A PARALLEL CIRCLE-COVER MINIMIZATION ALGORITHM *

Laurence BOXER
Department of Computer and Information Sciences, Niagara University, Niagara University, NY 14109, U.S.A.

Russ MILLER
Department of Computer Science, 226 Bell Hall, State University of New York, Buffalo, NY 14260, U.S.A.

2. Preliminaries

In this section, we review the CREW PRAM, some notation from [1,5] that is necessary for the description of our algorithm, as well as the parallel prefix and list ranking operations. It should be noted that the majority of fundamental results presented in [1,5] are not needed in this paper.

2.1. CREW PRAM

Briefly, a concurrent read, concurrent write parallel random access machine (CREW PRAM) may be described as a single instruction stream, multiple data stream (SIMD) machine (i.e., all processors execute the same instructions at the same time, but on potentially different data) consisting of a set of processors and a global memory, where concurrent reads from the same memory location are permitted but where concurrent writes to the same memory location are prohibited.
2.2. Array notation

Throughout this paper, elements of a set or array of size \( n \) will be indexed from 0 to \( n - 1 \).

2.3. Assumptions

Throughout the paper, we will make the following assumptions:

- We are given a set \( S = \{ A_0, A_1, \ldots, A_{n-1} \} \) of \( n \) arcs of the circle centered at the origin of the Euclidean plane with unit radius.
- Every arc \( A_i \) is nondegenerate, that is, contains more than one point.
- No single arc \( A_i \) covers the circle.
- No pair of distinct arcs have the same first or the same second endpoints. (A pair of arcs with the same first or the same second endpoints would be a pair in which one arc contains the other, so the smaller of the pair may be ignored.)
- No arc \( A_i \) has the point \( R \) whose cartesian coordinates are \((1, 0)\) as an endpoint.

There is no loss of generality in making these assumptions, since the negation of each of these conditions can be handled quite simply in the allotted time.

2.4. Arc notation

Given a set \( S \) of \( n \) circular arcs, \( A_i \in S \) is specified by the ordered pair \((x_i, y_i)\), where \( x_i \) and \( y_i \) are endpoints of arc \( A_i \), and \( y_i \) appears after \( x_i \) in the counterclockwise traversal of \( A_i \). The endpoints \( x_i \) and \( y_i \) will be represented by angular coordinates in radian measure with \( 0 < x_i < 2 \pi \), \( x_i < y_i < x_i + 2 \pi \), and \( y_i \neq 2 \pi \). We say \( A_j \) extends \( A_i \) if \( A_j \) contains \( y_i \), the second endpoint of \( A_i \). Observe that if \( A_j \) extends \( A_i \), then either \( x_j \leq y_i < x_j + 2 \pi \) or \( x_j + 2 \pi < y_i \). Define \( A_j \) to be the successor of \( A_i \), denoted \( \text{succ}(A_i) \), if \( A_j \) extends \( A_i \) and the subarc of \( A_j \) from \( y_i \) to \( y_j \) has maximal length among the members of \( \{ A_k | A_k \text{ extends } A_i \} \). \( A_k \) is the subarc of \( A_k \) from \( y_i \) to \( y_k \), \( k = 0, 1, \ldots, n - 1 \). We say \( j = \text{succ}_{\text{ind}}(A_i) \) if \( A_j = \text{succ}(A_i) \).

2.5. Parallel prefix

Given \( n \) items \( a_0, a_1, \ldots, a_{n-1} \), and a \( \Theta(1) \) time computable associative binary operation \( \otimes \),

the initial prefix problem is to compute all \( n \) initial prefixes \( a_0 \otimes a_1, \ldots, a_0 \otimes a_1 \otimes \ldots \otimes a_{n-1} \). When solved on a parallel machine, the initial prefix problem is known as parallel prefix. The parallel prefix algorithm we use is given in [4]. It assumes that processor \( P_i \) is initially responsible for \( a_i \), and that when the algorithm terminates \( P_i \) will know the \( i \)th prefix \( a_0 \otimes a_1 \otimes \ldots \otimes a_i \). The algorithm, which runs in \( \Theta(\log n) \) time on an \( n \) processor CREW PRAM, follows.

FOR \( j = 0 \) TO \( \lfloor \log n \rfloor - 1 \) DO
FORALL \( i \in \{ 2^j, 2^j + 1, \ldots, n - 1 \} \) DO IN PARALLEL
\[ \text{New}A_i := A_{i - 2^j} \otimes A_i \]
END FORALL;
FORALL \( i \in \{ 2^j, 2^j + 1, \ldots, n - 1 \} \) DO IN PARALLEL
\[ A_i := \text{New}A_i \]
END FORALL
END FOR

2.6. List ranking

Suppose we are given a partially ordered list of \( n \) elements represented by a contents array \( C \) and a next index array \( \text{next} \), where \( C_i \) is the value of the element at the \( i \)th location and \( \text{next}_i \) is the location of the element that follows \( C_i \) in the list. The list ranking operation finds for each element \( C_i \) the number of items in the list that follow \( C_i \). This result is stored in \( C_i.\text{counter} \). We say that \( C_i \) is a terminal element if it has no successor, in which case \( \text{next}_i = 1 \). Initially, \( C_i.\text{counter} = 0 \) if \( C_i \) is a terminal element, and \( C_i.\text{counter} = 1 \) otherwise. With these assumptions, the list ranking problem can be solved on a CREW PRAM in \( \Theta(\log n) \) time [3,6] by the following pointer doubling algorithm.

FOR \( \lfloor \log n \rfloor \) iterations DO
FORALL \( i \in \{ 0, \ldots, n - 1 \} \) DO IN PARALLEL
\[ C_i.\text{counter} := C_i.\text{counter} + C_{\text{next}_i}.\text{counter} \]
\[ \text{next}_i := \text{next}_{\text{next}_i} \]
END FORALL;
END FOR
3. Algorithm

In this section we present an optimal time and processor algorithm to solve the circle-cover minimization problem on a CREW PRAM. Steps 1–6 find the successor for each arc, step 7 determines whether or not any cover exists, and steps 8–10 use the successor information to determine a minimal cover.

(1) For each processor \( P_i \), initially responsible for arc \( A_j \), create the following two records: \((x_i, y_i, i, \infty)\) and \((y_j, y_j, i, \infty)\).

(2) Using the algorithm of Cole [2], sort all \( 2n \) records together by the first field in \( \Theta(\log n) \) time, with ties broken in favor of maximum value of the second field.

(3) A \( \Theta(\log n) \) time parallel prefix operation, using the max operator, is performed on the second fields of the \( 2n \) ordered records. If the \( j \)th sorted record is a \((y_j)\) record, the prefix \( a_0 \otimes a_1 \otimes \ldots \otimes a_j \) is then the maximum \( y_k \) such that \( x_k \leq y_k \leq y_j \). Results (indices \( k \) of the arcs yielding the maximal \( y_k \)) are stored in the fourth field of each record. In particular, if \( A_k = \text{succ}(A_j) \) and \( x_k \leq y_k \), then the fourth field of the \((y_j)\) record now has value \( k \).

(4) Since the data is circular, an arc \( A_j \) that contains the point \( R \), whose angular coordinate is \( 2\pi \), may have successor \( A_k \) such that \( x_k \leq y_k - 2\pi \). Notice that this successor would not have been found during step 3. To alleviate this situation, repeat steps 1–3 with \( 2\pi \) subtracted from the endpoints in all records representing arcs containing \( R \).

(5) Sort the \( 4n \) records in the union of the two lists created in steps 3 and 4 with respect to the third field in \( \Theta(\log n) \) time.

(6) In \( \Theta(1) \) time, all processors \( P_i \) can simultaneously determine \( \text{succ} \_\text{ind}(A_j) \) as follows. Let \( L \) be the ordered list of \( 4n \) records created in the previous step. The \((y_j)\) and \((y_j - 2\pi)\) records are the two records of \( L_{4i}, L_{4i+1}, L_{4i+2}, L_{4i+3} \) with identical first and second fields. By examining the arcs corresponding to the fourth entries of these two records, \( \text{succ} \_\text{ind}(A_j) \) may be determined.

(7) There is an arc that is its own successor if and only if \( \{A_0, \ldots, A_{n-1}\} \) is not a covering of the circle. In \( \Theta(\log n) \) time, it can be determined if an arc that is its own successor exists. If so, halt.

(8) Any minimal cover must contain an arc that contains the point \( R \) whose angular coordinate is \( 2\pi \). Thus it suffices to determine the minimum number of arcs required for an arc containing \( R \) to wrap around on itself. We first unwind the circular list to create a linear list, and then use a modification of the \( \Theta(\log n) \) time list ranking operation given in Section 2.6 to find the minimum number of arcs required for each arc containing \( R \) to wrap around on itself. The algorithm follows.

(a) Let \( Q = \{A_j | R \subseteq A_j \} \). Observe that for any index \( i \) we have \( A_i \in Q \) if and only if \( x_i < 2\pi < y_i \).

(b) \{Comment: Create the sink record for the linear list being created.\} In \( \Theta(1) \) time, processor \( P_0 \) creates a record \( c_\infty = (0, \infty, 0) \) whose components are named \( \text{counter, ext, and lastx} \), respectively.

(c) \{Comment.\} The arcs that contain \( R \) will appear twice in the list, once as sources, and once as last elements which point to the sink \( c_\infty \). This step creates the set of last elements.\} If \( A_i \in Q \), \( P_i \) creates the record \( c_{i+n} = (1, \infty, 0) \). The components are named as in (b). This step takes \( \Theta(1) \) time.

(d) \{Comment.\} This step completes the unwinding, using the original set of arcs in \( Q \) as the sources. Notice that care needs to be taken so that arcs that point to members of \( Q \) now point to the corresponding last element that was created in the previous step.\} In \( \Theta(1) \) time, every processor \( P_i \) creates a record \( c_i = (1, \text{ext}, 0) \), where

\[
\text{ext} = \begin{cases} \text{succ}\_\text{ind}(A_j) + n & \text{if } \text{succ}(A_j) \in Q; \\ \text{succ}\_\text{ind}(A_j) & \text{otherwise.} \end{cases}
\]

The first component, \( \text{counter} \), of these records serves as the counter for the list ranking operation. The second component, \( \text{ext} \), describes the "extension" of the arc \( A_j \) and represents the successor (in the linear list) as needed in list ranking. The component \( \text{lastx} \) is used in the modified list ranking operation to record the first endpoint of
the final arc (a member of $Q$) that extends the next-to-last arc in a minimal chain from $A_i$ to a member of $Q$.

(e) Suppose $j = \text{succ}_\text{ind}(A_i)$ and $A_j \in Q$. It may be that a cover starting with $A_k$ that includes $A_i$ can complete the covering with $A_k$, rather than $A_j$. Notice that this would reduce the number of arcs in a minimal covering starting with $A_k$ by 1. Thus, for each arc $A_i$ whose successor is in $Q$, we want to use the last arc that extends $A_i$ past $R$ in the modified list ranking operation. This information will be used to determine if the number of arcs in a cover can be reduced by 1. To determine the first element of the last arc that extends $A_i$ past $R$, perform the following.

(i) In $\Theta(\log n)$ time, sort $\{x_m | A_m \in Q\}$ into increasing order.

(ii) In $\Theta(\log n)$ parallel time, if $\text{succ}(A_i) \in Q$ then $P_i$ performs a binary search among the values $x_m$ ordered above to determine

$$c_{i, \text{lastx}} = \max\{x_m | A_m \in Q, x_m \leq y_i\}.$$ 

Then $c_{i, \text{lastx}}$ is the angular coordinate of the first endpoint of the last arc that extends $A_i$ beyond $R$.

(iii) {Comment: Notice that if $\text{succ}(A_i) \notin Q$ then $c_{i, \text{lastx}}$ remains set to 0.}

(f) In $\Theta(\log n)$ time, perform a list ranking operation as follows.

FOR [log n] iterations DO
    FOR ALL $i \in \{0,\ldots, n - 1\}$ DO IN PARALLEL
        \[ \text{next}_i := c_{i, \text{ext}}; \]
        \[ c_{i, \text{counter}} := c_{i, \text{counter}} + c_{\text{next}_i, \text{counter}}; \]
        \[ c_{i, \text{ext}} := c_{\text{next}_i, \text{ext}}; \]
        \[ c_{i, \text{lastx}} := \max\{c_{i, \text{lastx}}, c_{\text{next}_i, \text{lastx}}\}; \]
    END FOR ALL
END FOR

(9) If $A_i \in Q$ and $c_{i, \text{lastx}} \geq x_i$ then decrease $c_{i, \text{counter}}$ by 1. This is because $c_{i, \text{lastx}} \geq x_i$ implies $A_i$ extends the next-to-last arc of the list starting with $A_i$ beyond $R$, hence the last arc of this list has been unnecessarily counted in $c_{i, \text{counter}}$. This step requires $\Theta(1)$ parallel time.

(10) In $\Theta(\log n)$ time, determine $\min\{c_{i, \text{counter}} | A_i \in Q\}$, as well as the associated arc $A_i$, breaking ties arbitrarily.

Therefore, in $\Theta(\log n)$ time the minimum number of arcs needed to cover the circle is determined. Further, a simple postprocessing $\Theta(\log n)$ time algorithm will mark one such set of arcs, if desired.

4. Conclusion

This paper improves on the CREW PRAM algorithm of Bertossi [1] to solve the minimal circle-cover problem by giving an optimal $\Theta(\log n)$ time $n$ processor algorithm. Define the work performed by a CREW PRAM to be the product of the time and number of processors. Therefore, the work performed by our algorithm is $\Theta(n \log n)$, which is optimal [5].

Acknowledgment

The authors would like to thank the anonymous reviewer(s) for comments that improved the presentation and quality of this paper.

References