Parallel Computing 18 (1992) 807–810
North-Holland

PARCO 682

Short communication

Finding congruent regions in parallel *

Laurence Boxer

Department of Computer and Information Sciences, Niagara University, NY 14109, USA

Received 15 October 1990
Revised 18 September 1991, 11 January 1992

Abstract

Boxer, L., Finding congruent regions in parallel, Parallel Computing 18 (1992) 807–810.

Given a straight-line embedded plane graph G of n edges and a polygon P of m edges, $m \leq n$, we describe an algorithm for finding all polygons in G that are congruent to P. Our algorithm requires $\Theta(n \log n)$ time for a CREW PRAM with m processors. This improves upon the $O(n^2)$ time (with m processors) required by the systolic array algorithm of [7]. We also show the problem is in NC by showing how to implement our algorithm in $\Theta(\log n)$ time using mn processors.

Keywords. Systolic array algorithm; finding congruent regions; CREW PRAM; timing analysis.

1. Introduction

In [7], a systolic array algorithm is described to solve the following problem: Given a polygon P in the Euclidean plane and a straight-line embedded plane graph G, find all subpolygons of G that are congruent to P. This is a problem that arises in pattern recognition [4] and in computer vision [1]. The algorithm uses $O(n^2)$ time and $O(m)$ processors, where n and m are the numbers of edges of G and P, respectively. In this paper, we modify the result of [7] by solving the same problem on a CREW PRAM in $\Theta(n \log n)$ time, using m processors. We also show the problem is in NC, i.e. can be solved in polylogarithmic time with a polynomial number of processors, by showing our algorithm can be implemented on a CREW PRAM of mn processors in $\Theta(\log n)$ time. A preliminary version of this paper appears in [2].

The format of the paper is as follows. In Section 2, we discuss the notion of forward connection as presented in [7]. We present our algorithm in Section 3. A summary appears in Section 4.

* Research partially supported by a grant from the Niagara University Research Council.
Correspondence to: L. Boxer, Department of Computer and Information Sciences, Niagara University, NY 14109, USA.

2. Forward connection

We use the terminology of [7]. Let E_1 and E_2 be directed line segments with a common vertex v. Let $A(E_1, E_2)$ be the unique angle formed by a counterclockwise sweep from E_1 to E_2. Suppose P is a polygon whose edges are, in circular order, $c_0, c_1, \ldots, c_{m-1}$. Suppose G is a straight-line embedded plane graph whose edges are $e_0, e_1, \ldots, e_{n-1}$. We say e_i is *connected forwardly* to e_j with respect to c_k and $c_{k+1 \bmod m}$ if and only if $|e_i| = |c_k|$, $|e_j| = |c_{k+1 \bmod m}|$, and $A(e_i, e_j) = A(c_k, c_{k+1 \bmod m})$. A cycle in G formed by the ordered set of edges $\{e_{i_0}, \ldots, e_{i_p}\}$ is a polygon congruent to P if and only if $p = m - 1$ and there is a k such that for each $j = 0, 1, \ldots, m-1$, e_{i_j} is connected forwardly to $e_{i_{j+1 \bmod m}}$ with respect to $c_{k+j \bmod m}$ and $c_{k+j+1 \bmod m}$.

We will use the term *processing element* (*PE*) as a synonym for *processor*. We have the following.

Lemma 2.1 [7]. *Let e_i and e_j be directed edges of G and let k be fixed. Then a single PE can determine in $\Theta(1)$ time whether e_i is connected forwardly to e_j with respect to c_k and $c_{k+1 \bmod m}$.*

3. The algorithm

We assume input consists of the n edges $\{e_0, e_1, \ldots, e_{n-1}\}$ of the straight-line embedded plane graph G and the m edges $\{c_0, c_1, \ldots, c_{m-1}\}$ of the polygon P. Each edge is assumed to be described by the coordinates of its endpoints. These edges are not assumed to be directed; edge orientations will be introduced into the algorithm in order to be able to recognize congruences regardless of whether a reflection of P is required to match equal angles for forward connections (see *Fig. 1*). The edges of P are assumed to be cyclically ordered. We may assume without loss of generality that $m \leq n$, since there is no congruence to determine if $m > n$.

The $O(n^2)$ time required for the systolic array algorithm of [7] is dominated by steps in which, in the worst case, every edge of G examines, serially, every other edge of G in a search for forward connections. Our algorithm uses sorting to eliminate some comparisons, as well as the CREW PRAM's greater flexibility for parallelism as compared with the systolic array, in order to obtain faster running times. We describe the algorithm in the following steps:

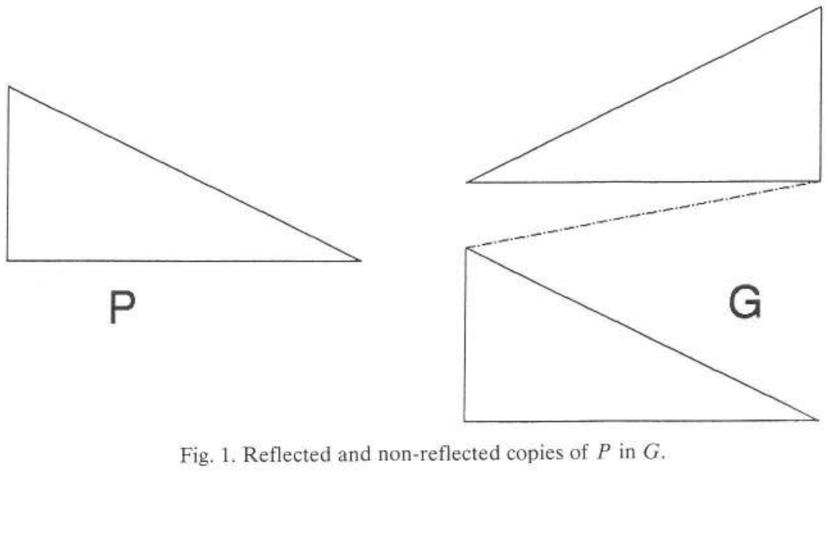

Fig. 1. Reflected and non-reflected copies of P in G.

1. Let a_j and b_j be the endpoints of e_j. Create two records for each of edge e_j with the following components:
   - *Index*, both with value j;
   - *Length*, both with value equal to $|e_j|$.
   - *End1*, with value a_j in one of the records and b_j in the other;
   - *Slope*, the slope of e_j (∞ if e_j is vertical).

   This step can be accomplished by m PEs in $\Theta(n/m)$ time, or by n PEs in $\Theta(1)$ time.
2. Sort the $2n$ records created in the previous step with respect to the keys Length, x-coordinate of End1, y-coordinate of End1, and Slope in decreasing order of importance. This step can be accomplished in $\Theta(n \log n)$ serial time, or by n PEs in $\Theta(\log n)$ time [3]. Let $Edge[0:2n-1]$ be the sorted list of records.
3. In parallel, for each $i = 0, \ldots, m-1$, PE_i conducts a binary search through the list *Edge* to determine $first_i$ and $last_i$, respectively the first and last indices j such that $Edge[j].Length = length(c_i)$. If no such index j exists, let $first_i = 0$, $last_i = -1$. This step requires $O(\log n)$ time.
4. In parallel, each PE_i, $i = 0, \ldots, m-1$, determines $count_i = last_i - first_i + 1$, which is twice the number of edges in G whose length equals that of c_i. This step requires $\Theta(1)$ time.
5. Compute $M = \min\{count_0, count_1, \ldots, count_{m-1}\}$ and note an index κ such that $count_\kappa = M$. If $M = 0$, halt, as there cannot be any congruences. Note we must have $M \le 2n$. This step requires $\Theta(\log m)$ time for m PEs.
6. Let $FC[0:2n-1, 0:m-1]$ be an array defined by

$$FC[i,k] = \begin{cases} j & \text{if } e_i \text{ is connected forwardly to } e_j \\ & \text{with respect to } c_k \text{ and } c_{k+1 \bmod m}; \\ \text{undefined} & \text{otherwise} \end{cases}$$

   It follows from Lemma 2.1 that $2mn$ serial binary searches of the array *Edge* can determine the entries of the array *FC* in $\Theta(n \log n)$ time using m PEs or in $\Theta(\log n)$ time using mn PEs.
7. In this step, we build descriptions of congruences. At the end of this step, we have a list *Cong* of $O(M)$ records with the following fields:
   - A circularly ordered list of edges of G whose union is congruent to P;
   - *Min_Index*, the unique minimal *Index* among the edges of the congruence;
   - *Adjacent_indices*, a sorted pair of indices of the edges adjacent to e_{Min_Index} in the congruence.

   The members of *Cong* are constructed as follows. Each $Edge[i]$, $i = first_\kappa, first_\kappa + 1, \ldots, last_\kappa$, can use the array *FC* to determine if $e_{Edge[i].Index}$ is an edge of a subpolygon S of G such that S and P are congruent, with $e_{Edge[i].Index}$ corresponding to c_κ. Using standard pointer-doubling techniques, this step can be done by m PEs in $O(M \log m) = O(n \log m)$ time and by $Mm = O(mn)$ PEs in $O(\log m)$ time.
8. If P has distinct edges of equal length, it is possible that we have duplicate markings of the same subpolygon of G. The final steps of the algorithm allow us to eliminate such duplication. Sort the $O(M)$ *Cong* records with respect to their *Min_Index* and *Adjacent_indices* fields in decreasing order of importance. This step can be performed in serial $O(n \log n)$ time or by $M = O(n)$ PEs in $O(\log M) = O(\log n)$ time [3].
9. Use a prefix computation to determine which of the ordered *Cong* records has the same *Min_Index* and *Adjacent_indices* fields as one of its predecessors in the ordered list. Mark the corresponding congruences as duplicates. This step can be performed in serial $O(n)$ time or by $M = O(n)$ PEs in $O(\log n)$ time [6].

It follows that the worst-case running times are $\Theta(n \log n)$ when m PEs are used, and $\Theta(\log n)$ time when mn PEs are used.

Note that when m PEs are used, the sort steps can be done faster [5] than the serial time given above, as can the prefix step, but the running time would still be dominated by the $\Theta(n \log n)$ step in which the entries of FC are computed.

4. Summary

We have given an algorithm for finding all subpolygons of a straight-line embedded plane graph G that are congruent to a given planar polygon P. The algorithm runs in $\Theta(n \log n)$ time on a CREW PRAM of m processors, which improves upon the running time of $O(n^2)$ for the systolic array algorithm of [7]. The algorithm can also be implemented on a CREW PRAM of mn processors in $\Theta(\log n)$ time.

The more general problem of finding all subpolygons in G that are *similar* to P may be solved by minor modification of the algorithm given above. Instead of requiring matched edges to have equal length, we require that the analog of 'forward connection' require matched pairs of edges to have lengths of equal proportions. Thus, the problem of finding all subpolygons in G that are similar to P may be solved within the same resources as discussed above. This observation is made in [7].

The optimality of the algorithm we have presented is an open question. The resource bound for our algorithm is $\Theta(mn \log n)$. On the other hand, it is possible that G consists of $\Theta(n/m)$ components, each of which is congruent to P. In this case, any solution to our problem would be required to produce a list of m ordered edges for each of the $\Theta(n/m)$ components of G. Thus, any algorithm that solves our problem must have a worst-case resource bound of $\Omega(n \log m)$.

Acknowledgement

Several suggestions from the anonymous referees were used to improve the presentation of this paper.

References

[1] D.H. Ballard and C.M. Brown, *Computer Vision* (Prentice-Hall, Englewood Cliffs, NJ, 1982).

[2] L. Boxer, An improved parallel algorithm for finding congruent polygons, *Proc. 7th Israeli Conf. on Artificial Intelligence and Computer Vision* (1990) 325–328.

[3] R. Cole, Parallel merge sort, *SIAM J. Comput.* 17 (4) (1988) 770–785.

[4] H. Dirilten and T.G. Newman, Pattern matching under affine transformation, *IEEE Trans. Comput.* 26 (3) (1977) 314–317.

[5] X. Guan and M.A. Langston, Time-space optimal parallel merging and sorting, *IEEE Trans. Comput.* 40 (1991) 596–602.

[6] C.P. Kruskal, L. Rudolf and M. Snir, The power of parallel prefix, *Proc. 1985 Internat. Conf. on Parallel Processing* (1985) 180–185.

[7] Z.C. Shih, R.C.T. Lee and S.N. Yang, A parallel algorithm for finding congruent regions, *Parallel Comput.* 13 (1990) 135–142.