Parallel Randomized Techniques for Some Fundamental Geometric Problems: A Survey

Suneeta Ramaswami

Department of Computer Science
Rutgers University
Camden, NJ 08102
`rsuneeta@crab.rutgers.edu`

1 Introduction

Recent years have seen significant advances in parallel algorithm design for problems in computational geometry. Some of the earliest work in this area was done by Chow [7] and Aggarwal *et. al.* [1]. In these papers, the authors gave parallel algorithms for various fundamental problems such as two-dimensional convex hulls, planar-point location, trapezoidal decomposition, Voronoi diagram of points, triangulation *etc.*, which are known to have sequential run-times of $O(n \log n)$. Most of their algorithms, though in NC, were not optimal in $P.T$ bounds and a number of them have since been improved. Atallah, Cole and Goodrich [4] demonstrated optimal deterministic algorithms ($O(n)$ processors and $O(\log n)$ run-time) for many of these problems by applying the versatile technique of cascading divide-and-conquer and building on data structures developed in [1]. Cole and Goodrich give further applications of this technique in [9]. See [5] for a comprehensive survey on deterministic parallel algorithms for computational geometry. Reif and Sen [21] also obtained optimal randomized parallel algorithms for a number of these problems; these algorithms use n processors and run in $O(\log n)$ time with high probability. See [18] for a survey on the use of randomization in parallel algorithms.

The important problems of constructing *convex hulls of points in three dimensions* and *Voronoi diagrams of points in two dimensions* , however, eluded optimal parallel solutions for a long time. Both these problems have sequential run-times of $O(n \log n)$. Aggarwal *et. al.* [1] gave $O(\log^2 n)$ and $O(\log^3 n)$ time algorithms (using $O(n)$ processors) for the Voronoi diagram and convex hull problems, respectively, and the technique of cascaded-merging could not be extended to these problems to improve their run-times [9]. Subsequently, Goodrich [10] has given an algorithm for 3-d convex hulls that does optimal *work*, but has $O(\log^2 n)$ run-time, and Amato and Preparata [3] have described an algorithm that runs in $O(\log n)$ time but uses $n^{1+\epsilon}$ processors.

Randomization, however, proves to be very useful in obtaining optimal run-time as well as optimal $P.T$ bounds. (Sorting can be reduced to these problems, and hence the best possible run-time will be $O(\log n)$ on EREW and CREW PRAMs.) Note that the lower bound of $\Omega(n \log n)$ for these problems also applies to randomized algorithms. In [20], Reif and Sen gave an optimal parallel randomized algorithm on the CREW PRAM for the construction of the convex hull of points in three dimensions. Since the problem of finding the Voronoi diagram of points in two dimensions can be reduced to the three-dimensional convex hull problem, they also obtained an optimal parallel method for the former. Their algorithm runs in $O(\log n)$ time using $O(n)$ processors, with high probability, and there are no known deterministic algorithms that match these bounds. We would like to point out that Levcopoulos, Katajainen and Lingas [13] gave an optimal expected time parallel algorithm for the Voronoi diagram of a randomly chosen set of points. However, the randomized algorithms surveyed in this paper make no assumption about the distribution of the input set.

Similarly, an optimal parallel solution to construct the *Voronoi diagram of line segments in the plane* also poses difficulties. Furthermore, the randomized technique presented by Reif and Sen [20] cannot be extended in a straightforward way to this problem. By designing a new randomized sampling technique to overcome some of the obstacles presented by the method in [20], Rajasekaran and Ramaswami [17] give an optimal $O(\log n)$-time randomized parallel algorithm for this problem using $O(n)$ processors. This algorithm is optimal in $P.T$ bounds and an $O(\log n)$ factor improvement in run-time over the previously best-known deterministic algorithm by Goodrich, Ó'Dúnlaing and Yap [11], providing further evidence of the usefulness of randomization in obtaining efficient algorithms. As in the previous case, there are no known deterministic algorithms that match these bounds.

Parallel randomized techniques have also led to efficient algorithms for *higher-dimensional convex hulls*, as shown by Amato, Goodrich and Ramos [2]. In particular, they demonstrate $O(\log n)$-time randomized parallel algorithms that perform optimal $O(n \log n + n^{\lfloor d/2 \rfloor})$ work in order to compute the intersection of a set of n halfplanes in d dimensions (the dual of the problem of computing the convex hull of n points in d dimensions). Ramos [19] has also shown that randomized methods can be used to give near-optimal parallel algorithms for some one-dimensional lower envelope problems. The remainder of this paper provides a brief survey of the randomized techniques used to design efficient parallel algorithms for several fundamental geometric problems.

2 The Use of Randomization in Computational Geometry

The technique of randomization has been used to design sequential as well as parallel algorithms for a wide range of problems. In particular, efficient randomized algorithms have been developed for a number of computational geometry problems. Recent work by Clarkson and Shor [8], Mulmuley [16], and Haussler and Welzl [12] has shown that random sampling can be used to obtain better upper bounds for various geometric problems such as higher-dimensional convex hulls, halfspace range reporting, segment intersections, linear programming, *etc.*

Clarkson and Shor [8] used random sampling techniques to obtain tight bounds on the *expected* use of resources by algorithms for various geometric problems. The main idea behind their general technique is to use random sampling to divide the problem into smaller ones. The manner in which the random sample is used to divide the original input into subproblems depends on the particular geometric problem under consideration. They showed that for a variety of such problems, given a randomly chosen subset R of the input set S, the *expected* size of each subproblem is $O(|S|/|R|)$ and the *expected* total size is $O(|S|)$. A sample that satisfies these conditions is said to be *good*, and *bad* otherwise. They showed that any randomly chosen sample is good with constant probability, and hence bad with constant probability as well. This allows us to obtain bounds on the *expected* use of resources, but does not give high probability results (i.e. bounds that hold with probability $\geq (1 - 1/n^\alpha)$, where n is the input size, and $\alpha > 0$). As pointed out by Reif and Sen [20], this fact proves to be an impediment in the parallel environment due to the following reason: Parallel algorithms for such problems are, typically, recursive. For sequential algorithms, since the expectation of the sum is the sum of expectations, it is enough to bound the expected run-time of each step. For recursive parallel algorithms, the run-time at any stage of the recursion will be the *maximum* of the run-times of the subproblems spawned at that stage. There is no way of determining the maximum of expected run-times without using higher moments. Moreover, even if we can bound the expected run-time at the lowest level of the recursion, this bound turns out to be too weak to bound the total run-time of the algorithm.

2.1 Randomized Techniques for Parallel Algorithm Design

Polling In [20], Reif and Sen give a novel technique to tackle the problem of finding good samples efficiently in a *parallel* recursive setting. A parallel recursive algorithm can be thought of as a *process tree*, where a node corresponds to a procedure at a particular stage of the recursion, and the children of that node correspond to the subproblems created at that stage. The basic idea of the technique given in [20] is to find at every level of the process tree, a good sample of size $O(n^\epsilon)$ with high probability (where n can be thought of as the size of either the original input or the input to a subproblem). By doing this, they can show that the run-time of the processes at level i of the tree is $O(\log n/2^i)$ with high probability and hence the run-time of the entire algorithm is $O(\log n)$ with high probability.

By choosing a number of random samples (say $g(n)$ of them; typically $g(n) = O(\log n)$), we are guaranteed that one of them will be good with very high likelihood. The procedure to determine if a sample is good or not will have to be repeated for each of the $g(n)$ samples. However, we would have to ensure that this would not cause the processor bound of $O(n)$ to be exceeded and this is done by *polling* i.e. using only a fraction of the input ($1/g(n)$, typically) to determine the “goodness” of a sample. The idea is that the assessment of a sample (good or bad) made by this smaller set is a very good reflection of the assessment that would be made by the entire set. Thus Reif and Sen give a method to find a good sample efficiently at every level of the process tree, and this idea is useful for converting expected value results into high probability results.

Two Stages of Sampling It is important to consider the following side-effect that occurs in such recursive algorithms: When a random sample is used to divide the original problem into smaller ones, the total size of the subproblems can be bounded to only within a constant multiple of n. In a recursive algorithm, this results in an increase in the total problem size as the number of recursive levels increases. For a sample size of $O(n^\epsilon)$, the depth of the process tree for a parallel randomized algorithm would be $O(\log\log n)$, and even this could result in a polylogarithmic factor increase in the total problem size.

In [8], Clarkson and Shor get around this problem by using only a constant number of levels of recursion in their algorithm. They are able to do this by combining the divide-and-conquer technique with incremental techniques (which are inherently sequential; see [8] for further details). Reif and Sen's [20] strategy to handle this problem is to eliminate redundancy at every level of the process tree. In other words, since it is known that the *final* output size is $O(n)$, it is possible to eliminate those input elements from a subproblem which do not contribute to the final output. By doing this, they bound the total problem size at *every* level of the process tree to be within $c'.n$ for some constant c'. This step is non-trivial and, in general, avoiding a growth in problem size in this manner can be quite complicated. Moreover, the strategy used to eliminate redundancy seems to be very problem-specific.

Rajasekaran and Ramaswami [17] describe a method to use *random sampling at two stages* of the algorithm, which helps to overcome the problem of increase in total input size as the algorithm proceeds down the process tree. Their approach gets rid of the need to eliminate redundancy at each level of the process tree. In other words, it is not necessary to devise a method to control total problem size at every level of the process tree. By choosing much larger sized samples (of size $O(n/\log^q n)$ for an appropriate q) in the first stage of their algorithm, the polylog factor increase in processor bound still maintains the *total* processor bound as $O(n)$ at this stage. If this larger sized sample is good, it will again divide the original input into smaller problems of roughly equal size and the total subproblem size will be $O(n)$. Since the sample size is larger, the subproblem size will be relatively small and can be solved by using non-optimal techniques. As before, to ensure high probability bounds, $O(\log n)$ such samples of larger size are chosen. Consequently, the two-staged sampling approach eliminates the problem posed by the polylog-factor increase in problem size. (The idea of two-staged sampling in a somewhat different form was independently discovered by Amato, Goodrich and Ramos [2], which they called *biased sampling.*)

Observe that the issue of bounding the total size of the subproblems does not come up in "one-dimensional" problems like sorting because each element of the input set can lie in exactly one subproblem. This is not the case for problems such as convex hull or Voronoi diagram construction.

2.2 Applications to Fundamental Geometric Problems

Convex Hull of Points in Three Dimensions The *convex hull* of a set of points is defined to be the smallest convex set that contains the points. The technique of *polling*, summarized above, is used to give an optimal parallel randomized algorithm on the CREW PRAM for constructing the convex hull of a set of n points in three dimensions [20]. More accurately, an algorithm for the dual problem of computing the intersection of n half-planes is given. The subproblems for the recursive calls are defined by as follows: Construct the convex hull of a random sample (of size n^ϵ, say) and consider a point p inside it. This convex hull is a polyhedron composed of faces. Each of these faces, along with the point p, forms a "wedge". Each such wedge will be intersected by a subset of planes from the input set: this set of planes defines the input to the subproblem defined by that wedge.

As described earlier, it is necessary to control the total size of all subproblems at each level of recursion. In [20], the authors achieve this by exploiting the geometric properties of the specific problem. They carry out an exhaustive case analysis in order to remove every half-plane that forms a redundant part of a subproblem input. In other words, they remove from each subproblem those half-planes that cannot possibly form part of the output. By doing this, they ensure that the total problem size at *each* level of recursion is at most $c'n$. Finally, by using a careful processor allocation strategy, they obtain an optimal algorithm for constructing the convex hull of a set of n points in three dimensions. Observe that this immediately gives an optimal algorithm for constructing the Voronoi diagram of a set of n points in the plane. This is due to the reduction from higher-dimensional convex hulls to Voronoi diagrams in one lower dimension; see [6] for details. Since the all-points nearest neighbor and Euclidean minimum spanning tree can be obtained from the Voronoi diagram of points, these problems can also be solved optimally in parallel.

Voronoi Diagrams *Voronoi diagrams* are elegant and versatile geometric structures with numerous applications. The Voronoi diagram or, more accurately, the *nearest point Voronoi diagram* of a set of objects S is defined as follows: The *Voronoi region* associated with an element from S is the set of all points in the plane that are closer to that element than to any other element in S. The nearest point Voronoi diagram is the union of all the Voronoi regions. Figure 1 shows the Voronoi diagram of a set of planar points. The boundary edges between Voronoi regions are called *Voronoi edges* and the vertices of the diagram are called *Voronoi vertices.* When S consists of a set of points, the Voronoi edges are all straight line segments, whereas if S consists of line segments, the Voronoi regions are bounded by parabolic arcs as well as straight line segments.

The technique of *two-staged sampling*, summarized in the previous section, is used to give an optimal parallel randomized algorithm on the CRCW PRAM for computing the Voronoi diagram of a set of line segments in the plane. Each Voronoi edge of the random sample defines a subproblem for the recursive steps. By developing efficient search strategies to determine subproblems, and to merge the recursively solved subproblems, an optimal parallel randomized algorithm for the Voronoi diagram of line segments in the plane is obtained. Note that in order to maintain a processor bound of $O(n)$, the larger sample sizes used in the first stage of the algorithm necessitate fast methods to determine subproblems

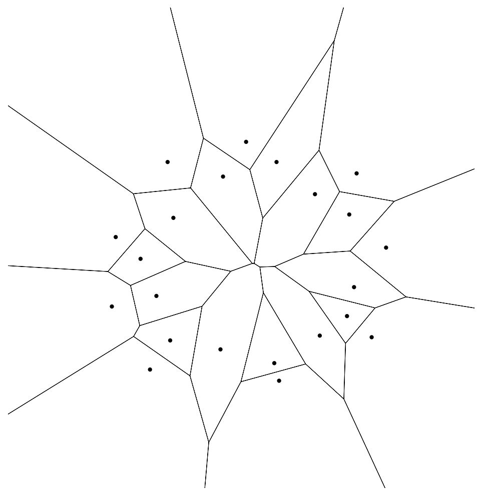

Fig. 1. The nearest point Voronoi diagram of a set of points.

(a less critical issue when the sample sizes are of size $O(n^{\epsilon})$). The two-staged sampling approach appears to be general enough to apply to other problems as well. For instance, Reif and Sen's algorithm [20] for three-dimensional convex hull can be simplified considerably by applying the idea of two-staged sampling. This method applies to the Voronoi diagram of points in the plane as well, thus giving an alternative optimal randomized parallel algorithm for this problem.

Higher-Dimensional Convex Hulls The higher-dimensional convex hull problem refers to the problem of computing the convex hull of a set of n points in an arbitrary d-dimensional space. Amato, Goodrich and Ramos [2] give $O(\log n)$ time randomized parallel algorithms on the EREW PRAM, using optimal $O(n \log n + n^{\lfloor d/2 \rfloor})$ work with high probability, for the dual problem of constructing the intersection of n halfspaces in d-dimensional space. Their algorithms are also based on parallel divide-and-conquer techniques, where the d-dimensional space is divided into cells and the halfspaces that intersect the cells define the subproblems. They use a technique similar to the two-staged sampling, which they call *biased sampling*, to avoid the total problem size increase at each level of the recursion. By combining this with other sophisticated geometric techniques (in particular, by using a parallel analogue [10] of Matoušek's shallow-cutting lemma [14, 15]), they obtain the stated result. In that paper, they also provide for the problem of half-space intersection in three dimensions a new "pruning" technique to eliminate redundancy at each level of recursion, which removes half-spaces that cannot contribute to the final output. This method is quite different from the one given by Reif and Sen [20].

The selection of the above results is meant to provide a flavor of parallel randomized techniques for some fundamental geometric problems. The list is certainly not claimed to be exhaustive, and several pertinent results have not been discussed since they lie outside the scope of this brief survey. It is hoped, however, that the selected results demonstrate the effectiveness of randomization in parallel algorithm design for problems in computational geometry.

References

1. A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. K. Yap. Parallel Computational Geometry. *Algorithmica*, 3:293–327, 1988.
2. N. Amato, M. Goodrich, and E. Ramos. Parallel Algorithms for Higher-Dimensional Convex Hulls. In *Proc. of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 683–694, October 1994.
3. N. M. Amato and F. P. Preparata. An NC^1 Parallel 3D Convex Hull Algorithm. In *Proc. 9th ACM Symp. on Computational Geometry*, 1993.
4. M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading Divide-and-Conquer: A Technique for Designing Parallel Algorithms. *SIAM J. Comput.*, 18(3):499–532, June 1989.
5. M. J. Atallah and M. T. Goodrich. Deterministic parallel computational geometry. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 497–536. Morgan Kaufmann Publishers Inc., 1993.
6. K. Q. Brown. *Geometric transforms for fast geometric algorithms.* PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1980.
7. A. Chow. *Parallel Algorithms for Geometric Problems.* PhD thesis, University of Illinois at Urbana-Champaign, 1980.
8. K. L. Clarkson and P. W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.
9. R. Cole and M. T. Goodrich. Optimal Parallel Algorithms for Polygon and Point-set Problems. *Algorithmica*, 7:3–23, 1992.
10. M. T. Goodrich. Geometric Partitioning Made Easier, Even in Parallel. In *Proc. 9th ACM Symp. on Computational Geometry*, 1993.
11. M. T. Goodrich, C. Ó'Dúnlaing, and C. K. Yap. Constructing the Voronoi Diagram of a Set of Line Segments in Parallel. In *Lecture Notes in Computer Science: 382, Algorithms and Data Structures, WADS*, pages 12–23. Springer-Verlag, 1989.
12. D. Haussler and E. Welzl. ϵ-nets and Simplex Range Queries. *Discrete and Computational Geometry*, 2:127–152, 1987.
13. C. Levcopoulos, J. Katajainen, and A. Lingas. An Optimal Expected-time Parallel Algorithm for Voronoi Diagrams. In *Proc. of the First Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 190–198. Springer-Verlag, 1988.
14. J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.
15. J. Matoušek. Reporting points in halfspaces. *Comput. Geom Theory Appl.*, 2(3):169–186, 1992.
16. K. Mulmuley. A Fast Planar Partition Algorithm. In *Proc. 20th IEEE Symp. on the Foundations of Computer Science*, pages 580–589, 1988.
17. S. Rajasekaran and S. Ramaswami. Optimal parallel randomized algorithms for the voronoi diagram of line segments in the plane and related problems. In *Proc. of the 10th Annual ACM Symp. on Computational Geometry*, pages 57–66, Stony Brook, New York, June 1994. Full paper submitted to *Algorithmica*.

18. S. Rajasekaran and S. Sen. Random sampling techniques and parallel algorithm design. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 411–451. Morgan Kaufmann Publishers Inc., 1993.
19. E. Ramos. Construction of 1-d lower envelopes and applications. In *Proc. of the 13th Annual ACM Symp. on Computational Geometry*, pages 57–66, Nice, France, June 1997.
20. J. H. Reif and S. Sen. Optimal Parallel Randomized Algorithms for Three Dimensional Convex Hulls and Related Problems. *SIAM J. Comput.*, 21(3):466–485, 1992.
21. J. H. Reif and S. Sen. Optimal Randomized Parallel Algorithms for Computational Geometry. *Algorithmica*, 7:91–117, 1992.