JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 3, 492–507 (1986)

Efficient Parallel Solutions to Some Geometric Problems

MIKHAIL J. ATALLAH* AND MICHAEL T. GOODRICH

Department of Computer Sciences, Purdue University, West Lafayette, Indiana 47907

Received May 13, 1985

This paper presents new algorithms for solving some geometric problems on a shared memory parallel computer, where concurrent reads are allowed but no two processors can simultaneously attempt to write in the same memory location. The algorithms are quite different from known sequential algorithms, and are based on the use of a new parallel divide-and-conquer technique. One of our results is an $O(\log n)$ time, $O(n)$ processor algorithm for the convex hull problem. Another result is an $O(\log n \log \log n)$ time, $O(n)$ processor algorithm for the problem of selecting a closest pair of points among n input points. © 1986 Academic Press, Inc.

1. INTRODUCTION

Since they involve asking basic questions about sets of points, lines, polygons, etc., geometric problems arise often in many applications (see [14] for examples). We are interested in finding parallel algorithms solving some of these problems which are efficient in terms of both their running time and the number of processors used. Efficient sequential algorithms for solving geometric problems often use the divide-and-conquer paradigm: to solve a problem of size n solve two subproblems of size $n/2$, and then "marry" the results of these two recursive calls. Unfortunately, trying to "parallelize" sequential algorithms based on this paradigm often yields suboptimal parallel solutions. Such is the case for the convex hull and the closest-pair problems, for example. Indeed, the efficient parallel algorithms we give for solving these problems turn out to be quite different from the known sequential algorithms.

Throughout this paper, the computational model used is the synchronous parallel model in which processors share a common memory in which concurrent reads are allowed, but no two processors can simultaneously write to the

*This research was supported by the Office of Naval Research under Contract N00014-84-K-0502 and the National Science Foundation under Grant DCR-84-51393, with matching funds from A.T.&T.

same memory location. We henceforth refer to this model as the CREW PRAM (Concurrent Read Exclusive Write Parallel RAM), as it is commonly called. Using this model of parallel computation, we are interested in achieving the highest speedup possible using only $O(n)$ processors (this restriction on the numbers of processors is crucial, since the problems we consider could trivially be solved in logarithmic time if the number of processors used were of no concern, e.g., $O(n^2)$).

The technique which is common to all of our algorithms is a new parallel version of divide-and-conquer. The main idea is to divide the problem into many subproblems (e.g., $\sqrt{n}$), instead of just two; solve all the subproblems recursively in parallel; and, when the parallel recursive call returns, marry all the subproblem solutions quickly in parallel. As one may suspect, performing the marry step quickly in parallel is the most difficult aspect of using this technique, and, as we demonstrate, oftentimes requires new insights into the structure of the problem being solved. In Section 2 we use this technique to design an $O(\log n)$ time, $O(n)$ processor parallel algorithm for constructing planar convex hulls and related problems. (We have recently learned that the convex hull result was independently discovered by Aggarwal *et al.* [1].) This improves on the previous parallel algorithm for constructing planar convex hulls on a CREW PRAM, which ran in $O(\log^2 n)$ time using $O(n)$ processors, given by Chow in [9]. Our algorithm is optimal with respect to both the time and the number of processors used, since this problem has an $\Omega(n \log n)$ time sequential lower bound [23]; hence, an obvious $\Omega(\log n)$ time lower bound for the CREW PRAM computational model when using $O(n)$ processors. Another problem for which we use the parallel divide-and-conquer technique is that of finding the closest pair among a set of n input points, which we present in Section 3. Our algorithm for this problem runs in $O(\log n \log \log n)$ time using $O(n)$ processors.

In some of our algorithms we make use of the fact that the parallel prefix of a sequence of n integers can be computed in $O(\log n)$ time using $O(n/\log n)$ processors [12, 13]. Recall that in the parallel prefix problem we are given an array of integers $(a_1, a_2, \ldots, a_n)$ and wish to compute all the partial sums $s_k = \sum_{i=1}^{k} a_i$. We also make use of the known result that, on this model of parallel computation, n objects can be sorted in $O(\log n)$ time using $O(n)$ processors [2, 15]. Unfortunately, the constant involved in the time complexity of these algorithms is very large. This does not mean that our algorithms are impractical, however, for one can easily substitute a more practical sorting algorithm, such as that presented in [6, 22], at any point where sorting is required in our algorithms. Using the parallel merge–sort algorithm of [6, 22] introduces an additional factor of $\log \log n$ in our time complexity bounds, but significantly reduces the constant term. Thus, our algorithms are of both theoretical and practical interest.

To simplify the exposition, we assume that no three points in the input set are collinear and that the points have distinct x (resp. y) coordinates (our results can easily be modified for the general case).

2. Convex Hull

Given n points in the plane, the planar convex hull problem is that of finding which of these points belong to the perimeter of the smallest convex region (a polygon) containing all n points. This problem has applications in many fields, including computer graphics, computer vision, and statistics [14]. As mentioned earlier, the convex hull problem has an $\Omega(n \log n)$ time sequential lower bound [23], and this bound is achievable [11, 18, 19].

Several authors have addressed the question of finding parallel solutions to this problem. Chazelle [8] shows how to solve the problem on a linear array of processors in a systolic fashion in $O(n)$ time. Miller and Stout, in Ref. [16], present an $O(\sqrt{n})$ time solution on an n-node mesh-connected computer. Although both of these algorithms are optimal for the computational models for which they were designed, implementing them on a CREW PRAM would lead to suboptimal algorithms. The only known previous parallel algorithm solving this problem on a CREW PRAM is due to Chow [9], and runs in $O(\log^2 n)$ time using $O(n)$ processors. In this section we present a new parallel algorithm which solves the planar convex hull problem in $O(\log n)$ time on a CREW PRAM with $O(n)$ processors. As mentioned earlier, our algorithm is optimal (to within a constant factor).

We first present some definitions and observations. Let R be a set of points in the plane. We denote a clockwise listing of the points which belong to the convex hull of R by CH(R). Let u and v be the points of R with the smallest and largest x-coordinates, respectively. Clearly, u and v are both in CH(R). They divide CH(R) into two sets: an upper hull, consisting of points from u to v, inclusive, in the clockwise listing of CH(R), and a lower hull, consisting of points from v to u, inclusive. We denote a clockwise listing of the points in the upper hull of R by UH(R), and a similar listing of the points in the lower hull by LH(R). Given a set S of n points in the plane the following algorithm will compute CH(S).

Algorithm CH

Input. A set S of n points in the plane.

Output. The list CH(S). That is, the points of the convex hull of S listed in clockwise order.

Method. The main idea of our algorithm is to divide the problem into $\sqrt{n}$ subproblems of size $\sqrt{n}$ each, solve the subproblems recursively in parallel, and combine the solutions to the subproblems quickly (that is, in $O(\log n)$ time) and with a linear number of processors.

Step 1. Sort the n points by x-coordinate, and partition S into sets R_1, $R_2, \ldots, R_{\sqrt{n}}$, each of size $\sqrt{n}$, divided by vertical cut-lines, such that R_i is left of R_j if $i < j$ (see Fig. 1).

Step 2. Recursively solve the convex hull problem for each R_i,

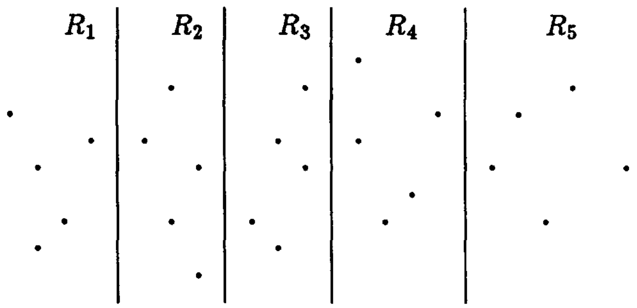

FIG. 1. A partitioning of S into $\sqrt{n}$ sets, an example with $n = 25$.

$i \in \{1, 2, \ldots, \sqrt{n}\}$, in parallel. After this parallel recursive call returns we will have CH(R_i) for each R_i.

Step 3. Find the convex hull of S by computing the convex hull of the union of the $\sqrt{n}$ convex polygons CH(R_1), . . . , CH($R_{\sqrt{n}}$). This is done using algorithm COMBINE, which will be described later in this section.

End of Algorithm CH

THEOREM. *Algorithm* CH *finds the convex hull of a set of n points in the plane in* $O(\log n)$ *time on a* CREW PRAM *with* $O(n)$ *processors.*

Proof. We give this proof assuming that algorithm COMBINE (used in Step 3) is correct and takes $O(\log n)$ time and $O(n)$ processors. (This will be justified once we describe algorithm COMBINE later in this section.) That Step 1 can be done in $O(\log n)$ time and $O(n)$ processors follows from the results of [2, 15]. Thus the running time, $T(n)$, of the algorithm can be expressed in the recurrence relation $T(n) = T(\sqrt{n}) + b \log n$, where b is some constant, which has solution $T(n) = O(\log n)$. The number of processors needed, $P(n)$, satisfies the recurrence $P(n) = \max\{\sqrt{n}\, P(\sqrt{n}), cn\}$, where c is a constant, which has solution $P(n) = O(n)$. This completes the proof, subject to the already stated assumption about Step 3 and algorithm COMBINE (yet to be described). ■

The rest of this section deals with the problem of implementing Step 3 of algorithm CH in time $O(\log n)$ and with $O(n)$ processors. This is done by using algorithm COMBINE, described below. For convenience, we choose to describe the algorithm for the problem of computing the upper hull, since that of computing the lower hull is symmetrical. In the algorithm description, when we talk about the *upper common tangent* between CH(R_i) and CH(R_j), we mean the common tangent such that both CH(R_i) and CH(R_j) are below it. Also, when we say that a point p is "to the left" of another point q, we mean that the x-coordinate of p is less than that of q.

Algorithm COMBINE

Input. The collection of convex polygons $\mathrm{CH}(R_1)$, $\mathrm{CH}(R_2)$, . . . , $\mathrm{CH}(R_{\sqrt{n}})$. Recall that these input polygons are separated by vertical lines, and that none of them has more than $\sqrt{n}$ vertices. Also recall that $\mathrm{CH}(R_i)$ is to the left of $\mathrm{CH}(R_j)$ if $i < j$.

Output. The upper convex hull $\mathrm{UH}(S)$ of the vertices of the union of the $\mathrm{CH}(R_i)$'s.

Method. The main idea is to find, in parallel for each $\mathrm{CH}(R_i)$, which of its vertices are on $\mathrm{UH}(S)$. This is done by assigning $\sqrt{n}$ processors to each $\mathrm{CH}(R_i)$ and having each of these processors compute the upper common tangent between $\mathrm{CH}(R_i)$ and one of the other input polygons. The details follow.

Step 1. In parallel for each $i \in \{1, 2, \ldots, \sqrt{n}\}$ use $\sqrt{n}$ processors to find those points of $\mathrm{CH}(R_i)$ which belong to $\mathrm{UH}(S)$ by doing the following:

Step 1.1. Find the $\sqrt{n} - 1$ upper common tangents between $\mathrm{CH}(R_i)$ and the remaining $\sqrt{n} - 1$ other input polygons. Let $T_{i,j}$ denote the upper common tangent between $\mathrm{CH}(R_i)$ and $\mathrm{CH}(R_j)$, where $T_{i,j}$ is represented by its point of contact with $\mathrm{CH}(R_i)$ and its point of contact with $\mathrm{CH}(R_j)$. A tangent $T_{i,j}$ is easily computed in $O(\log n)$ time by one processor, using a binary-search technique due to Overmars and Van Leeuwen [17]. Therefore all of $T_{i,1}, \ldots, T_{i,\sqrt{n}}$ can be computed in $O(\log n)$ time by the $\sqrt{n}$ processors assigned to $\mathrm{CH}(R_i)$.

Step 1.2. Let V_i be the tangent with smallest slope in $\{T_{i,1}, \ldots, T_{i,i-1}\}$ (i.e., V_i is the smallest-slope tangent which "comes from the left" of $\mathrm{CH}(R_i)$), and let W_i be the tangent with largest slope in $\{T_{i,i+1}, \ldots, T_{i,\sqrt{n}}\}$ (i.e., W_i is the largest-slope tangent which "comes from the right" of $\mathrm{CH}(R_i)$). Let v_i be the point of contact of V_i with $\mathrm{CH}(R_i)$, and let w_i be the point of contact of W_i with $\mathrm{CH}(R_i)$. Both V_i and W_i can be found in $O(\log n)$ time by the $\sqrt{n}$ processors assigned to $\mathrm{CH}(R_i)$.

Step 1.3. Since neither V_i nor W_i can be vertical, they intersect and form an angle (with interior pointing upward). If this angle is less than 180° (as in Fig. 2), then none of the points of $\mathrm{CH}(R_i)$ belong to $\mathrm{UH}(S)$. Otherwise, (as in Fig. 3) all the points from v_i to w_i, inclusive, belong to $\mathrm{UH}(S)$.

Step 2. Step 1 has computed, for every $i \in \{1, \ldots, \sqrt{n}\}$, all the points of $\mathrm{CH}(R_i)$ which belong to $\mathrm{UH}(S)$ (possibly none). This step compresses each of these lists into one list to get $\mathrm{UH}(S)$. This can be done in $O(\log n)$ time and $O(n)$ processors (e.g., by using a parallel prefix computation).

End of Algorithm COMBINE

That COMBINE runs in time $O(\log n)$ and $O(n)$ processors should be clear from the comments made in the algorithm description. The correctness of

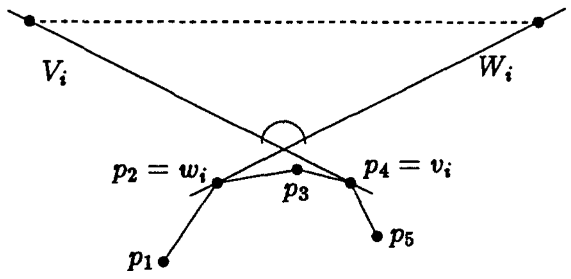

FIG. 2. An illustration of the case when none of CH(R_i)'s points are in UH(S) because V_i and W_i form an angle which is less than 180°.

COMBINE depends on the correctness of Step 1.3. The correctness of Step 1.3 for the case when the angle between V_i and W_i is less than 180°, depicted in Fig. 2, follows from the fact that in that case the straight-line segment joining the other endpoints of V_i and W_i (shown dashed in Fig. 2) is entirely above CH(R_i); hence, no vertex of CH(R_i) can belong to UH(S). The correctness of Step 1.3 for the case when the angle between V_i and W_i is greater than 180°, depicted in Fig. 3, follows from the fact that the points from all the other CH(R_j)'s are below V_i and W_i. This establishes the correctness of algorithm COMBINE.

Thus, we can construct the convex hull of n points in $O(\log n)$ time using $O(n)$ processors on a CREW PRAM. The convex hull problem is a fundamental problem in computational geometry and is used as a building block in many other geometric algorithms. For example, our algorithm can be used to find the common intersection of n half-planes in $O(\log n)$ time using $O(n)$ processors, by using a duality transformation of [7, 20]. It can also be used as a preprocessing step in conjunction with the algorithm of [10] for finding the diameter of a convex polygon to find a farthest pair of points in $O(\log n)$ time using $O(n)$ processors. The problem of finding a closest pair of points does not follow from the convex hull problem, however. We deal with the closest-pair problem in the next section.

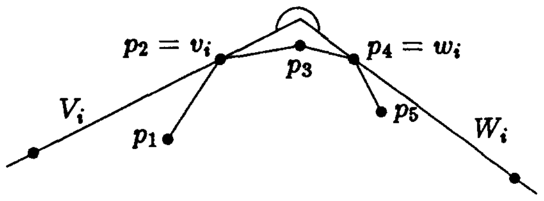

FIG. 3. The points p_2, p_3, and p_4, are in UH(S), because V_i and W_i form an angle which is at least 180°.

3. Closest Pair

Given n points in the plane, the closest-pair problem is that of choosing two points that are closest (i.e., the distance between them is smallest). This problem has applications in answering basic proximity questions of sets of objects, such as monitoring airplanes in air-traffic control. We are not aware of any previous work done in finding parallel solutions to this problem. A trivial $O(\log n)$ time parallel algorithm exists, but it requires a quadratic number of processors. Here we are investigating what time bound can be achieved with only $O(n)$ processors. Parallelizing what seems to be the most promising sequential algorithm [4, 5] on $O(n)$ processors only leads to an $O(\log^2 n)$ time algorithm. Applying a divide-and-conquer technique similar to the one we used in the convex hull problem, we show how to solve the closest-pair problem in $O(\log n \log \log n)$ time using $O(n)$ processors on a CREW PRAM.

As in our solution to the convex hull problem, we will be dividing the input set of points into $\sqrt{n}$ subsets divided by vertical cut-lines. Let $R_1, \ldots, R_{\sqrt{n}}$ be these subsets in left-to-right order, i.e., R_i is left of R_j if $i < j$. We define the *region-width* of a point set R_i to be the distance between the two vertical cut-lines separating R_i from R_{i-1} and R_{i+1}, respectively. Note: the region-width of R_1 and $R_{\sqrt{n}}$ is defined to be ∞. We present the closest-pair algorithm CP below.

Algorithm CP

Input. A set S of n points in the plane.

Output. A closest pair of points in S.

Method. Before giving the details, we present a high-level description of the various steps of the algorithm. First, we partition S into $\sqrt{n}$ sets, of size $\sqrt{n}$ each, using vertical cut-lines, and recursively solve the closest-pair problem for each. Taking the closest of the $\sqrt{n}$ pairs returned by the parallel recursive call gives us a closest pair of points in S not separated by a cut-line. Let δ be the distance between these two points. For our combining step to run quickly (i.e., in $O(\log n)$ time) there should not be more than a constant number of cut-lines which are within δ of one another. Since this may not be the case at present, we do not perform our combining step at this point. Instead, we repartition S by removing cut-lines between adjacent point sets with region-widths which are "too small," thereby coalescing the two sets into one. This gives us a better distribution of the remaining vertical cut-lines. Even after coalescing, we still do not combine the subproblems, because in removing a cut-line we coalesce previously solved subproblems into conglomerates which must now be re-solved. Consequently, for each conglomerate point set, we use the $\sqrt{n}$ divide-and-conquer technique again, dividing the conglomerate horizontally, and solving each of the resulting horizontally divided sets recursively. Dividing the conglomerate point sets horizontally guarantees that cut-lines will be far enough from each other to allow for a combining step which runs in $O(\log n)$ time. So, after recursively solving the horizontal problems, we are now ready to combine the solutions to the subproblems. We do this by first combining the solutions to the horizontally divided sets, and then combining the solutions to the vertically divided sets. Below we give a high-level description of each step in the algorithm. We define the *smallest distance*, $\delta(R)$, of a point set R to be the distance between a closest pair of points in R. If we always associate a specific closest pair of points with a smallest distance, then we can reformulate the closest-pair problem as the following: given a set S of n points in the plane, compute $\delta(S)$. This is the formulation we will use.

High-Level Description of CP

Step 1. Partition S into point sets $R_1, R_2, \ldots, R_{\sqrt{n}}$, each of size $\sqrt{n}$, separated by vertical cut-lines such that R_i is left of R_j if $i < j$ (see Fig. 1). Let I denote the index set $\{1, 2, \ldots, \sqrt{n}\}$.

Step 2. Recursively compute $\delta(R_i)$ for each R_i, $i \in I$, in parallel.

Step 3. Compute $\delta = \min\{\delta(R_i) \mid i \in I\}$.

Comment. The pair associated with δ is a closest pair of points in S not separated by any of the vertical cut-lines which separate the R_i's from one another.

Step 4. Repartition S into $\{H_1, H_2, \ldots, H_l\}$, $l \leq \sqrt{n}$, so that there are never more than two vertical cut-lines which are within δ of each other. The new partition is obtained by starting with $R_1, \ldots, R_{\sqrt{n}}$, and repeatedly removing cut-lines between pairs of adjacent regions whose region-widths are both less than δ, coalescing two sets into one each time. Let I' denote the new index set $\{1, 2, \ldots, l\}$.

Comment. From this point on in the algorithm when we refer to vertical cut-lines we mean the ones which survived this repartitioning step.

Step 5. In parallel for each H_i, check if $\delta(H_i) < \delta$, and, if so, assign $\delta_i = \delta(H_i)$. The method we use to test this is such that we only compute $\delta(H_i)$ if it is less than δ. If we detect that $\delta(H_i) \geq \delta$ (without explicitly computing $\delta(H_i)$), then we assign $\delta_i = \delta$. Note that if H_i is one of the original sets ($H_i = R_j$ for some j), then we can assign $\delta_i = \delta$ immediately. Otherwise, if H_i resulted from coalescing two or more of the original sets ($H_i = R_j \cup \cdots \cup R_{j+d}$ for some j and d), then this computation is done by using the $\sqrt{n}$-divide-and-conquer technique again. We divide each such H_i by horizontal cut-lines into $\sqrt{|H_i|}$ subsets $r_{i,j}$, solve each $r_{i,j}$ recursively, and combine all the subproblem solutions in $O(\log n)$ time.

Step 6. Compute $\delta' = \min\{\delta_i \mid i \in I'\}$.

Comment. Note that $\delta' \leq \delta$, and that the pair associated with δ' is a closest pair of points in S not separated by any of the vertical cut-lines which separate the H_i's from one another.

Step 7. Find all pairs of points in S which are separated by a vertical cut-line and are closer than δ' to one another. If there are such pairs of points then $\delta(S)$ is the distance between a closest such pair; otherwise, $\delta(S) = \delta'$.

End of High-Level Description of CP

We now show how to perform each of the above steps quickly in parallel. It is trivial to do the partitioning of Step 1 and the computation of Step 3 in $O(\log n)$ time using $O(n)$ processors. So we begin our detailed description of the algorithm CP with Step 4. Recall that at this point in the algorithm we have found a closest pair of points in S not separated by any of the vertical cut-lines that separate the R_i's from the each other, and that δ is the distance between this pair of points.

Details of Step 4. We perform this repartitioning step by the simple divide-and-conquer procedure REPARTITION which follows. For simplicity of expression let $k = \sqrt{n}$.

Procedure REPARTITION

Step 4.1. Let $\mathfrak{R} = \{R_1, R_2, \ldots, R_k\}$. Divide the collection $\mathfrak{R}$ into two contiguous collections $\mathfrak{R}_1 = \{R_1, \ldots, R_{k/2}\}$ and $\mathfrak{R}_2 = \{R_{k/2+1}, \ldots, R_k\}$ ($\mathfrak{R}_1$ begin to the left of $\mathfrak{R}_2$). Assign to $R_{k/2}$ the same region-width in $\mathfrak{R}_1$ as it had in $\mathfrak{R}$. Similarly, assign to $R_{k/2+1}$ the same region-width in $\mathfrak{R}_2$ as it had in $\mathfrak{R}$.

Step 4.2. Recursively apply procedure REPARTITION to $\mathfrak{R}_1$ and $\mathfrak{R}_2$ in parallel. Assume that after the parallel recursive call returns there will be no two adjacent point sets in $\mathfrak{R}_1$, or in $\mathfrak{R}_2$, which both have region-width less than δ. (This is the invariant we will maintain.)

Step 4.3. If the region-width of the rightmost point set in $\mathfrak{R}_1$ and the region-width of the leftmost point set in $\mathfrak{R}_2$ are both less than δ, then coalesce them into one point set by removing the cut-line between them. Otherwise, do nothing. In coalescing two point sets we must compress the list of points sets, removing one point set. This can be done in $O(1)$ time using $O(n)$ processors, giving the new partition $\{H_1, H_2, \ldots, H_l\}$, $l \leq \sqrt{n}$.

End of REPARTITION

Analysis of Step 4. The correctness of the above implementation of Step 4 follows by a simple inductive argument. The time complexity $T_4(n)$ of Step 4 is determined by the recurrence relation $T_4(n) = T_4(n/2) + b_4$, where b_4 is some constant, which has solution $T_4(n) = O(\log n)$. The processor bound $P_4(n)$ is determined by the recurrence relation $P_4(n) = \max\{2P_4(n/2), c_4 n\}$, for some constant c_4, which has solution $P_4(n) = O(n)$.

Details of Step 5. Recall that in Step 5 we wish to check if $\delta(H_i) < \delta$, and if so, assign $\delta_i = \delta(H_i)$ (and, if not, then assign $\delta_i = \delta$). The main idea of Step 5 is to perform essentially the same computation as Steps 1–3 above for each H_i, except that cut-lines in Step 5 are horizontal instead of vertical. The main idea behind the method for combining subproblem solutions is to construct for each subproblem set $r_{i,j}$ two special "candidate" sets, which consist of points in $r_{i,j}$ which could be close to points in some other $r_{i,k}$, and then for each point p search in a constant number of these sets to find a point closest to p which is separated from p by a horizontal cut-line. The details follow.

Step 5.1. Sort the points in H_i by y-coordinate and partition H_i into subsets $r_{i,1}, r_{i,2}, \ldots, r_{i,\sqrt{n_i}/2}$, separated by horizontal cut-lines, each of size of $2\sqrt{n_i}$ (where $n_i = |H_i|$), and such that $r_{i,j}$ is below $r_{i,k}$ if $j < k$ (see Fig. 4). Let J_i denote $\{1, 2, \ldots, \sqrt{n_i}/2\}$.

Step 5.2. Recursively compute $\delta(r_{i,j})$ for each $r_{i,j}$, $j \in J_i$, in parallel.

Step 5.3. Compute $\min\{\delta(r_{i,j}) \mid j \in J_i\}$, and let ϵ_i be the smaller of this value and δ.

Comment. ϵ_i is no greater than the distance between a closest pair of points in H_i not separated by a horizontal cut-line. We are now ready to do the combining step of the divide-and-conquer.

Step 5.4. In parallel for each $j \in J_i$ construct the set $N_{i,j}$ and $S_{i,j}$, where $N_{i,j}(S_{i,j})$ is the set of points in $r_{i,j}$ which are within ϵ_i of $r_{i,j}$'s northern (southern) cut-line. (This can be done by a parallel prefix computation.) Sort the points of each $N_{i,j}$ and $S_{i,j}$ by x-coordinate.

Step 5.5. In parallel for every point $p \in H_i$ construct the set $D_i(p)$,

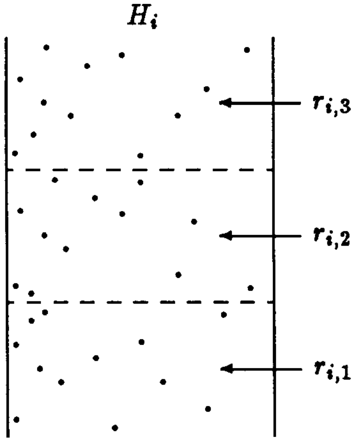

FIG. 4. A horizontal partitioning of a point set H_i with $n_i = 36$.

where $D_i(p)$ is defined as follows: If $r_{i,j}$ is the point set containing p, then $D_i(p)$ is the set of all points in $N_{i,j-3} \cup N_{i,j-2} \cup N_{i,j-1} \cup S_{i,j+1} \cup S_{i,j+2} \cup S_{i,j+3}$ with x-coordinate within ϵ_i of p's x-coordinate (if one of the sets in the above union is not defined, then use $\varnothing$ in its place). We will show later (in Lemmas 3.1 and 3.2) that there can be no more than three horizontal cut-lines within ϵ_i of one another and that $|D_i(p)| = O(1)$. Thus, $D_i(p)$ can be constructed for any p by performing $O(1)$ binary searches, and this step can be performed in $O(\log n_i)$ time using $O(n_i)$ processors.

Step 5.6. In parallel for each $p \in H_i$ find a point in $D_i(p)$ closest to p (provided that $D_i(p) \neq \varnothing$). Call this point $q(p)$ for each p, and let $d(p)$ be the distance from p to $q(p)$ ($d(p) = \infty$ if $D_i(p) = \varnothing$).

Step 5.7. Compute $\min\{d(p) \mid p \in H_i\}$ and take δ_i to be the minimum of this distance and ϵ_i.

Comment. Note if $\delta(H_i) < \delta$, then $\delta_i = \delta(H_i)$, and $\delta_i = \delta$ otherwise.

Analysis of Step 5. The analysis of Step 5 is quite involved. We begin the proof of correctness by proving the following lemma.

LEMMA 3.1. *There are no more than three horizontal cut-lines which are within ϵ_i of one another in any point set H_i considered in Step 5, $i \in I'$.*

Proof. Since $\epsilon_i \leq \delta$ for all $i \in I'$, it is sufficient that we prove that there are no more than three horizontal cut-lines which are within δ of one another in any point set H_i. Suppose there are four horizontal cut-lines within δ of one another in some H_i. Let $Q = r_{i,j} \cup r_{i,j+1} \cup r_{i,j+2}$, $j \in J_i$, be the set of points which are bounded by these lines. Let $d \geq 2$ be the number of original point sets which were coalesced to create H_i. Then $n_i = |H_i| = d\sqrt{n}$, and $|r_{i,j}| = 2\sqrt{n_i} = 2d^{1/2}n^{1/4}$, for all $j \in J_i$. Since $\sqrt{n} \geq d$, $|r_{i,j}| \geq 2d$, for all $j \in J_i$. Recalling the method for Step 4, note that each of the d original point sets must have had region-width less than δ to have been coalesced. The value δ was found by solving the closest-pair problem for each original point set, so there can be at most four points in Q for any of the d original point sets which were coalesced to form H_i (see Fig. 5). Thus, $|Q| \leq 4d$. But since Q contains 3 $r_{i,j}$'s, $|Q| \geq 6d$. This is obviously a contradiction. ■

Lemma 3.1 and the definition of $D_i(p)$ imply that for every point p in H_i, if a point $q \in H_i$ has x- and y-coordinates both within ϵ_i of p's x- and y-coordinates (respectively), then $q \in D_i(p)$. Thus, when we find a closest point to p in $D_i(p)$ we are in fact finding a closest point to p in H_i separated from p by a horizontal cut-line (provided $D_i(p) \neq \varnothing$). Recall that, for each p, if $d(p) < \infty$, then $d(p)$ is the distance to such a point. Also recall that ϵ_i is the smaller of δ and $\min\{\delta(r_{i,j}) \mid j \in J_i\}$. Therefore, taking the minimum of ϵ_i and the minimum $d(p)$ value in Step 5.7 gives us $\delta(H_i)$ if $\delta(H_i) < \delta$, and gives us δ otherwise. This establishes the correctness of Step 5.

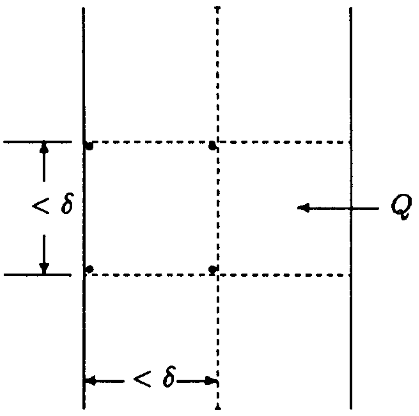

FIG. 5. The upper bound on number of points in Q which were all in one of the original point sets is 4.

We now turn to the complexity bounds of Step 5. We first prove the following lemma.

LEMMA 3.2. $|D_i(p)| = O(1)$ *for every* $p \in H_i$.

Proof. Recall that ϵ_i is no greater than the distance between a closest pair of points in H_i not separated by a horizontal cut-line. Thus, there can be no more than six points in any $2\epsilon_i \times \epsilon_i$ rectangular subset of any $r_{i,j}$ (see Fig. 6), for any H_i. Hence, there can be at most six points in $D_i(p)$ taken from any $N_{i,j-k}$ or $S_{i,j+k}$, $k = 1, 2, 3$. Therefore, $|D_i(p)| \leq 36$ for any p. ∎

Lemma 3.2 implies that Steps 5.5 and 5.6 run in $O(\log n_i)$ and $O(1)$ time, respectively, using $O(n_i)$ processors, for any H_i. From observations already made in this paper we know that Steps 5.1, 5.3, 5.4, and 5.7 run in $O(\log n_i)$ time using $O(n_i)$ processors, for any H_i. Thus, the running time $T_5(n)$ of Step 5 is characterized by the recurrence relation $T_5(n) = \max\{T(2\sqrt{n_i}) + b_5 \log n_i \mid i \in I'\}$, where b_5 is some constant and $T(n)$ is the time complexity

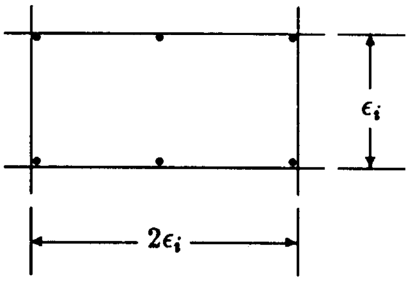

FIG. 6. The upper bound on number of points in any $D_i(p)$ selected from any one $N_{i,j}$ or $S_{i,j}$ is 6.

of the entire CP algorithm. Since $n_i \leq n$ for all $i \in I'$, we can rewrite this as $T_5(n) \leq T(2\sqrt{n}) + b_5 \log n$ (this is the formulation we will use in analyzing the time complexity of the algorithm CP). The processor bound $P_5(n)$ is determined by the recurrence relation $P_5(n) = \Sigma_{i \in I'} \max\{\frac{1}{2}\sqrt{n_i}\, P(2\sqrt{n_i}), c_5 n_i\}$, where c_5 is some constant and $P(n)$ is the number of processors needed by the algorithm CP.

Details of Step 6. It is clear that we can compute $\delta' = \min\{\delta_i \mid i \in I'\}$ in $O(\log n)$ time using $O(n)$ processors.

Details of Step 7. Recall that in Step 7 we wish to find all pairs of points in S which are separated by a vertical cut-line and are closer than δ' to one another. The method is essentially the same as the combining steps of Step 5 (Steps 5.4–5.7). The details follow.

Step 7.1. In parallel for each $i \in I'$ construct the set E_i and W_i, where E_i (W_i) is the set of points in H_i which are within δ' of the eastern (western) cut-line for H_i. (This can be done by a parallel prefix computation.) Sort the points of each E_i and W_i by y-coordinate.

Step 7.2. In parallel for every point $p \in S$ construct the set $D(p)$, where $D(p)$ is defined as follows (similar to $D_i(p)$): If H_i is point set containing p, then $D(p)$ is the set of all points in $E_{i-2} \cup E_{i-1} \cup W_{i+1} \cup W_{i+2}$ with y-coordinate within δ' of p's y-coordinate (if any set in the above union is not defined, then use $\emptyset$ in its place). Clearly, $D(p)$ can be constructed for any p by performing $O(1)$ binary searches.

Comment. Recall that from the repartitioning done in Step 4, there cannot be more than two vertical cut lines with δ of one another.

Step 7.3. In parallel for each $p \in S$ find a point in $D(p)$ closest to p, and call it $q(p)$. Let $d(p)$ be the distance from p to $q(p)$ ($d(p) = \infty$ if $D(p) = \emptyset$).

Step 7.4. Compute $\min\{d(p) \mid p \in S\}$ and take $\delta(S)$ to be the minimum of this distance and δ'.

Analysis of Step 7. It follows from an argument similar to the one used in the proof of Lemma 3.2 that $|D(p)| = O(1)$ for all $p \in S$. Thus, Step 7 can be performed in $O(\log n)$ time using $O(n)$ processors. We turn to the proof of correctness. For each p, if a point q is separated from p by a vertical cut-line and has x- and y-coordinates both within δ' of p's x- and y-coordinates (resp.), then $p \in D(p)$. This is because after performing the repartition procedure of Step 4, there are at most two vertical cut-lines which are within δ of one another (hence, within δ' of one another). So, it is correct to set $\delta(S)$ to the smaller of $\min\{d(p) \mid p \in S\}$ and δ'.

End of Algorithm CP

We summarize the above discussion in the following theorem.

THEOREM 3.3. *The algorithm CP finds a closest pair of n points in the plane in $O(\log n \log\log n)$ time using $O(n)$ processors on a CREW PRAM.*

Proof. The correctness of CP follows from a simple inductive argument based on the discussion presented above. Combining the time complexity analysis for each of the above steps we get that the time complexity $T(n)$ of the algorithm CP is characterized by the recurrence relation $T(n) \leq T(\sqrt{n}) + T(2\sqrt{n}) + b \log n$, where b is some constant, which has solution $T(n) = O(\log n \log\log n)$. The processor bound $P(n)$ of the algorithm CP is characterized by the recurrence relation

$$P(n) = \max\left\{cn,\ \sqrt{n}\,P(\sqrt{n}),\ \sum_{i\in I'} \max\left\{c_5 n_i,\ \frac{1}{2}\sqrt{n_i}\,P(2\sqrt{n_i})\right\}\right\},$$

where c and c_5 are constants. Using the fact that $\Sigma_{i\in I'} n_i = n$, we get that $P(n) = O(n)$. ■

It is worth mentioning that our algorithm will work with any of the L_k distance metrics. Recall that in any of the L_k metrics every point within a specific distance ϵ from a point p has x- and y-coordinates both within ϵ of p's x- and y-coordinates. Therefore, since we define $D_i(p)$ ($D(p)$) so that it contains all points separated from p by a horizontal (vertical) cut-line and with x- and y-coordinates both within ϵ_i (δ') of p's x- and y-coordinates (resp.), then $\delta(S)$ will be computed correctly no matter which L_k metric we use to define distance.

4. CONCLUSION

We gave efficient parallel algorithms for solving some geometric problems. Namely, we have shown how to solve the planar convex hull problem, and related problems, in $O(\log n)$ time and the closest-pair problem in $O(\log n \log\log n)$ time on a CREW PRAM with $O(n)$ processors. This, of course, implies that given a fixed number of processors, say k, one can solve the planar convex hull problem, and related problems in $O((n/k)\log n)$ time and the closest-pair problem in $O((n/k)\log n \log\log n)$ time, by using the k processors to simulate the $O(n)$ processors used in our algorithms. The new parallel divide-and-conquer technique we presented for solving these problems is very general, and can be helpful in tackling other geometric problems as well. For example, the authors used this technique, in conjunction with a parallel technique analogous to plane sweeping, to solve the problem of triangulating a simple polygon in $O(\log n \log\log n)$ time using $O(n)$ processors [3].

Acknowledgments

We would like to thank Greg Shannon for his comments and careful reading of an earlier draft of this work. We would also like to thank the referees for their comments, which considerably improved the presentation of Section 3.

References

1. Aggarwal, A., Chazelle, B., Guibas, L., Ó'Dúnlaing, C., and Yap, C. Parallel computational geometry. *Proc. 25th IEEE Symp. on Foundations of Computer Science*, 1985, pp. 468–477.
2. Ajtai, M., Komlós, J., and Szemerédi, E. Sorting in c log n parallel steps. *Combinatorica* **3** (1983), 1–19.
3. Atallah, M. J., and Goodrich, M. T. Efficient plane sweeping in parallel. *Proc. 2nd AMC Symp. on Computational Geometry*, in press; also available as Purdue Tech. Rep. CSD-TR-563, Mar. 1986.
4. Bentley, J. Multidimensional divide-and-conquer. *Comm. ACM* **23,** 4 (Apr. 1980), 214–229.
5. Bentley, J. L., and Shamos, M. I. Divide-and-conquer in multidimensional space. *Proc. Symp. on Theory of Comp.*, 1976, pp. 220–230.
6. Borodin, A., and Hopcroft, J. E. Routing, merging, and sorting on parallel models of computation. *J. Comput. System Sci.* **30,** 1 (Feb. 1985), 130–145.
7. Brown, K. Q. Geometric transformations for fast geometric algorithms. Ph.D. dissertation, Department of Computer Science, Carnegie–Mellon University, Pittsburgh, Pa., Dec. 1979 (cited in [15]).
8. Chazelle, B. Computational geometry on a systolic chip. *IEEE Trans. Comput.* **C-33,** 9 (Sept. 1984), 774–785.
9. Chow, A. Parallel algorithms for geometric problems. Ph.D. thesis, University of Illinois, Urbana–Champaign, Dec. 1980.
10. Goodrich, M. T. An optimal parallel algorithm for the all nearest-neighbor problem for a convex polygon. Purdue University Computer Science Tech. Rep. CSD-TR-533, Aug. 1985.
11. Graham, R. L. An efficient algorithm for determining the convex hull of a finite planar set. *Inform. Process. Lett.* **1** (1972), 132–133.
12. Kruskal, C. P., Rudolph, L., and Snir, M. The power of parallel prefix. *Proc. 1985 Int. Conf. on Parallel Processing*, St. Charles, Ill., pp. 180–185.
13. Ladner, R. E., and Fischer, M. J. Parallel prefix computation. *J. Assoc. Comput. Mach.* (Oct. 1980), 831–838.
14. Lee, D. T., and Preparata, F. P. Computational geometry—A survey. *IEEE Trans. Comput.* **C-33,** 12 (Dec. 1984), 1072–1101.
15. Leighton, T. Tight bounds on the complexity of parallel sorting. *IEEE Trans. Comput.* **C-34,** 4 (Apr. 1985), 344–354.
16. Miller, R., and Stout, Q. F. Computational geometry on a mesh-connected computer. *Proc. 1984 IEEE Int. Conf. on Parallel Processing*, pp. 66–73.
17. Overmars, M. H. and Van Leeuwen, J. Maintenance of configurations in the plane. *J. Comput. System Sci.* **23** (1981), 166–204.
18. Preparata, F. P. An optimal real-time algorithm for planar convex hulls. *Comm. ACM* **22,** 7 (July 1979), 402–405.
19. Preparata, F. P., and Hong, S. J. Convex hulls of finite sets of points in two and three dimensions. *Comm. ACM* **20, 2,** (Feb. 1977) 87–93.

20. Preparata, F. P., and Muller, D. E. Finding the intersection of n half-spaces in time $O(n \log n)$, *Theoret. Comput. Sci.* **8** (1979), 45–55.

21. Shamos, M. I., and Hoey, D. Closest-point problems. *Proc. 16th IEEE Symp. on Foundations of Comp. Sci.*, 1975, pp. 151–162.

22. Valiant, L. Parallelism in comparison problems. *SIAM J. Comput.* **4**, 3 (Sept. 1975), 348–355.

23. Yao, A. C. A lower bound to finding convex hulls. *J. Assoc. Comput. Mach.* **28** (1981), 780–787.