to obtain; they should have a time complexity close to O(1).
Therefore, the computationally more expensive problem of finding for m query points their nearest neighbors among a set of n given sites has been considered; this problem will be referred to as the *multi-*

point nearest neighbor problem. For the remainder of this paper, we will assume, for simplicity, that m=n.

As in the sequential case, this problem has been solved via construction of the Voronoi diagram [LP84] for the set of sites. For the PRAM of size n, Aggarwal et al. [ACGDY85] have presented an $O(\log^2 n)$ time Voronoi diagram construction algorithm; for the mesh-of-processors an $O(\sqrt{n})$ time algorithm is described in [JL87a] and [JL87b]. Both algorithms resemble the well known sequential divide-and-conquer algorithm [SH75]. First, the point set is divided into two linearly separable subsets and, recursively, the two Voronoi diagrams are constructed for these sets. The main step is to construct the "line" at which the two Voronoi diagrams are merged. Either algorithm identifies, in parallel, those edges in the Voronoi diagrams which are intersecting the merge line; subsequently, this merge line is actually computed.

Besides its application to the nearest neighbor problem, the Voronoi diagram is an interesting structure of its own, has a lot of other applications, and generalizes to other metrics, higher orders, higher dimensions and for types of objects other than points (see e.g. [LP84]).

Assume that the Voronoi diagram has been computed as described. To solve the multi-point nearest neighbor problem, for each query point q, the Voronoi polygon containing q is to be located. This problem is a particular instance of the more general multi-point location problem: Given a planar straight-line graph defining a subdivision of the plane, determine for each query point q from a given set the region of the subdivision q falls into.
For the mesh-of-processors, the multi-point location problem (and, hence, the multi-point nearest neighbor problem) can be solved in time $O(\sqrt{n})$ [JL87a], [JL87b]. For the PRAM, Dadoun and Kirkpatrick [DK87] present a very interesting solution. They compute for any given planar subdivision, on a PRAM of size n, the well known subdivision hierarchy [Ki83] in time $O(\log n \log^* n)$ (or $O(\log n)$ expected time). The general structure of the algorithm and the final result is the same as in the sequential case [Ki83]. Once the subdivision hierarchy has been constructed, every processor can locate one point in $O(\log n)$ sequential time; hence, their methods results in a $O(\log n \log^* n)$ (or $O(\log n)$ expected) time solution to the multi-point location and multi-point nearest neighbor problem.

5. Visibility Problems

Problems dealing with the visibility of objects have been studied extensively in computer graphics, robotics and computational geometry. Different aspects of visibility problems have been considered.
Let P be a set of segments or polygonal region possibly containing holes. Two points p and q are mutually visible if the line-segment connecting them does not intersect any edges of P or of the holes (if present).
Compute Visibility Of Segments

For a set of line-segments in the plane or in 3-space, determine which (portions of) segments are visible from a given point.

Compute Visibility Between Segments

For a given set of line-segments compute all pairs of segments which are mutually visible, where two segments see each other if there exists a pair of mutually visible points, located on different segments.

This problem arises e.g. in VLSI-design for the particular instance of orthogonal line-segments.

Compute Visibility Polygon

For a polygon with or without holes determine the portion of the polygon that is visible from a given view-point p. This visibility problem arises in computer graphics as well as in robotics (e.g., in planning a shortest path for a polygonal object among polygonal obstacles). A view-point may be located either inside or outside of the polygonal region. Both of these problems are referred to as *perspective visibility problems*. The *parallel visibility problem* arises if the view-point is located at infinity. An alternate way of seeing this problem is to determine which regions of the polygonal region(s) are illuminated by a light-source located at infinity. Notice that the two models are equivalent; i.e., one can transform one problem instance into the other by applying a simple geometric transformation.

The Compute Visibility Of Segments problem for n vertical line-segments can be solved in O(log n) time on a mesh-of-trees architecture, as shown by Lodi and Pagli [LP86]. Their paper gives a detailed low-level implementation as well as an area-time (AT^2) lower bound of $\Omega(n^2 \log^2 n)$ for the visibility problem.

For the linear processor array Asano and Umeo [AU87] gave an O(n) algorithm to compute the visibility polygon from a point (or from infinity) for a polygonal region possibly containing holes. They classify each vertex v to be of one of four types, using the turns between the viewpoint, v, and the vertices adjacent to v (in the polygonal order). Based on this, a 2-stage algorithm was designed, where the first stage consists of loading the array with the vertices and determining their type, while the second stage transmits the edges checking whether there exist edges which intersect the rays from the view-point. By combining vertex-type and intersection information the visibility problem is then solved.

On the mesh-of-processors work has been done, both, in image space (i.e., for digitized objects) as well as in object space.

For a digitized set of objects stored on a mesh-of-processors of size n, the visibility polygon in the parallel model can be determined in $O(\sqrt{n})$ time [DHSS87]. They split the image into strips parallel to the direction of visibility. Through each strip, in parallel, a hole is is sent in the direction of visibility which represents the portion of the strip that is still visible from the light source. It must be ensured that at any time both (1) the size of messages sent by each processor, as well as (2) the number of messages sent or received by a processing element are bounded by a constant.

Using message passing by layers starting at the point of visibility and updating visibility wedges instead of strips, the perspective model of visibility has been solved in $O(\sqrt{n})$ time in [H88].

For polygons, possibly with holes, in object representation (i.e., each edge of the polygon or a hole is stored in one arbitrary processor) $O(\sqrt{n})$ time algorithms for both models of visibility have been presented in [D88]. For the point visibility problem (and similarly for parallel visibility), the polygon is split by rays, emanating from the view point, into sectors of equal number of vertices. For each sector the problem is solved recursively in parallel, and then the results (i.e., the portions of the visibility polygon in each sector) are merged. The merge step is obvious; however, the split step creates problems since edges that intersect a split ray have to be duplicated (one part for each adjacent sector). This may create O(n) new edges in each recursion step which would result in an over-flow of the mesh. The solution consists of several steps after each split phase which delete edges or parts of edges which are found to be invisible and, if this is not sufficient, determine a better split ray that intersects fewer edges.

As mentioned earlier, Atallah and Goodrich implemented the plane sweep technique in parallel [AG86a]. One of the applications of this technique is an O(log n log log n) algorithm for the problem Compute Visibility Of Segments on an n processor PRAM architecture.

Very recently, Dehne and Pham [DP88] presented, for the hypercube architecture, $O(\log^2 n)$ time algorithms for solving the visibility problems, in image space, in the parallel and perspective model of computation by reducing it to a generalized partial sum operation and a tree partial sum operation, respectively.

6. Decomposition Problems

Decomposition problems have been studied in pattern recognition, image analysis and robotics. The problem is to decompose an object into simpler often more meaningful components. A common task is to decompose an object into convex parts. Decompositions are obtained by adding segments inside the object. Several types of decompositions arise: *partitionings*, i.e., decompositions where no two components may overlap, and *coverings* where components may overlap. A decomposition may introduce new vertices, called *Steiner-points*, or may consist entirely of segments, called diagonals, which join only existing vertices. An example of the former category is the *trapezoidal partitioning*, where an object given as a simple polygon is to be partitioned into trapezoids such that each newly inserted segment is horizontal and incident to a vertex of the polygon. An example of a decomposition without additional vertices is the *triangulation*; i.e., the task of partitioning a simple polygon into triangles by inserting only diagonals. A triangulation of a set of points is a graph whose vertices are the points and whose edges are a maximal number of diagonals making each bounded face a triangle.

For a survey of sequential decomposition techniques the reader is referred to [KS85]. This reference deals in particular with decompositions in which some criterion is to be minimized; e.g., finding a minimum weight triangulation (where the weight is the total length of all diagonals added), or minimizing the number of convex polygons resulting from a partitioning of a simple polygon into convex pieces, etc.

The following problems have been studied in the parallel models of computation.

Trapezoidal Partitioning

Find a partitioning of the input polygon into trapezoids

Triangulate Line Segments

Find a triangulation of the input line segments which are typically assumed to intersect at most at their endpoints to include as a special case the triangulation of a polygon

Triangulate Point Set

Find a triangulation of the input point set in 2 or higher dimension

For the linear processor array, trapezoidal partitioning and triangulation of polygonal regions [AU77] and triangulation of sets of points [C84] can be performed in linear time. Trapezoidal decomposition on a mesh-of-processors of size n can be performed in $O(\sqrt{n})$ time as shown by Jeong and Lee [JL87]. Their algorithm is based on the $O(\sqrt{n})$ solution to the multi-point location problem discussed in Section 4.

As mentioned above, Atallah and Goodrich developed a parallel implementation of the sequential plane-sweep technique for the PRAM. They showed that, based on this parallel plane sweep, the problems Trapezoidal Partitioning of Polygon and Triangulate Line Segments can be solved in O(log n loglogn) time using O(n) processors and O(n log n) space. If space is restricted to O(n) then the time bound is $O(\log^2 n)$ [AG86a]. According to Yap [Y87] the O(log n loglog n) bound has been improved to O(log n) by Atallah, Cole and Goodrich. Optimal, randomized O(log n) time, parallel algorithms for triangulation and trapezoidal partitioning were given in [RS87].

An optimal PRAM algorithm for triangulating a set of points in the plane was presented by Merks [M86]. The algorithm is optimal in that it takes O(log n) time using n processors. Again, the divide and conquer paradigm of splitting the point set into $\sqrt{n}$ subproblems is applied. They first show how to reduce the problem into the simpler one of triangulating a set of points located in a triangle. Then they show how to solve this simpler problem instance. The final merge is to connect these subsolutions.

More recently, Wang and Tsin gave another algorithm for this problem [WT87]. Their approach was discussed in the section on convex hull problems and we refer the reader to that section.

ElGindy gave an $O(f(d) \log^2 n)$ time PRAM algorithm for triangulating points in d-dimensional space, where $f(d) = d^4 \log(4^d/(4^d-1))$. The number of processors used by the algorithm is O(n/log n). The algorithm is an efficient parallel implementation of the sequential algorithm proposed in [AE86]. For fixed dimensions the speed-up (product of time and number of processors) is optimal. It remains open whether optimal speed-up can be achieved using O(n) processors for dimensions d>2.

References

[ACGDY85] Aggarwal, A., B. Chazelle, L. Guibas, C. O. Dunlaing, and C. Yap, "Parallel computational geometry", *Proc. IEEE Symp. on Found. of Computer Science*, Portland, Oregon, Oct. 1985, pp. 468-477

[A82] Akl, S., "A constant-time parallel algorithm for computing convex hulls", *BIT* 22, 1982, pp. 130-134

[A84] Akl, S., "Optimal parallel algorithms for computing convex hulls and for sorting", *Computing* 33,1 11, 1984, pp. 1-11

[AG85] Atallah, M. J. and M. T. Goodrich, "Efficient parallel solutions to geometric problems" , *Proc. 1985 Int'l Conference on Parallel Processing*, Aug. 20-23, 1985, pp. 411-417

[AG86a] Atallah, M. J. and M. T. Goodrich, "Efficient plane sweeping in parallel (Preliminary Version)", *Proc. Second ACM SIGGRAPH Symposium on Computational Geometry*, Yorktown Heights, NY, June 2-4, 1986, pp. 216-225

[AG86b] Atallah, M. J. and M. T. Goodrich, "Parallel algorithms for some functions of two convex polygons", *Proc. 24th Allerton Conference on Comm., Control and Comput.*, Monticello, Ill., Oct. 1-3, 1986, pp.758-767

[AU87] Asano, T. and H. Umeo, "Systolic algorithms for computing the visibility polygon and triangulation of a polygonal region", *Proc. Int'l Workshop on Parallel Algorithms and Architectures*, Suhl, GDR, May 25-30, 1987, pp.77-85

[AE86] Avis, D. and H. ElGindy, "Triangulating simplicial points set in space: extended abstract", *Proc. ACM Symp. on Computational Geometry*, York Town Heights, 1986, pp. 133-142

[BM87] Boxer, L., R. Miller, "Parallel dynamic computational geometry", Tech. Rep. 87-11, Dept. of Computer Science, SUNY at Buffalo, Buffalo, NY, Aug. 1987

[C84] Chazelle, B. M., "Computational geometry on a systolic chip", *IEEE Trans. on Computers*, Vol. C-33, No. 9, Sept. 1984, pp.774-785

[CCL87] Chen, G.H., M.-S. Chern, and R.C.T. Lee, "A new systolic architecture for convex hull and half-plane intersection problems", *BIT* 27, 1987, pp. 141-147

[C81] Chow, A., "Parallel algorithms for geometric problems", *PhD. thesis*, Dept. of Computer Science, Univ. of Illinois, Urbana-Champaign, 1980

[CM81] Chazelle, B. M., L. M. Monier, "A model of computation for VLSI with related complexity results", *Proc. 13th ACM Symp. on Theory of Comp.*, 1981

[DK87] Dadoun, N., D. G. Kirkpatrick, "Parallel processing for efficient subdivision search", *Proc. Third ACM SIGGRAPH Symposium on Computational Geometry, Waterloo*, 1987, pp. 205-214

[D85b] Dehne, F., "Solving geometric problems on mesh-connected and one-dimensional processor arrays", *Proc. 11th Int'l Workshop on Graph-theoretic Concepts in Computer Science (WG'85)*, June 18-21, 1985, Schloß Schwanberg, Würzburg, W.-Germany, Trauner Verlag 1985, pp. 43-59

[D85c] Dehne, F., "A one dimensional systolic array for the largest empty rectangle problem", *Proc. 23rd Annual Allerton Conference on Comm., Control and Comput.*, Monticello, Ill., Oct. 2-4, 1985, pp. 518-524

[D86b] Dehne, F., "O(n1/2) Algorithms for the maximal elements and ECDF searching problem on a mesh-connected parallel computer", *Info. Proc. Let.* 22 , 1986, pp. 303-306, May 1986

[D87a] Dehne, F., "Computational geometry and VLSI", *1987 CompEuro Conference*, Hamburg, FRG, May 11-15, 1987, pp. 870-875

[DHSS87] Dehne, F., A.-L. Hassenclover, J.-R. Sack, and N. Santoro, "Parallel visibility on a mesh-connected parallel computer", *Int. Conference on Parallel Processing and Applications*, L'Aquila, Italy, September 23-25, 1987, pp. 173-180

[DS87c] Dehne, F. and I. Stojmenovic, "An optimal parallel solution to the ECDF searching problem for higher dimensions on a mesh-of-processors", *Proc. 25th Annual Allerton Conference on Comm., Control and Comput.*, Monticello, Ill., Sept.30-Oct.2, 1987, pp. 660-661

[DP88] Dehne, F. and Q. T. Pham, "Visibility algorithms for binary images on the hypercube and perfect-shuffle computer", to appear in *Proc. IFIP Working Conference on Parallel Processing*, Pisa, Italy, April 25-27, 1988

[DSS87] Dehne, F., J.-R. Sack, and N. Santoro, "Computing on a systolic screen: hulls, contours and applications", *Conference on Parallel Architectures and Languages Europe*, Eindhoven, The Netherlands, June 15-19, 1987

[D80] Dyer, C. R., "A fast parallel algorithm for the closest pair problem", *Info. Proc. Let.* 11:1, 1980, pp. 49-52

[E86] ElGindy, H., "A parallel algorithm for triangulating simplicial point sets in space with optimal speed-up", *Proc. 24th Annual Allerton Conference on Comm., Control and Comput.*, Monticello, Ill., Oct. 1-3, 1986

[G87] Goodrich, M., "Finding the convex hull of a sorted point set in parallel", *Info. Proc. Let.* 26 , 1987/88, pp. 173-179

[H88] Hassenklover, A.-L. "Visibility for binary images on a mesh-connected computer", Master's thesis, School of Computer Science, Carleton University, April 1988

[Ki83] Kirkpatrick, D. G., "Optimal search in planar subdivisions", *SIAM Journal of Computing* 12:1,1983, pp. 28-35

[JL87a] Jeong, C. S. and D. T. Lee, "Parallel geometric algorithms on mesh-connected computers", *Proc. IEEE Fall Joint Computer Conference*, 1987

[JL87b] Jeong, C. S. and D. T. Lee, "Parallel geometric algorithms on mesh-connected computers", Tech. Rep. 87-02-FC-01 (Revised), Technological Inst., Northwestern Univ., Evanston, Ill, 1987

[KS85] Keil, J. M. and J.-R. Sack, "Minimum decomposition of polygonal objects", in Computational Geometry, ed. G.T. Toussaint, North Holland, Amsterdam, The Netherlands, 1985

[KE86] Kumar, V. and M. Eshaghian, "Parallel geometric algorithms for digitized pictures on mesh of trees", *Proc. 1986 Int'l Conference on Parallel Processing*, St. Charles, Ill., Aug. 19-22, 1986, pp. 270-273

[KY85] Kozen, D. and C.K. Yap, "Algebraic cell decomposition in NC", *Proc. IEEE Symp. on Found. of Computer Science*, Portland, Oregon, Oct. 1985, pp. 515-521

[LP84] Lee, D.T. and F. P. Preparata, "Computational geometry - a survey", *IEEE Trans. on Computers*, Vol. C-33, No. 12, Dec. 1984, pp. 1072-1101

[LP86] Lodi, E. and L. Pagli, "A VLSI solution to the vertical segment visibility problem", *IEEE Trans. on Computers, Vol. C-35, No. 10, Oct. 1986, pp. 923-928.*

[Lu86] Lu, M. "Constructing the Voronoi diagram on a mesh-connected computer", *Proc. 1986 Int'l Conference on Parallel Processing*, St. Charles, Ill., Aug. 19-22, 1986, pp. 806-811

[LV85] Lu, M. and P. Varman, "Solving geometric proximity problems on mesh-connected computers", *Proc. 1985 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, pp.249-255, Nov. 1985

[LV86] Lu, M. and P. Varman, "Mesh-connected computer algorithms for rectangle intersection problems", *Proc. 1986 Int'l Conference on Parallel Processing*, St. Charles, Ill., Aug. 19-22, 1986, pp. 301-307

[M86] Merks, E., "An optimal parallel algorithm for triangulating a set of points in the plane", *Int'l Journal on Parallel Programming*, Vol. 15, No. 5, 1986, pp. 399-411

[MC80] Mead, C. A. and L. A. Conway, "Introduction to VLSI Systems", Addison-Wesley, Reading, MA, 1980

[MM87] Miller, R. and S. E. Miller, "Using the hypercube to determine geometric properties of digitized pictures", *Proc. Int'l Conf. Parallel Processing*, 1987, pp. 638-640

[MS84] Miller, R. and Q. F. Stout, "Computational geometry on a mesh-connected computer", *Proc. Int.l Conf. on Parallel Processing*, 1984, pp. 66-73

[MS85] Miller, R. and Q. F. Stout, "Geometric algorithms for digitized pictures on a mesh-connected computer", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 2, March 1985, pp 216-228

[MS86] Miller, R. and Q. F. Stout, "Mesh computer algorithms for computational geometry", Tech. Rept. 86-18, Department of Computer Science, Univ. of Buffalo, 1986

[MS87] Miller, R. and Q. F. Stout, "Mesh computer algorithms for line segments and simple polygons", *Proc. Int.l Conf. on Parallel Processing*, Aug. 1987, pp. 282-285

[NMB81] Nath, D., S. N. Maheshwari, and P.C. P. Bhatt, "Parallel algorithms for convex hull determination in two dimensions", *Conf. Analysis Problem Classes and Programming for Parallel Computing*, 1981, pp. 358-372

[PS85] Preparata, F. P. and M. I. Shamos, "Computational Geometry, An Introduction", Springer Verlag Berlin, Heidelberg, New York, Tokyo1985

[PV79] Preparata, F. P. and J. E. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation", *Proc. 20th IEEE Symp. on Found. of Comp. Sci.*, 1979, pp.140-147

[RS87] Reif, J. and S. Sen, "Optimal randomized parallel algorithms for computational geometry", *Proc. Int'l Conf. Parallel Processing*, 1987, pp. 270-277

[SH75] Shamos, M. I., D. Hoey, "Closest point problems", *Proc. 16th Symp. on Foundations of Computer Science*, 1975, pp. 152-162

[St85] Stout, Q. F., "Pyramid computer solutions for the closest pair problem", *J. Algorithms*, No. 6, 1985, pp. 200-212

[S87b] Stojmenovic´, I. "Parallel computational geometry", Tech. Rept. CS-87-176, Computer Science, Washington State University, 1987

[TK77] Thompson, C. D. and H. T. Kung, "Sorting on a mesh-connected parallel computer", *C. ACM*, Vol. 20, No. 4, April 1977, pp. 263-271

[UL84] Ullman, J. D., "Computational Aspects of VLSI", Principles of Computer Science Series, Computer Science Press, 1984

[W85] Wagener, H., "Parallel computational geometry using polygonal order", PhD. thesis Technical University of Berlin, FRG, 1985

[WT87] Wang, C. A. and Y. H. Tsin, "An O(log n) time parallel algorithm for triangulating a set of points in the plane, *IPL* 25, 1987, pp. 55-60

[Y87] Yap, C.K., "What can be parallelized in computational geometry", *Proc. International Workshop on Parallel Algorithms and Architectures*, Suhl, GDR, May 25-30, 1987, pp. 184-195