1985

An Optimal Parallel Algorithm For the All Nearest -- Neighbor Problem for a Convex Polygon

Michael T. Goodrich

Report Number:
85-533

AN OPTIMAL PARALLEL ALGORITHM FOR THE ALL NEAREST-NEIGHBOR PROBLEM FOR A CONVEX POLYGON

Michael T. Goodrich

CSD-TR-533
August 1985

An Optimal Parallel Algorithm For the All Nearest-Neighbor Problem for a Convex Polygon

Michael T. Goodrich

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

August, 1985

CSD-TR-533

Abstract

In this paper we give a parallel algorithm for finding the nearest-neighbor vertex of each vertex of a convex polygon. Our algorithm runs in $O(\log n)$ time using $O(n/\log n)$ processors, in the parallel computation model CREW PRAM (Concurrent-Read, Exclusive-Write Parallel RAM). This implies that the all nearest-neighbors problem for a convex polygon can be solved in $O(n/p + \log n)$ time using p processors, which is optimal.

Keywords: Computational geometry, parallel algorithms, nearest-neighbor problem, convex polygons, parallel merge.

1 Introduction

Problems of a geometric nature arise often in many areas where a fast solution is essential [6]. Thus it seems natural that we should be interested in finding efficient parallel algorithms to solve such problems. Previous work in this area addressed the convex hull problem, the closest pair problem, Voronoi diagrams, segment intersection, and polygon triangulation, among others [1,2,3].

In this paper we give an optimal parallel algorithm solving the All Nearest-Neighbor Problem for a Convex Polygon: given a convex polygon with n vertices, find the nearest-neighbor vertex of each vertex of the polygon. This problem has been studied for the sequential computation model, and an optimal $O(n)$ time algorithm is known [4,5,9]. The fast serial algorithm for this problem makes repeated use of the "plane-sweep" paradigm, in which one sequentially scans through a set of objects, usually updating some data structure with each new object encountered. Since the fast serial algorithm relies on this paradigm, it doesn't seem likely that we can simply "parallelize" the known fast sequential algorithm to get an efficient parallel algorithm.

We present a new parallel algorithm for this problem which runs in $O(\log n)$ time using $O(n/\log n)$ processors. This, of course, implies that the problem can be solved in $O(n/p + \log n)$ time using p processors, which is optimal. The parallel computation model we use is one in which all processors share a common memory, and no two processors may simultaneously write to the same memory cell, but concurrent reads are allowed. This model is usually refered to as a CREW PRAM (Concurrent-Read, Exclusive-Write Parallel RAM).

The algorithm consists of two parts: a partition phase and a merge phase. We present the partition phase in section 2, and the merge phase in section 3. Our algorithm makes repeated use of a parallel merging technique which may be useful in finding efficient parallel algorithms for other geometric problems.

2 The Partition Phase

Let $P = (v_1, v_2, \ldots, v_n)$ be the clock-wise listing of the verticies of some convex polygon P. A vertex chain $C = (v_i, v_{i+1}, \ldots, v_j)$[1] is said to have the *semi-circle* property if (i) v_i and v_j are two farthest vertices in C, and (ii) if all the vertices of C are contained in a circle with diameter $\overline{v_i v_j}$. Let v_a and v_c be two farthest vertices of P, and let v_b (v_d) be a vertex which is farthest to the left (right) of the line $\overline{v_a v_c}$. In [5] Lee and Preparata show that the vertices v_a, v_b, v_c, and v_d partition P into 4 vertex chains with the semi-circle property ($C_1 = (v_a, \ldots, v_b)$, $C_2 = (v_b, \ldots, v_c)$, $C_3 = (v_c, \ldots, v_d)$, and $C_4 = (v_d, \ldots, v_a)$), and that the nearest-neighbor vertex in C_i of any $v_k \in C_i$ is v_{k-1} or v_{k+1} (see Figure 1). In this section we show how to find the vertices v_a, v_b, v_c, and v_d

[1] All subscripts are assumed to be $(\mathrm{mod}\ n) + 1$.

efficiently in parallel. We will show how to use these vertices to solve the all nearest-neighbor problem for P in the section which follows.

We first show how to find a pair of farthest vertices in P. The algorithm DIAMETER, presented below, accomplishes this in $O(\log n)$ time using $O(n/\log n)$ processors.

Algorithm DIAMETER:

Input: A convex polygon $P = (v_1, v_2, \ldots, v_n)$.

Output: Two farthest points, v_a and v_c, in P.

Method: The cyclic ordering of the vertices of P determines a direction for each edge. Treating each edge as a vector, translate this set of edge vectors to the origin. In [7] Shamos observed that any line through the origin of this vector diagram intersects two sectors which correspond to anti-podal vertices (see Figure 2). Since we wish to find all anti-podal vertices in $O(\log n)$ time, we cannot use the method of rotating a line containing the origin through the set of vectors, as Shamos did. Instead, we divide the set of vectors in two by the x-axis, reflect one of subsets about the x and y axes, and use a parallel merging procedure to enumerate all anti-podal vertices. Then, by taking a maximum over the $O(n)$ pairs we find a farthest pair of vertices in P. The details follow.

Step 1. Using the cyclic ordering of the vertices of P as determining a direction on each edge, and treating edges as vectors, translate the set of edge vectors to the origin.

Step 2. Tag the vectors (edges) in the vector diagram which are above the x-axis "red," and tag the remaining vectors "blue."

Step 3. Reflect the set of blue vectors about the y-axis, and then about the x-axis.

Comment: Note that the set of red vectors and the set of blue vectors are now similarly ordered by the angle each vector makes with the x-axis.

Step 4. Use the merging algorithm of Shiloach and Vishkin [8] to merge these two sorted lists in $O(\log n)$ time using $O(n/\log n)$ processors.

Note: Although Shiloach and Vishkin designed their merging algorithm for the CRCW PRAM (Concurrent-Write), their method will work on the CREW PRAM also.

Comment: Now, given any red vector we can find the blue vector which precedes it and the one which succeeds it in $O(1)$ time, and vice versa.

Step 5. For each red vector find the two blue vectors which precede it and succeed it in the merged list. This determines two pairs of anti-podal vertices for each red vector.

Step 6. Repeat Step 5 for blue vectors.

Comment: Steps 5 and 6 enumerate all anti-podal pairs of vertices of P (in fact, some pairs are counted twice).

Step 7. Take the max over all the distances between anti-podal vertices to find a farthest pair of vertices in P. This can be done in $O(\log n)$ time using the familiar $O(n/\log n)$ processor max-finding algorithm, in which each processor finds a maximum of $O(\log n)$ elements sequentially, and then all the processors find the maximum of these values in parallel, using a "binary tree" combining rule.

End of algorithm DIAMETER.

Theorem: Algorithm DIAMETER correctly finds a farthest pair of vertices of a polygon P in $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM.

Proof: The correctness of the algorithm DIAMETER should be clear from the comments made above, so we turn immediately to the time and processor bounds. Note that Steps 1, 2, 3, 5, and 6 could be solved in $O(1)$ time if we were using $O(n)$ processors, since we are doing $O(1)$ work for each of $O(n)$ objects in each of these steps. Thus we can perform each of these steps in $O(\log n)$ time using $O(n/\log n)$ processors. We have already observed that Steps 4 and 7 can be done in $O(\log n)$ time using $O(n/\log n)$ processors, so this completes the proof. ∎

We use the algorithm DIAMETER to help partition P into four vertex chains with the semi-circle property in the algorithm PARTITION presented next. The algorithm PARTITION runs in $O(\log n)$ time using $O(n/\log n)$ processors.

Algorithm PARTITION:

Input: A polygon $P = (v_1, v_2, \ldots, v_n)$.

Output: A partitioning of P into four vertex chains $C_1 = (v_a, \ldots, v_b)$, $C_2 = (v_b, \ldots, v_c)$, $C_3 = (v_c, \ldots, v_d)$, and $C_4 = (v_d, \ldots, v_a)$, each with the semi-circle property.

Method: We use the algorithm DIAMETER to find a farthest pair of vertices in P (v_a and v_c), and then find vertices which are farthest left and right of the line $\overline{v_a v_c}$, respectively. As we have already noted this determines four vertex chains with the semi-circle property.

Step 1. Use the algorithm DIAMETER to find a farthest pair (v_a, v_c) of vertices in P.

Step 2. Find a vertex v_b (v_d) which is a farthest point left (right) of the line $\overline{v_a v_c}$ by taking a max of all the distances of vertices to the line $\overline{v_a v_c}$. This can be done in $O(\log n)$ time using $O(n/\log n)$ processors using the familiar $O(\log n)$ time, $O(n/\log n)$ processor max-finding algorithm. The points $\{v_a, v_b, v_c, v_d\}$ divide P into 4 vertex chains $C_1 = (v_a, \ldots, v_b)$, $C_2 = (v_b, \ldots, v_c)$, $C_3 = (v_c, \ldots, v_d)$, and $C_4 = (v_d, \ldots, v_a)$, each with the semi-circle property.

End of Algorithm PARTITION.

Theorem: The algorithm PARTITION correctly partitions a convex polygon into four vertex chains with the semi-circle property in $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM.

Proof: The correctness of the algorithm PARTITION follows from the lemma by Lee and Preparata [5] which states that the vertices v_a, v_b, v_c, and v_d partition P into four vertex chains with the semi-circle property. The time and processor bounds follow immediately from the observations made above. ∎

In the section which follows we show how to solve the All Nearest-Neighbors Problem for a Convex Polygon, using the algorithm PARTITION.

3 The Merge Phase

The algorithm NEIGHBORS solves the All Nearest-Neighbor Problem for a Convex Polygon in $O(\log n)$ time using $O(n/\log n)$ processors.

Algorithm NEIGHBORS:

Input: A convex polygon $P = (v_1, v_2, \ldots, v_n)$.

Output: An array NN, with $NN(i)$ being the nearest-neighbor vertex of the vertex v_i.

Method: We use the algorithm PARTITION to divide P into four vertex chains which have the semi-circle property. We can solve the all nearest-neighbor problem for each vertex chain in $O(\log n)$ time using $O(n/\log n)$ processors, because the intra-chain nearest neighbor for any vertex in one of these the vertex chains can be determined in $O(1)$ time (just by looking at predecessor and successor vertices). We then marry the solutions to each consecutive pair of vertex chains, in turn. To marry the solutions to two consecutive vertex chains, say C_1 and C_2, we project the points of C_1 and C_2 onto the line which separates them, and merge these two sorted lists. This then allows us to find inter-chain nearest neighbors for any vertex in $O(1)$ time. Thus we can marry the solutions to two consecutive vertex chains in $O(\log n)$ time using $O(n/\log n)$ processors; hence, can solve the All Nearest-Neighbor problem for P in those some bounds. The details follow.

Step 1. Use the algorithm PARTITION to partition P into four vertex chains $C_1 = (v_a, \ldots, v_b)$, $C_2 = (v_b, \ldots, v_c)$, $C_3 = (v_c, \ldots, v_d)$, and $C_4 = (v_d, \ldots, v_a)$, each with the semi-circle property.

Step 2. Solve the ANN problem for each vertex chain in parallel, finding an intra-chain nearest neighbor x_i for each vertex v_i.

Comment: Since each vertex chain has the semi-circle property, each nearest neighbor can be found by simply looking at immediate successor and predecessor vertices.

Step 3. Let L be the line separating C_1 and C_2 (in this case the line through the vertex v_b perpendicular to the line $\overline{v_a v_c}$). Translate all the points of C_1 and C_2 to L (see Figure 3). Let C_1' (C_2') be the set of translations of vertices in C_1 (C_2).

Step 4. Merge the two sorted lists C_1' and C_2' in $O(\log n)$ time with $O(n/\log n)$ processors using the method of [8].

Step 5. For each $v_i' \in C_1'$ find the projections y_i' and z_i' in C_2' which precede and follow v_i' in the merged list.

Comment: Since we merged the two lists in Step 5, this can be done for any $v_i' \in C_1'$ in $O(1)$ time by one processor. Thus, $O(n/\log n)$ processors can perform this step in $O(\log n)$ time.

Step 6. For each i compare the distances from v_i to x_i, y_i and z_i to see which is closer.

Step 7. Repeat Steps 3 through 6 to marry the solutions to C_2 and C_3, C_3 and C_4, and C_4 and C_1, each pair in turn. This finds the nearest neighbor vertex for each $v_i \in P$.

End of Algorithm NEIGHBORS.

Theorem: The algorithm NEIGHBORS correctly finds the nearest-neighbor vertex of every vertex of a convex polygon in $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM.

Proof: The correctness of the algorithm depends heavily on the fact that in marrying the solutions to a consecutive pair of vertex chains C_1 and C_2 we know that for any vertex $v_i \in C_1$ the nearest neighbor of v_i in C_2 must have a projection which is a predecessor or successor in C_2' of v_i' in the merged list. This was proved by Lee and Preparata in [5]. Having observed that, the correctness of the algorithm follows from the above discussion.

To prove the time and processor bounds we observe that Steps 2, 3, and 6 can be done $O(\log n)$ time using $O(n/\log n)$, since each step requires doing $O(1)$ work for each of $O(n)$ objects. We have already observed that Steps 1, 4, and 5 run in $O(\log n)$ time using $O(n/\log n)$ processors. Thus, the entire algorithm runs within these bounds. ∎

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap, "Parallel Computational Geometry," to appear in *Proc. 25th IEEE Symp. on Foundations of Computer Science, 1985.*

[2] M.J. Atallah and M.T. Goodrich, "Efficient Parallel Solutions to Geometric Problems," *1985 Proc. Int. Conf. on Parallel Processing*, St. Charles, IL., pp. 411–417.

[3] A. Chow, "Parallel Algorithms for Geometric Problems," Ph.D. dissertation, Computer Science Dept., University of Illinois at Urbana-Champaign, 1980.

[4] A. Fournier and Z. Kedem, "Comments on the All Nearest-Neighbor Problem for Convex Polygons," *Information Processing Letters*, Vol. 9, No. 3, October 1979, pp. 105–107.

[5] D.T. Lee and F.P. Preparata, "The All Nearest-Neighbor Problem for Convex Polygons," *Information Processing Letters*, Vol. 7, No. 4, June 1978, pp. 189–192.

[6] D.T. Lee and F.P. Preparata, "Computational Geometry—A Survey," *IEEE Trans. on Computers*, Vol. C-33, No. 12, December 1984, pp. 1072–1101.

[7] M.I. Shamos, "Geometric Complexity," *Proc. of 7th Symp. on Theory of Computing, 1975*, pp. 224–233.

[8] Y. Shiloach and U. Vishkin, "Finding the Maximum, Merging, and Sorting in a Parallel Computation Model," *Journal of Algorithms*, Vol. 2, 1981, pp. 88–102.

[9] C.C. Yang and D.T. Lee, "A Note on the All Nearest-Neighbor Problem for Convex Polygons," *Information Processing Letters*, Vol. 8, No. 4, April 1979, pp. 193–194.

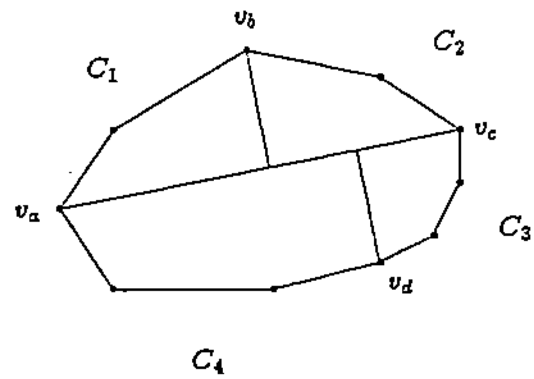

Figure 1: A partitioning of P into 4 vertex chains with the semi-circle property.

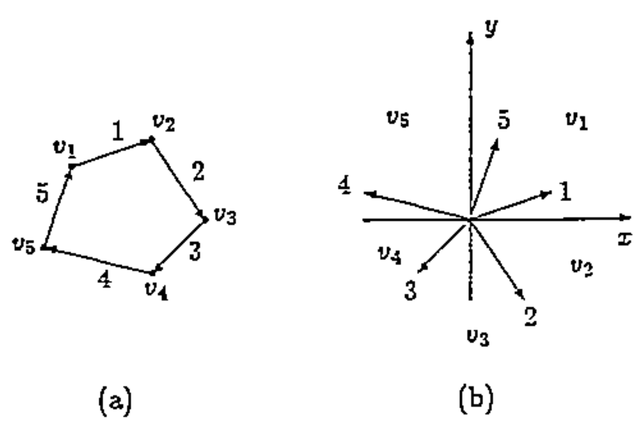

Figure 2: Treating edges as vectors, translate the set of edges to the origin. Note that vertices in the polygon (a) correspond to sectors in the vector diagram (b).

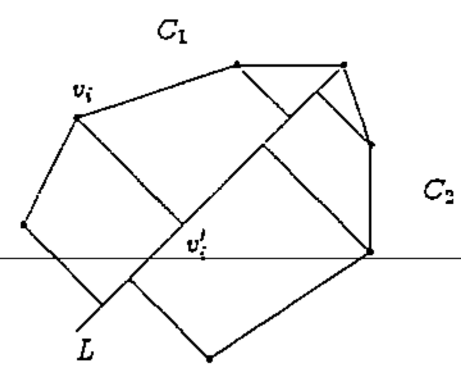

Figure 3: The projection of vertices of consecutive vertex chains onto the line L which separates them.