JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 15, 383-395 (1999)

Short Paper

Computational Geometry on the Broadcast Communication Model*

CHANG-BIAU YANG
Department of Applied Mathematics
National Sun Yat-Sen University
Kaohsiung, Taiwan 804, R.O.C.
E-mail: cbyang@math.nsysu.edu.tw

In this paper, we solve three geometric problems, including the ranking, convex hull and closest pair problems, under the broadcast communication model. To solve these problems, we propose a general scheme, the p-division approach, which is based upon the divide-and-conquer strategy. In the 2-dimensional space, the time complexities of our algorithms for solving these problems are all $O(n+\frac{n}{p}\log\frac{n}{p})$, where n is the number of input points and p is the number of processors used. Furthermore, our algorithms are all conflict-free and optimal. In the k-dimensional space, $k \geq 3$, our ranking algorithm requires $O(\frac{n}{p}\log^{k-1}\frac{n}{p}+n\log^{k-2}\frac{n}{p})$ time.

Keywords: broadcast communication, computational geometry, parallel algorithm, divide-and-conquer, conflict-free

1. INTRODUCTION

The **broadcast communication model** consists of some processors sharing one common channel for communications [6, 12, 15, 17, 19-21]. In this model, all processors can communicate with one another only through a shared communication channel. Whenever a processor broadcasts messages, any other processor can receive the messages via the shared channel. If more than one processor attempts to broadcast simultaneously, a **broadcast conflict** occurs. When a conflict occurs, we have to use a conflict resolution scheme to resolve the conflict. The famous Ethernet, a local area network, is an implementation of such model. An algorithm is **conflict-free** if no conflict occurs during execution of the algorithm. Of course, a conflict-free algorithm has better performance since no resolution time is needed.

Received April 11, 1996; accepted May 1, 1998.
Communicated by Wen-Lian Hsu.
*This research work was partially supported by the National Science Council of the Republic of China under contract NSC-82-0208-M-110-027.

Some researchers have solved problems under the broadcast communication model. Dechter and Kleinrock [6], Marberg and Gafni [12], Ramarao [15], Tang and Chiu [17], and Yang, Lee and Chen [21] have proposed sorting algorithms under this model. Yang et al. solved some graph problems [19, 20] and the maximum finding problem [16] under this model.

Computational geometry [14] has many applications in various areas, such as computer graphics, pattern recognition, VLSI layout, computer-aided design and computer-aided manufacture. Thus, computational geometry has attracted the attention of many computer scientists. In this paper, we shall solve some geometric problems under the broadcast communication model. They are the ranking, convex hull and closest pair problems.

Many researchers have proposed sequential [2, 10, 13] or parallel [1, 4, 8, 9, 18] algorithms to solve the convex hull problem. Almost all of these algorithms are based upon the divide-and-conquer strategy. For parallel algorithms, Aggarwal et al. [1], and Atallah and Goodrich [4] used the $\sqrt{n}$ divide-and-conquer method, Fjallstrom et al. [8] used the $n^{\frac{1}{3}}$ divide-and-conquer method, and Goodrich [9] used the $\sqrt{\frac{n}{d}}$ divide-and-conquer method, where d is an integer denoted as the depth of recursion.

For solving the closest pair problem, some sequential [5] or parallel [4, 7] algorithms are available. Bentley [5] proposed a sequential algorithm and Atallah et al. [3] proposed a parallel algorithm to solve the ranking problem. As with the algorithms for solving the convex hull problem, the algorithms for solving these problems are almost all based upon the divide-and-conquer strategy.

Since all three problems can be solved using the divide-and-conquer strategy, in this paper, we shall propose a general scheme, the p-division approach, to solve them under the broadcast communication model. When the 2-dimensional space is considered, all of these algorithms require $O(n+\frac{n}{p}\log\frac{n}{p})$ time where n is the number of input points and p is the number of processors used. All of these algorithms are conflict-free. Furthermore, we shall prove that they are all optimal under the broadcast communication model. When the k-dimensional space is considered, $k \geq 3$, the ranking algorithm requires $O(\frac{n}{p}\log^{k-1}\frac{n}{p}+n\log^{k-2}\frac{n}{p})$ time.

The rest of this paper is organized as follows. In Section 2, we shall give the general scheme, the p-division approach, for solving the geometric problems. In Section 3, we shall give the lower bound of time required for solving a class of problems under the broadcast communication model. Sections 4 through 6 are devoted to a discussion of the ranking, convex hull and closest pair problems, respectively. Finally, concluding remarks will be given in Section 7.

2. THE *P*-DIVISION APPROACH

Since all of the geometric problems we shall solve in this paper can be solved using the divide-and-conquer strategy, we can generalize this strategy to the ***p*-division** approach. Suppose that n input points are given. The p-division approach divides the input data into p parts, each consisting of $\frac{n}{p}$ points, where p is the number of processors used. In each

processor, a sequential divide-and-conquer algorithm is invoked. Finally, the results obtained in all processors are merged together. The outline of the *p*-division approach under the broadcast communication model with *p* processors is as follows.

Step 1: Sort the input points according to their *x*-values. Then partition the input points into *p* disjoint subsets, denoted as S_1, S_2,..., S_p, according to their *x*-values.
Step 2: Using a sequential algorithm, each processor takes one S_i as its input instance to solve the problem which we desire to solve.
Step 3: Each processor broadcasts its points one by one. After one of the processors finishes its broadcast, all other processors update their status. In other words, merging is done in this step.

To accomplish Step 1, we apply the sorting algorithm proposed by Yang et al. [21], which is conflict-free. In Step 2, a sequential algorithm is used to solve the problem in each processor. For all of the problems on the 2-dimensional space that we will solve in this paper, there are optimal sequential algorithms, all of which need $O(n \log n)$ time if the input size is n [5, 10, 11]. Step 3 merges the results obtained in Step 2. Since the points are broadcast one by one, Step 3 is also conflict-free. Thus, the entire algorithm is conflict-free. The time required for each step is as follows:

Step 1: $O(n + \frac{n}{p}\log\frac{n}{p})$**;**
Step 2: $O(\frac{n}{p}\log\frac{n}{p})$;
Step 3: $O(n)$.

Therefore, the total time complexity of our algorithm is $O(n + \frac{n}{p}\log\frac{n}{p})$.

3. LOWER BOUNDS

Yang et al. [21] gave the lower bound of the sorting problem under the broadcast communication model. In this section, we shall extend the lower bound to a larger class of problems and prove it with a similar technique.

A problem *P* with *n* inputs is said to have the **neighboring property** if there exists linear ordering on the inputs so that any algorithm used to solve *P* has to at least perform some operations on every pair of neighboring data elements under the ordering. For example, the sorting problem has the neighboring property. Note that before the problem is solved, the linear ordering of *P* is unknown. In other words, if *P* has the neighboring property, then every pair of input data elements may be one neighboring pair. Otherwise, the problem has been partially solved. Let a_1, a_2,..., a_n be the linear ordering of *P*. In the ordering, a_{k-1} is called the **predecessor** of a_k, $2 \le k \le n$.

Lemma 1: Let a_1, a_2,..., a_n be the linear ordering of a problem *P*. Suppose that it is stored in *p* processors arbitrarily, and that the number of data elements stored in processor *i* is n_i, $1 \le i \le p$. Let n_j be the maximum among all n_i's, $1 \le i \le p$. If $n_j < \frac{n}{2}$, then there exists one case such that a_{k-1} and a_k, $2 \le k \le n$ are not stored in the same processor [21].

Proof: To prove this lemma, such a case should be generated. Without loss of generality, we assume that n is even. Since the data stored in the processors are arbitrary, any pattern for storing data in processors is possible. In one of the storing patterns, processor i, $1 \le i \le p$, stores the data in the positions $n_1 + n_2 + ... + n_{i-1} + 1$ through $n_1 + n_2 + ... + n_i$ of the ordering $a_1a_3a_5... a_{n-1}a_2a_4.... a_n$. We claim that in this storing pattern, a_{k-1} and a_k, $2 \le k \le n$, will not be stored in the same processor. Two cases should be considered as follows.

Case 1: k is odd. If both a_k and a_{k-1} are stored in processor i, $1 \le i \le p$, then the subsequence $a_ka_{k+2}... a_{n-1}a_2a_4... a_{k-1}$, whose length is $\frac{n}{2}$, would be stored in processor i. However, this is impossible since $n_i < \frac{n}{2}$. Thus, processor i can not store both a_k and a_{k-1}.

Case 2: k is even. This case is similar to case 1. □

Theorem 1: Under the broadcast communication model, the lower bound for an algorithm using p processors, $p \ge 1$, to solve a problem P having the neighboring property with n inputs is $\Omega(n)$ under the assumption that each input is preloaded into exactly one of the p processors.

Proof: Suppose that n_i data elements are preloaded into processor i. Here, $\sum_{i=1}^{p} n_i = n$. Let n_j be the maximum among all n_i's, $1 \le i \le p$. Let $a_1, a_2, a_3, ..., a_n$ be the linear ordering of P. Since P has the neighboring property, we have to at least perform some operations on a_{k-1} and a_k, $2 \le k \le n$. Two cases are to be considered.

Case 1: $n_j \ge \frac{n}{2}$. For each data element in processor j, processor j either performs the necessary operations on it and its predecessor, or j broadcasts it to other processors to perform the operations. This requires at least $\Omega(\frac{n}{2}) = \Omega(n)$ time.

Case 2: $n_j < \frac{n}{2}$. Without loss of generality, it is assumed that n is even. Consider the following $\frac{n}{2}$ data elements: $a_2, a_4, a_6,..., a_n$. For each element a_k where $2 \le k \le n$ and n is even, by Lemma 1, there exists such a case that a_{k-1}, which is the predecessor of a_k, is not stored in the same processor with a_k. In order to perform the necessary operations on a_k and a_{k-1}, either a_k or a_{k-1} must be broadcast. At least $\frac{n}{2}$ broadcasts are required if we want to perform the necessary operations on $a_2, a_4, a_6,..., a_n$ with their predecessors, respectively. Thus, at least $\Omega(n)$ time is required.

In both of these cases, this theorem holds. □

4. THE RANKING PROBLEM

In this section, we shall propose an algorithm to solve the ranking problem [3, 5]. Let S be a set of points on a 2-dimensional plane. Suppose that $u, v \in S$, $u = (x_u, y_u)$ and $v = (x_v, y_v)$. If $x_u > x_v$ and $y_u > y_v$, then u is said to **dominate** v. Let $T = \{v \mid v \in S \text{ and } u \text{ dominates } v\}$. The **rank** of u with respect to S, denoted as $Rank(u, S)$, is defined as $|T|$, which is the cardinality of T. For a given set S of points, the **ranking problem** is to calculate $Rank(u, S)$ for each point $u \in S$.

For a point $u = (w, z)$, w is said to be the x-**value** of u, denoted as $x(u)$, and z is said to be the y-**value** of u, denoted as $y(u)$. Let S_1 and S_2 be two sets of points. If $S_1 \cap S_2 = \phi$ and for any $u \in S_1$, $v \in S_2$, $x(u) < x(v)$, then S_1 is said to be on the **left** of S_2, denoted as $x(S_1) < x(S_2)$.

Now, we shall use the *p*-division approach to solve the ranking problem under the broadcast communication model. It is assumed that *p* processors, numbered as 1, 2,..., *p*, are used. And initially, the *n* input points are stored in these *p* processors evenly. Our algorithm is as follows.

Algorithm Ranking.
Input: A set *S* of *n* planar points.
Output: The ranks of all points in *S*.

Step 1: Sort the *n* points in *S* according to their *x*-values. Denote the subset of points stored in processor *i* as S_i. After sorting, $x(S_i) < x(S_j)$ if $i < j$.
Step 2: Each processor *i* computes $Rank(u, S_i)$ for each $u \in S_i$ and sorts the points in S_i into an ascending sequence according to their *y*-values.
Step 3: $i \leftarrow 1$. Repeat the following substeps *p* times.
  Step 3.1: Processor *i* broadcasts the points in S_i one by one according to the ascending sequence of the *y*-values.
  Step 3.2: Each processor *j*, $j > i$, updates the ranks of the points in S_j. For each point $u \in S_j$, the rank of *u* is increased by the number of points in S_i whose *y*-values are less than $y(u)$. After this substep, for each point $u \in S_j$, $Rank(u, S_j \cup S_1 \cup S_2 \cup ... \cup S_i)$ is obtained.
  Step 3.3: $i \leftarrow i+1$.

The problem in Step 3.2 corresponds to that of finding the ranks of all elements in two sorted lists, consisting of the *y*-values. This can be accomplished by the linear merging algorithm on two sorted lists, which requires linear time. Thus, the time required for each step is as follows:

Step 1: $O(n + \frac{n}{p}\log\frac{n}{p})$ [21];
Step 2: $O(\frac{n}{p}\log\frac{n}{p})$;
Step 3: $O(\frac{n}{p} \cdot p) = O(n)$.

Hence, the time complexity of Algorithm Ranking is $O(n + \frac{n}{p}\log\frac{n}{p})$.

Theorem2: Under the broadcast communication model, the lower bound for an algorithm using *p*, $p \geq 1$, processors to solve the ranking problem with *n* points is $\Omega(n + \frac{n}{p}\log\frac{n}{p})$.

Proof: Under any parallel computation model, the time required for an algorithm using *p*, $p \geq 1$, processors to sort *n* data elements is $\Omega(\frac{n\log n}{p})$ since the lower bound for a sequential sorting algorithm to sort *n* data elements is $\Omega(n \log n)$. Combining with Theorem 1, we obtain that the lower bound for sorting under the broadcast communication model is $\Omega(n + \frac{n\log n}{p}) = \Omega(n + \frac{n}{p}\log\frac{n}{p})$ since the sorting problem has the neighboring property. Also, the sorting problem can be reduced to the ranking problem since the former is a 1-dimensional ranking problem. □

By Theorem 2, under the broadcast communication model, our algorithm for the ranking problem is optimal.

Bentley [5] proposed a multidimensional divide-and-conquer strategy to sequentially solve some geometric problems, including the ranking problem, the maxima finding problem and the closest pair problem. For the ranking problem with n points in the k-dimensional space, $k \geq 3$, his algorithm requires $O(n \log^{k-1} n)$ time and the merging step takes $O(n \log^{k-2} n)$ time. Combining his algorithm and Algorithm Ranking, we have the following results if we solve the ranking problem in the k-dimensional space:

Step 1: $O(n + \frac{n}{p}\log\frac{n}{p})$;
Step 2: $O(\frac{n}{p}\log^{k-1}\frac{n}{p})$;
Step 3: $O(n + p \cdot \frac{n}{p}\log^{k-2}\frac{n}{p}) = O(n\log^{k-2}\frac{n}{p})$.

Thus, the total time required for solving the k-dimensional ranking problem is $O(\frac{n}{p}\log^{k-1}\frac{n}{p} + n\log^{k-2}\frac{n}{p})$, $k \geq 3$.

5. THE CONVEX HULL PROBLEM

In this section, we shall propose an algorithm to find the convex hull of a set of planar points [1, 2, 4, 8-10, 13, 18]. Let S be a set of points on a 2-dimensional plane. The **convex hull** of S is the smallest convex region containing S. It is well known that the vertices of the convex hull of S, denoted as $Hull(S)$, is a subset of S. Therefore, for a given set S of points, the **convex hull problem** is to find $Hull(S)$ and to report these points in a clockwise or counterclockwise sequence.

Given two sets S_1 and S_2 of points, suppose $x(S_1) < x(S_2)$. The **upper tangent (lower tangent)** of $Hull(S_1)$ and $Hull(S_2)$ is a straight line such that all the points in $S_1 \cup S_2$ are below (above) the line. For example, as shown in Fig. 1, the lines p_1p_2 and q_1q_2 are the **upper tangent** and the **lower tangent**, respectively. p_1 and p_2 are called the **top points**,

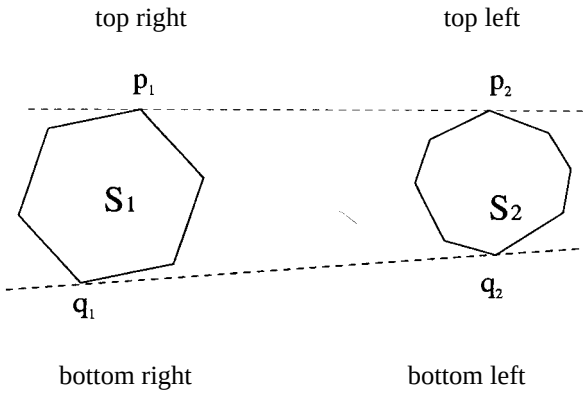


Fig. 1. The tangent lines of two convex hulls.

and q_1 and q_2 are called the **bottom points**. Specifically, $p_1(q_1)$ is the **top (bottom) right point** with respect to S_2 since S_2 is to the right of p_1 and $p_2(q_2)$ is the **top (bottom) left point** with respect to S_1.

Let $S_1, S_2,..., S_p$ be p sets of points in which $x(S_1) < x(S_2) < ... < x(S_p)$. For S_r, $1 \le r \le p$, let $t_1, t_2,..., t_{r-1}$ be the top left points on S_r with respect to $S_1, S_2,..., S_{r-1}$. The point t_i is called the **absolutely top left point** of S_r if the upper tangent corresponding to t_i has the smallest slope among all the upper tangents corresponding to $t_1, t_2,..., t_{r-1}$. An example illustrating the absolutely top left point is shown in Fig. 2. The **absolutely top right point**, **absolutely bottom left point** and **absolutely bottom right point** can also be defined in a similar manner, respectively.

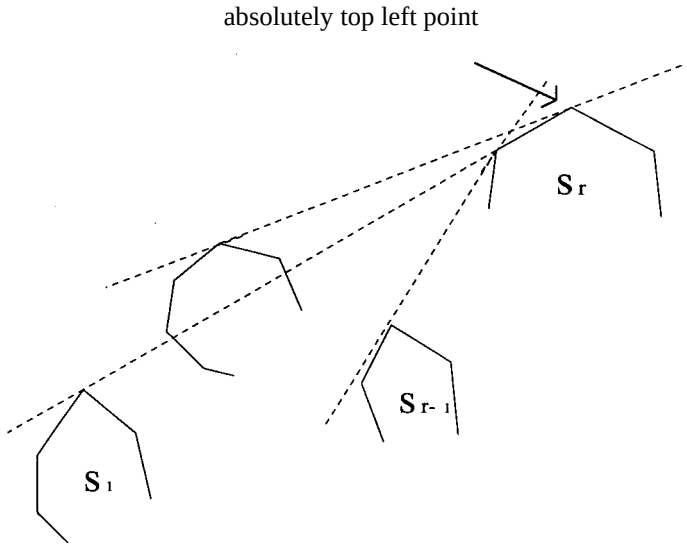


Fig. 2. The absolutely top left point.

Our algorithm for solving the convex hull problem is as follows.

Algorithm Convex-Hull.
Input: A set S of n planar points.
Output: The convex hull of S.

Step 1: Sort the n points in S according to their x-values. Denote the subset of points stored in processor i as S_i. After sorting, $x(S_i) < x(S_j)$ if $i < j$.
Step 2: Each processor i finds $Hull(S_i)$.
Step 3: $i \leftarrow 1$. Repeat the following substeps p times.
Step 3.1: Processor i broadcasts the points in $Hull(S_i)$ one by one according to a clockwise sequence.
Step 3.2: Each processor j, $j \neq i$, merges $Hull(S_i)$ and $Hull(S_j)$ and then deletes the points which are not in $Hull(Hull(S_i) \cup Hull(S_j))$ from $Hull(S_j)$ [10]. After merging is performed, the top point and the bottom point of $Hull(S_j)$ are obtained.

Step 3.3: $i \leftarrow i + 1$.

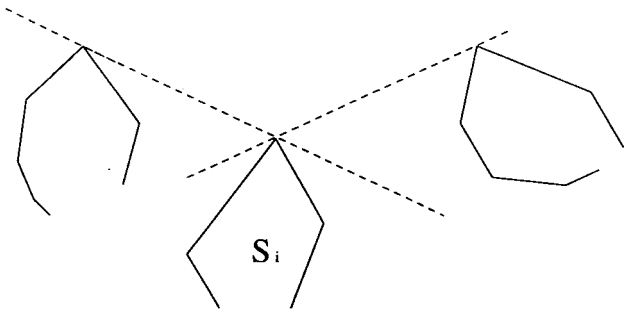
Step 4: Each processor i checks its $Hull(S_i)$. If a point is both a top (bottom) left point and a top (bottom) right point, check its two corresponding tangents. If these two tangents create a reflexive angle, then delete this point from $Hull(S_i)$.

Step 5: Select an interior point for $Hull(S)$. Sort the points remaining in all $Hull(S_i)$'s according to their polar angles with respect to the interior point.

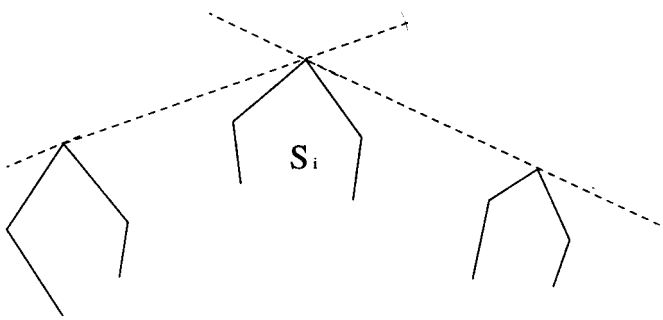
We shall show that all the points remaining after Step 4 are the vertices of the convex hull. Then, our algorithm will find the convex hull after Step 5 is performed. For simplicity, we are only concerned with the properties of the top points. The bottom points have the same properties.

Proposition 1: Let $S_1, S_2, ..., S_p$ be p sets of points in which $x(S_1) < x(S_2) < ... < x(S_p)$. Suppose that $r \in S_i$, r is both the absolutely top left point and the absolutely top right point. $r \notin Hull(S_1 \cup S_2 \cup ... \cup S_p)$ if and only if the two corresponding upper tangents of r create a reflexive angle.

As shown in Fig. 3, it can be easily seen that Proposition 1 is correct.



(a)



(b)

Fig. 3. A point which is both the absolutely top left and top right point. (a) A reflexive angle. (b) Not a reflexive angle.

Proposition 2: Let $S_1, S_2,..., S_p$ be p sets of points in which $x(S_1) < x(S_2) < ... < x(S_p)$. Suppose that $r_1, r_2,..., r_k$, $r_1 \neq r_k$, is a consecutive clockwise subsequence of $Hull(S_i)$ in which r_1 is the absolutely top left point and r_k is the absolutely top right point. Then, $r_1, r_2,..., r_k \in Hull(S_1 \cup S_2 \cup ... \cup S_p)$.

Clearly, as shown in Fig. 4, Proposition 2 is correct.

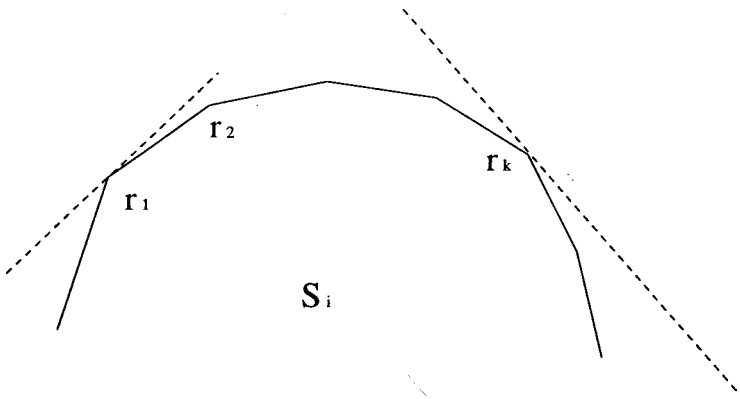


Fig. 4. A part of a convex hull.

In Algorithm Convex-Hull, after Step 3 is executed, the survivors in each S_i are those points fulfilling one of the following conditions. (1). The points are absolutely top and/or bottom points. (2). The points are on the boundary of $Hull(S_i)$ and between the absolutely left point and the absolutely right point. Step 4 deletes the points which are not on the final convex hull. Thus, by Proposition 1 and Proposition 2, our algorithm works correctly.

The time required for each step in our algorithm is as follows:

Step 1: $O(n + \frac{n}{p}\log\frac{n}{p})$ [21];

Step 2: $O(\frac{n}{p}\log\frac{n}{p})$ [10];

Step 3: $O(\frac{n}{p} \cdot p) = O(n)$ [10];

Step 4: $O(\frac{n}{p})$;

Step 5: $O(n + \frac{n}{p}\log\frac{n}{p})$ [21].

Thus, the total time needed by Algorithm Convex-Hull is $O(n + \frac{n}{p}\log\frac{n}{p})$.

Preparata and Hong proved that the lower bound of a sequential algorithm for solving the convex hull problem is $\Omega(n \log n)$ by reducing the 2-D maxima finding problem to it [13].

Lemma 2: Under the broadcast communication model, the lower bound for an algorithm using p, $p \geq 1$, processors to solve the 2-D maxima finding problem with n points is $\Omega(n + \frac{n}{p}\log\frac{n}{p})$.

Proof: Let $u_1, u_2, ..., u_n$ be the n input points, where $u_i = (a_i, b_i)$, $1 \le i \le n$. Suppose that $a_1 < a_2 < ... < a_n$ and $b_1 > b_2 > ... > b_n$. In this case, u_i, $2 \le i \le n$, must be at least compared with u_{i-1}; otherwise, we can not determine whether u_i is a maximum or not. Thus, the 2-D maxima finding problem has the neighboring property. Theorem 1 can be applied. Besides, the lower bound of a sequential algorithm for finding all maxima among n points is $\Omega(n \log n)$ [11], thus this theorem holds. □

With Lemma 2, we obtain the following theorem.

Theorem 3: Under the broadcast communication model, the lower bound for an algorithm using p, $p \ge 1$, processors to find the convex hull of n points is $\Omega(n + \frac{n}{p} \log \frac{n}{p})$.

By Theorem 3, Algorithm Convex-Hull is optimal under the broadcast communication model.

6. THE CLOSEST PAIR PROBLEM

In this section, we shall give an algorithm to solve the closest pair problem [4, 5, 7] under the broadcast communication model.

Let $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ be two points on a 2-dimensional plane. The **distance** between p_1 and p_2, denoted as $Dis(p_1, p_2)$, is defined as $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. For two sets, S_1 and S_2 of points, let $Close(S_1, S_2)$ denote the pair of points p_1 and p_2, $p_1 \in S_1$, $p_2 \in S_2$, such that $Dis(p_1, p_2) \le Dis(p_i, p_j)$, for any $p_i \in S_1$ and any $p_j \in S_2$. The **closest pair problem** is that of finding $Close(S, S)$ for a given set S of points. Our algorithm is as follows.

Algorithm Closest-Pair.
Input: A set S of n planar points.
Output: The closest pair in S.

Step 1: Sort the n points in S according to their x-values. Denote the subset of points stored in processor i as S_i. After sorting, $x(S_i) < x(S_j)$ if $i < j$.

Step 2: Each processor i finds the x-value of the rightmost point in S_i, denoted as r_i. Processor 1 also finds the x-value of the leftmost point in S_1, denoted as r_0. Each processor i broadcasts its r_i sequentially. Each processor i keeps r_{i-1} and r_i.

Step 3: Each processor i finds $Close(S_i, S_i)$. Denote $Dis(Close(S_i, S_i))$ as d_i. Then, each processor i sorts the points in S_i into an ascending sequence according to their y-values.

Step 4: Find the minimum of all d_i's. Denote it as d.

Step 5: $i \leftarrow 1$. Repeat the following substeps p times.

Step 5.1: Processor i broadcasts its r_{i-1}, r_i and its points one by one according to an ascending sequence.

Step 5.2: Each processor j computes $t_j = r_{i-1} - r_j$ if $j < i$ and each processor j computes $t_j = r_{j-1} - r_i$ if $j > i$.

Step 5.3: Each processor j uses the merging technique to find $Close(S_j, S_j \cup S_i)$ if $t_j \le d$. After this substep, $Close(S_j, S_j \cup S_1 \cup S_2 \cup ....\cup S_i)$ is obtained.

Step 5.4: $i \leftarrow i+1$.

Step 6: Find the minimum of all $Dis(Close(S_j, S))$'s and d. The corresponding pair of points are the answer.

This can be easily done in each step of Algorithm Closest-Pair except in Step 5.3. To accomplish Step 5.3, we can use the merging technique proposed by Bentley [5]. The time required for the merging procedure in Step 5.3 is $O(\frac{n}{p})$ since there are $O(\frac{n}{p})$ points in the sets being merged. Also, the time required for Step 3 is $O(\frac{n}{p}\log\frac{n}{p})$ [5]. Thus, the total time required for Algorithm Closest-Pair is $O(n+\frac{n}{p}\log\frac{n}{p})$.

Theorem 4: Under the broadcast communication model, the lower bound for an algorithm using p, $p \ge 1$, processors to find the closest pair of n points is $\Omega(n+\frac{n}{p}\log\frac{n}{p})$.

Proof: The 1-dimensional closest pair problem is a special case of this problem. Furthermore, the 1-dimensional problem has the neighboring property since we have to at least calculate the distance between every pair of neighboring points. Thus, the 2-dimensional problem also has the neighboring property. The lower bound of a sequential algorithm for solving the closest pair problem is $\Omega(n \log n)$ [14]. Combining this fact with Theorem 1, it is clear that this theorem holds. □

With Theorem 4, we conclude that Algorithm Closest-Pair is optimal under the broadcast communication model.

7. CONCLUDING REMARKS

In this paper, under the broadcast communication model, we have proposed a general scheme, the p-division approach, to solve some geometric problems, including the ranking, convex hull and closest pair problems. All of these problems have been considered in a 2-dimensional space. The ranking problem has also been discussed for a k-dimensional space.

These problems possess a common property, the neighboring property. Thus, their lower bounds under the broadcast communication model can be proved using the same technique. Why can they be solved using the p-division approach? Each of them can be solved using a sequential divide-and-conquer method originally. Furthermore, when one problem is divided into more than two subproblems, the sequence of merging subproblems is arbitrary. If one problem has this property, it can also be solved using the p-division approach under the broadcast communication model.

REFERENCES

1. A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing and C. Yap, "Parallel computational geometry," *Algorithmica*, Vol. 3, No. 3, 1988, pp. 293-327.
2. S. G. Akl and G. T. Toussaint, "A fast convex hull algorithm," *Information Processing Letters*, Vol. 7, No. 5, 1978, pp. 219-222.
3. M. J. Atallah, R. Cole and M. T. Goodrich, "Cascading divide-and-conquer: a technique for designing parallel algorithms," *SIAM Journal on Computing*, Vol. 18, No. 3, 1989, pp. 499-532.
4. M. J. Atallah and M. T. Goodrich, "Efficient parallel solutions to some geometric problems," *Journal of Parallel and Distributed Computing*, Vol. 3, No. 4, 1986, pp 492-507.
5. J. L. Bentley, "Multidimensional divide-and-conquer," *Communications of the ACM*, Vol. 23, No. 4, 1980, pp. 214-229
6. R. Dechter and L. Kleinrock, "Broadcast communications and distributed algorithms, " *IEEE Transactions on Computers*, Vol. C-35, No. 3, 1986, pp. 210-219.
7. C. R. Dyer, "A fast parallel algorithm for the closest pair problem," *Information Processing Letters*, Vol. 11, No. 1, 1980, pp. 49-52.
8. P. O. Fjallstrom, J. Katajainen, C. Levcopoulos and O. Petersson, "A sublogarithmic convex hull algorithm," *BIT*, Vol. 30, No. 3, 1990, pp. 378-384.
9. M. T. Goodrich, "Finding the convex hull of a sorted point set in parallel," *Information Processing Letters*, Vol 26, No. 1, 1987, pp. 173-179.
10. R. Graham, "An efficient algorithm for determining the convex hull of a planar set," *Information Processing Letters*, Vol. 1, No. 4, 1972, pp. 132-133.
11. H. T. Kung, F. Luccio and F. P. Preparata, "On finding the maxima of a set of vectors, " *Journal of the ACM*, Vol. 22, No. 4, 1975, pp. 469-476.
12. J. M. Marberg and E. Gafni, "Sorting and selection in multi-channel broadcast networks" in *Proceedings of* 1985 *International Conference on Parallel Processing*, pp. 846-850.
13. F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Communications of the ACM*, Vol. 20, No. 2, 1977, pp. 87-93.
14. F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, 1985, Springer-Verlag, New York.
15. K. V. S. Ramarao, "Distributed sorting on local area networks," *IEEE Transactions on Computers*, Vol. C-37, No. 2, 1988, pp. 239-243.
16. S.-H. Shiau and C. B. Yang, "A fast maximum finding algorithm on broadcast communication," *Information Processing Letters*, Vol. 60, No. 2, 1996, pp.81-89.
17. C. Y. Tang and M. J. Chiu, "Distributed sorting on the serially connected local area networks," in *Proceedings of* 1989 *International Conference on Networks*, Singapore, 1989, pp. 458-462.
18. B. F. Wang and G. H. Chen, "Constant time algorithms for sorting and computing convex hulls," in *Proceedings of International Computer Symposium*, Hsinchu, Taiwan, 1990, pp. 607-612.
19. C. B. Yang, "Reducing conflict resolution time for solving graph problems in broadcast communications," *Information Processing Letters*, Vol. 40, No. 6, 1991, pp. 295-302.

20. C. B. Yang, R. C. T. Lee and W. T. Chen, "Parallel graph algorithms based upon broadcast communications," *IEEE Transactions on Computers*, Vol. C-39, No. 12, 1990, pp. 1468-1472.
21. C. B. Yang, R. C. T. Lee and W. T. Chen, "Conflict-free sorting algorithms under single-channel and multi-channel broadcast communication models," in *Proceedings of International Conference on Computing and Information also in Lecture Notes in Computer Science*, Vol. 497, Ottawa, Canada, 1991, pp. 350-359.

Chang-Biau Yang (楊昌彪) received the BS degree in electronic engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 1982 and the MS degree in computer science from National Tsing-Hua University, Hsinchu, Taiwan, in 1984. Then, he obtained his Ph. D. degree in computer science from National Tsing-Hua University in 1988. He is currently an associate professor of the Department of Applied Mathematics, National Sun Yat-Sen University. His research interests include computer algorithms, interconnectin networks, fault-tolerant computing and data compression.