Optimal Randomized Parallel Algorithms for Computational Geometry

Pat Morin

Abstract

A summary of the results achieved in the paper "Optimal Randomized Parallel Algorithms for Computational Geometry," by Reif and Sen is given. These results include optimal randomized PRAM algorithms for the construction of Kirkpatrick's point location structure, and the construction of a data structure called a nested plane sweep tree. The plane sweep tree is used to solve a number of popular computational geometry problems optimally. All results hold with high probability.

1 Introduction

Parallel algorithms for computational geometry problems have received much attention from the algorithms community in recent years. Many of these algorithms make use of randomization to make the problems tractable.

In [?] Reif and Sen present a number of randomized algorithms for computational geometry problems which run in logarithmic time using a linear number of processors. Most of these algorithms are based on a data structure which the authors call a *nested plane sweep tree*. The goal of this work is to provide a simplified treatment of the results obtained in Reif and Sen's paper. All these algorithms run on the CREW PRAM.

The remainder of the paper is organized as follows: Section ?? reviews Kirkpatrick's point location data structure and presents a parallel algorithm for building this structure in logarithmic time. Section ?? reviews the well known segment tree data structure and its parallel counterpart the plane sweep tree. Section ?? defines the nested plane sweep tree and presents a parallel algorithm for its construction. Section ?? shows how the nested plane sweep tree can be used to solve a number of popular geometric problem in logarithmic time. Finally, Section ?? concludes and presents some open problems in this area.

2 Parallelizing Kirkpatrick's Algorithm

Recall Kirkpatrick's algorithm [?] for point location in a triangulated planar straight line graph. The algorithm proceeds in rounds. During each round, a maximal independent set of bounded degree vertices is chosen and deleted. The "holes" created by these deletions are then retriangulated. Kirkpatrick showed that the size of this set is a constant fraction of the total number of vertices in the graph. Thus, after $O(\log n)$ rounds, the graph is reduced to a constant number of vertices. Finding the independent set takes time linear in the current number of vertices, yielding linear time algorithm.

In order to create an $O(\log n)$ time parallel algorithm, we need to reduce the time needed to choose the independent set to $O(1)$. To achieve this we use a generalization of a technique known as *random mate*. The algorithm for choosing our independent set works as follows:

Algorithm Choose-Independent-Set:

1. Assign a sex (male/female) with equal probability to each vertex with degree $\leq d$. The males are the candidates for an independent set.
2. For each edge, if it connects two male vertices, then mark both vertices as dead.
3. The independent set consists of the set of all living male vertices.

Clearly this method produces an independent set of vertices and runs in constant time. The question which remains is to determine whether the size of the independent set is a constant fraction of the number of vertices in the graph. To see this we consider the following. It can be shown that there exists an independent set, S of degree $\leq d$ and of size αn, $0 < \alpha < 1$, for which the shortest path between any two vertices in S is of distance 3. With high probability, roughly half of the vertices of S are male. The probability of any particular male surviving is at least $\frac{1}{2^d}$ since it has at most d neighbours. Since d is a constant, this probability is a constant. Also, the probability of a given male in S surviving is independent of any other male in S surviving. Thus we can apply Chernoff bounds to show that the number of surviving males is $\tilde{O}(n)$

This yields an $\tilde{O}(\log n)$ algorithm for constructing Kirkpatrick's point location structure on a triangulated planar straight line graph using $O(n)$ processors. The number of processors can be reduced to $O(\frac{n}{\log n})$ using the well known technique of processor scheduling, thereby making the algorithm work-optimal.

This work also has applications to the construction of Voronoi diagrams. Using concurrent read capability, n queries can be executed on a planar subdivision of size n in $\tilde{O}(\log n)$ time using $O(n)$ processors. In [?], an algorithm is given for constructing the Voronoi diagram of n points in the plane in $O(\log^3)$ time using $O(n)$ processors. This algorithm consists of $O(\log n)$ rounds in which each round consists of performing point n point location queries. Thus using the scheme presented above, the running time of the Voronoi diagram construction algorithm can be improved to $\tilde{O}(\log^2 n)$.

-bb-error = =

Figure 1: A Segment Tree

3 Segment Trees

The segment tree is a data structure used to store 2D non-intersecting line segments. It has the following properties:

- It is a complete binary tree on the ranges defined by the projection of the $2n$ endpoints of the segments on the x axis.
- The range of a parent node is the union of the range of its two children.
- Associated with a node, v, is a *catalogue*, $C(v)$ containing the list of segments which span the node's range, but do not span the range of the node's parent.
- The catalogues are sorted by the *below* relationship. That is, if we shoot a ray vertically so that it intersects all segments in a catalogue, the segments are ordered by the order in which they are struck by the ray.

Figure **??** shows a set of segments and its associated segment tree. The main type of query handled by a segment tree is the *next element search* query. Next element search queries take the following form: Given a point p, which segment lies directly above (or below) p. This query can be answered by a segment tree in $O(\log n)$ time, where n is the number of segments in the tree.

In [?] a parallel variation on the segment tree called a *plane sweep tree* is described. A plane sweep tree is essentially a segment tree augmented with some additional information. A parallel plane sweep tree construction algorithm exists which runs in $O(\log^2 n)$ time using $O(n)$ processors. The same algorithm can construct a plane sweep tree in $O(\log n)$ time using $O(n^{1+\epsilon})$, $\epsilon > 0$, processors. We mention the plane sweep tree since it is the building block of the nested plane sweep tree presented in the next section.

4 The Nested Plane Sweep Tree

The nested plane sweep tree can be defined recursively as follows:

Algorithm Build-NPST:

1. Construct a plane sweep tree from a sample of $\sqrt{n}$ segments. This can be done in $O(\log n)$ time using $O(n)$ processors.
2. The trapezoidal decomposition of these segments partition the plane into $O(\sqrt{n})$ trapezoids.
3. For each of these trapezoids, recursively construct a nested plane sweep tree for all the remaining segments which intersect it.

In order for the nested plane sweep tree to be efficient, we need to choose our sample segments so that no trapezoid intersects more than $O(\sqrt{n}\log n)$ segments. In this case the running time is $T(n,n) = T(\sqrt{n}, \sqrt{n}\log n) + O(\log n) = O(\log n)$. It can be shown that with probability $\geq \frac{1}{2}$, a randomly chosen set of line segments will satisfy this condition. Therefore, if we choose $c\log n$ sets of segments, then the probability of at least one of them satisfying this condition is $1 - \frac{1}{n^c}$.

The difficulty with this approach is that testing whether a set of segments satisfies this condition takes $O(\log n)$ time and $O(n)$ processors (we must perform $O(n)$ queries on a nested plane sweep tree of size $O(\sqrt{n})$). Thus to test all $O(\log n)$ sets would take $O(\log^2 n$ time.

Luckily, we can test a set of segments by using random sample of size $O(\frac{n}{\log n})$. If X is the number of segments which intersect a given trapezoid, then we can show using Chernoff bounds that the number these segments in our sample is $\tilde{O}(\frac{Xn}{\log n})$. Thus the sample selection algorithm proceeds as follows:

Algorithm Sample-Select:

1. Choose $O(\log n)$ sets of samples, where each set has $\sqrt{n}$ segments in it.
2. Partition the processors into $O(\log n)$ groups of size $O(\frac{n}{\log n})$.
3. Have each group of processors test a set of samples using $O(\frac{n}{\log n})$ randomly selected segments.

Step 1 and Step 2 of the algorithm take constant time. Step 3 takes $O(\log n)$ time since each processor executes $O(1)$ queries on a plane sweep tree of size $\sqrt{n}$. Thus the time to find a good sample is $O(\log n)$ and the nested plane sweep tree can be constructed in $\tilde{O}(\log n)$ time.

Like the segment tree, the nested plane sweep tree can be used to answer next element search queries. The time to answer n next element search queries on a nested plane sweep tree of size n is $\tilde{O}(\log n)$ using $O(n)$ processors. In the following section, we show how this can be used to solve a number of geometric problems.

5 Applications of the Nested Plane Sweep Tree

In this section we discuss applications of the nested plane sweep tree discussed in the previous section. The discussion begins with problem related to trapezoidation, and proceeds to dominance problems in 2 and 3 dimensions.

5.1 Trapezoidal Decomposition of a Simple Polygon

Using the nested plane sweep tree, trapezoidal decomposition of a simple polygon can be done in $\tilde{O}(\log n)$ time using $O(n)$ processors. The trapezoidation algorithm works as follows:

Algorithm Trapezoidal-Decomposition:

1. Build the nested plane sweep tree for the edges of the polygon.
2. Locate the edge directly above and below each vertex of the polygon.
3. Determine whether the vertical rays shot in the previous step are inside or outside the polygon.
4. Rays inside the polygon should be included as edges in the trapezoidation, rays outside the polygon should not.

5.2 Triangulation of a Simple Polygon

Using a nested plane sweep tree, triangulation of a simple polygon can be done in $O(\log n)$ time using $O(n)$ processors. The triangulation algorithm proceeds as follows:

Algorithm Triangulate:

1. Compute a trapezoidal decomposition of the polygon using the nested plane sweep tree.
2. Decompose the polygon into one-sided monotone polygons using the trapezoidal decomposition (for details see [?]).
3. Triangulate each of the one sided monotone polygons in linear time using a linear number of processors (for details see [?])

5.3 Visibility from a Point

The visibility polygon of a point can be computed in $\tilde{O}(\log n)$ time using $O(n)$ processors. This can be accomplished by applying a standard transformation and computing the lower

-bb-error = =

Figure 2: Computing the lower envelope

-bb-error = =

Figure 3: The 2-dimensional maxima problem

envelope of line segments. The lower envelope algorithm is illustrated in Figure ?? and proceeds as follows:

Algorithm Lower-Envelope:

1. Sort the endpoints of the segments on their x-coordinate, creating $2n-1$ intervals.
2. For each interval, I, determine which segment is visible by performing a next element query of the form $(mid_x(I), -\infty)$.

5.4 Maxima Problems

The d-dimensional maxima problem takes as input a set of points in $\Re^d$ and determines the subset of these which are not dominated on all dimensions by any other point in the set. Figure ?? illustrates an example of the 2-dimensional maxima problem.

5.4.1 The 2-D Maxima Problem

The 2-D maxima problem can be solved using the nested plane sweep tree in $\tilde{O}(\log n)$ time using $O(n)$ processors. The algorithm is illustrated in Figure ?? and proceeds as follows:

Algorithm 2D-Maxima:

1. Build the nested plane sweep tree on the segments $(0, y_i, x_i, y_i)$, where the (x_i, y_i) are the positions of the input points.
2. For each point, determine if it is a maxima by determining whether or not a segment lies above it. If no segment lies above it, it is a maxima.

-bb-error = =

Figure 4: Solving the 2-dimensional maxima problem using the nested plane sweep tree.

-bb-error = =

Figure 5: The marked point could be a maxima if its z coordinate is greater than that of the two points above it.

5.4.2 The 3-D Maxima Problem

A solution to the 3-D maxima problem can be found by noting that a point, p, is a 3-D maxima if none of the segments which lie above p have a larger z coordinate p. This situation is illustrated in Figure ??.

Thus, we can efficiently solve the 3-dimensional maxima problem by augmenting the nested plane sweep tree. More specifically, for each catalogue segment, we store the maximum z-coordinate of all the segments above it in the catalogue. Since the catalogues are sorted by the above relationship, this is easily done using a prefix sum operation. There are at most $O(n \log n)$ catalogue segments in all, and any single catalogue contains at most n segments and a prefix sum of n items can be computed in $O(\log n)$ time using $O(\frac{n}{\log n})$ processors. Therefore, The tree can be augmented with this information using only an additional $O(\log n)$ time, yielding an $\tilde{O}(\log n)$ time algorithm for the 3-D maxima problem.

5.4.3 The 2-D Two Set Dominance Problem

The 2-D two set dominance counting problem takes as input two sets of points, U and V, of size n. The problem consists of counting, for each point in V, the number of points in U which dominate it. The 2-D two set dominance counting problem can be solved in $\tilde{O}(\log n)$ time using $O(n)$ processors using the following algorithm:

Algorithm 2-Set-Dominance-Counting:

1. Make segments from the points in U and build a nested plane sweep tree on these segments.
2. Search the tree for each point in V, but when searching a catalogue, sum the number of items in the tree which fall above the point.

6 Conclusions

As demonstrated by the application in Section ??, the segment tree and its parallel counterparts the plane sweep tree and nested plane sweep tree are useful data structures for solving a wide variety of problems in computational geometry. Open problems in this area could include implementations of segment tree variants for other parallel computing models. In fact, in [?] a parallel segment tree scheme is presented for the coarse grained multicomputer model.

References

[ACG$^+$85] A. Aggarwall, B. Chazelle, L. Guibas, C. O'Dunlaing, C. Yap, "Parallel Computational Geometry," in *Proc. of the 25th Annual IEEE Symposium on Foundations of Computer Science*, pp. 468–477, 1985.

[AtGo86] M. Attalah, M. Goodrich, "Efficient Plane Sweeping in Parallel," in *Proc. of the 2nd ACM Symposium on Computational Geometry*, pp. 216–255, 1986.

[DCR96] F. Dehne, A. Chan, A. Rau-Chaplin, "Coarse Grained Parallel Next Element Search," 1996.

[Kirk79] D. G. Kirkpatrick, "Optimal Search in Planar Subdivisions," in *SIAM J. Computing*, 12, pp. 18–27, 1979.

[ReSe92] J. Reif, S. Sen, "Optimal Randomized Parallel Algorithms for Computational Geometry," in *Algorithmica*, 7, pp. 91–117, 1992.