Proc. 1988 International Conference on Parallel Processing, vol III, Penn. State U., pp. 104-107

CONSTANT-TIME GEOMETRY ON PRAMS

Preliminary Version

Quentin F. Stout
Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122 USA

Abstract

Given n points chosen uniformly and independently from the unit square, it is shown that a parallel random access machine (PRAM) with n processors can solve several geometric problems in constant expected time, achieving linear speedup. The PRAM is assumed to be synchronous, with concurrent read and "collision detecting" write, where if two or more processors write to the same memory location simultaneously then the memory value becomes "collision". Problems solvable in constant expected time include determining for each point whether it is an extreme point of the convex hull, determining for each point if it is dominated by any other points, determining for each dominated point a maximal point that dominates it, finding the closest pair of points, and finding the furthest pair of points. These results extend to points chosen uniformly from the unit cube in d-dimensional space, and to many nonuniform distributions.

1. Introduction

It has well-known that synchronous concurrent read, concurrent write parallel random access machines (CRCW PRAMs) are strictly more powerful than most other parallel computers. For example, an n processor CRCW PRAM can determine the minimum of n numbers in $\Theta(\log \log n)$ time [Val], while it is easy to show that on a PRAM with either an exclusive read (ER) or an exclusive write (EW), or on a distributed memory machine, at least $\Omega(\log n)$ time is needed. However, Valiant's results do not really need the full power of concurrent writes, and we show that a weaker property, here called *detecting write (DW)*, can solve many problems equally rapidly. DW is intermediate between CW and EW, in that if two or more processors write to the same memory location at the same time, then the value becomes "collision", no matter what values were being written. Valiant's approach can be utilized on a CRDW PRAM and still finish in only $\Theta(\log \log n)$ time.

It is also well-known that some geometric problems involving data known to be chosen from a uniform distribution can be solved faster, in the expected case, then the same problems for arbitrary data. For example, n points chosen uniformly and independently from the real interval [0,1] can be serially sorted in $\Theta(n)$ expected time, as opposed to the $\Omega(n \log n)$ expected time for comparison-based sorts. Given n points chosen uniformly and independently from the unit square in 2-space, the convex hull and nearest neighbor of each can be determined by a serial computer in $\Theta(n)$ expected time [BeSh, BWY], as opposed to the $\Omega(n \log n)$ time needed for arbitrary planar data sets [Yao].

This paper shows that by combining the synchronous CRDW PRAM model with use of randomization in the algorithms and data sets generated randomly via uniform distributions, several geometric problems can be solved in constant expected time. Note that the best n processor CRCW PRAM algorithm known for sorting n points chosen uniformly from [0,1] takes $\Theta(\log n)$ time, i.e., it is not known how to sort such data sets any faster than arbitrary data sets. Since determining the extreme points of the convex hull of a set of points in the plane requires as much time as sorting [Yao], this would seem to imply that it takes $\Omega(\log n)$ time for a CRCW PRAM to determine which points are extreme points, even when the data is generated from a uniform distribution. However, the proof that determining extreme points is as hard as sorting holds only for a worst-case analysis, and we will show that the extreme points can be determined in constant expected time in a CRDW PRAM with a linear number of processors. To the best of our knowledge, these are the first constant expected-time algorithms for these problems on any parallel machine with only a linear number of processors.

Because of length limitations, results will be given with only sketches of their proof. The algorithms involve doing preliminary work which, with high probability, reduces the data set to a small number of points remaining to be considered. For these points, the processor to point ratio is very high and significantly different techniques can be utilized. However, the remaining points must be first moved together into a small array so that all the processors can locate them and help in the processing. Usually the points would be packed into the initial positions of the array, but this would take more than constant time. Therefore the array is made larger than the expected number of points remaining, and the points are mapped to random locations. The array must be large enough so that with high probability no two points are mapped to the same location, but it must also be small enough so that the processor to array size ratio remains sufficiently high.

The algorithms also have the property that if a step is reached where something undesired happens, then they resort to a standard worst-case polylogarithmic time CREW PRAM algorithm. Since this happens with very small probability, the expected time remains $\Theta(1)$.

Throughout, no attempt has been made to optimize constants.

2. Results

The phrase *randomly chosen points* will mean points chosen independently and uniformly on the unit square $[0,1] \times [0,1]$ in Euclidean 2-space. Algorithms will also require that processors *(PEs)* generate pseudo-random integers in given intervals. It is assumed that these can be computed in constant time, and that they are uniformly and independently distributed on the interval. *Distance* will be measured with the Euclidean metric, though any other L_p-metric could be used. A point in a finite set S is an *extreme point of* S if it is one of the corners of the smallest convex polygon containing S. The point (x_1,y_1) *dominates* the point (x_2,y_2) if $x_1 \geq x_2$ and $y_1 \geq y_2$. A point is *maximal* in a set if it is not dominated by any other point in the set.

The following lemma is used whenever many points have been eliminated from further consideration, and those remaining must be compressed into a small array so that processors can find them. If the expected number of points remaining is k, then the array will be at least of size k^2. Each processor holding such a point must find a place in the array to put the point.

2.0.1 Lemma On a CRDW PRAM of k processors, in $\Theta(1)$ time each can probably be allocated a unique position in an initialized integer array of k^2 positions, with probability of failure $o(k^{-0.5})$.

Sketch: Suppose each position of the integer array is initialized to -1. Each PE writes its ID (a unique positive integer) to a random array position, and then reads that position. If it reads its ID then that is its allocated position, while otherwise a conflict occured. To determine if any processors experienced conflicts, PE 0 writes *"false"* to a boolean variable `problems`. Next, if any PE experienced a conflict it writes "true" to `problems`, while otherwise it pauses. Now all PEs read `problems`, and are finished if and only if it is false. Otherwise (i.e., if it is true or conflict), another round is repeated by those PEs without allocated positions, where now each such PE first reads the location it picked and does not write to it if it is already allocated. The process is repeated 3 times. One can show that the probability that some PE still has not been allocated a position is $o(k^{-0.5})$.

The proof of the lemma can be extended to show that for any $a > 1$ and $b > 0$, there is a constant $C(a,b)$ such that k processors can be allocated a position in an array of k^a positions in $C(a,b)$ iterations, with probability of failure $o(k^{-b})$.

2.1 Maximal Points and Extreme Points

The following lemma is based on a modification of Valiant's observation that a synchronous CRDW PRAM of k^2 processors can determine the maximum of k values in constant (worst-case) time. The reason for including "Not a Point" as a value is because later algorithms will place a few points into a large array, and hence many entries will not correspond to points.

2.1.1 Lemma In a CRDW PRAM of n processors, suppose each entry of a global array $p[0..n^{1/3}-1]$ contains a point or the value NAP (Not A Point). Then in constant worst-case time the maximal points can be determined, and for each nonmaximal point a maximal dominating point can be determined.

Proof: Let $s=n^{1/3}$, and assume that arrays maximal:[0..s-1] of boolean and dominator:[0..s-1] of point are initialized to true and NAP, respectively. When finished, maximal[i] is true if and only if p[i] is a maximal point, and if p[i] is a nonmaximal point then dominator[i] is one of its maximal dominators. Each PE executes the following algorithm, where i represents the index of the PE ($0 \leq i \leq n-1$), a local variable which is already initialized. Other local variables in each PE are i1, i2, and i3. A temporary global boolean array T:[0..s-1, 0..s-1] is also used. Throughout, whenever conditional instructions occur where some PEs may take one branch and others take the other branch, it is implied that pauses are inserted so that all PEs complete each branch in the same time.

1. Read p[i], and if it is NAP then write false to maximal[i].
2. *Let* $i1 = i \text{ div } s^2$, $i2 = (i \text{ div } s) \bmod s$, *and* $i3 = i \bmod s$.
   {Notice that for each i1,i2,i3 triple with $0 \leq i1,i2,i3 \leq s-1$, there is exactly one PE with that triple}.
3. If i3=0 then write true to T[i1,i2]. {At the end of step 5, T[i1,i2] will still be true only if p[i2] dominates p[i1].}
4. Read p[i1], p[i2], and p[i3]. If p[i1] or p[i2] are not points, or if p[i2] does not dominate p[i1], then write false to T[i1,i2] and go to 6.
5. Otherwise, if p[i3] is a point and p[i3] dominates p[i1] and i3<i2 then write false to T[i1,i2]. {This signals that, even though p[i2] could be used to show that p[i1] is not maximal, there is a dominating point of smaller index and p[i2] should not be used. It doesn't matter whether T[i1,i2] ends up with the value false or "collision"}.
6. If i3=0 then read T[i1,i2], and if it is true then write false to maximal[i1].
7. {At this point, maximal is correctly determined for all positions. Now for each nonmaximal point we locate the maximal dominator of minimal index. These steps are similar to steps 3-6}
   If i3=0 then write true to T[i1,i2].
8. Read maximal[i2] and maximal[i3]. If p[i1] or p[i2] is not a point, or if p[i2] does not dominate p[i1], or if p[i2] is not maximal, then write false to T[i1,i2] and go to 10.
9. Otherwise, if p[i3] dominates p[i1], p[i3] is maximal, and i3<i2 then write false to T[i1,i2].
10. If i3=0 then read T[i1,i2], and if it is true then write p[i2] to dominator[i1].

Since each step takes constant time, the algorithm finishes in constant time.

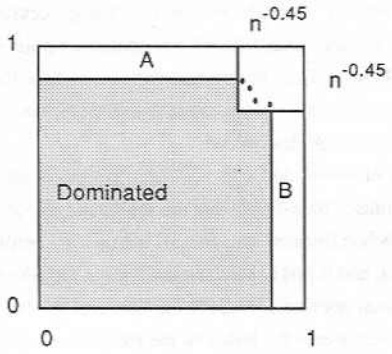

Figure 1

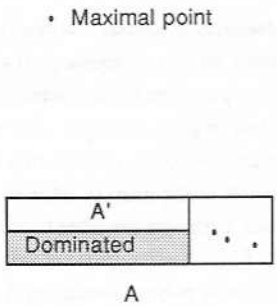

Figure 2

2.1.2 Theorem On a CRDW PRAM with n processors, given a set of n randomly chosen points, in constant expected time it can be determined which points are maximal. Further, in constant expected time, for each dominated point one of the maximal points dominating it can be determined.

Sketch: A sequence of steps is used to continually reduce the number of candidate extreme points for the next step. First it is determined if there are any points in the "corner" $[1\text{-}n^{-0.45},1] \times [1\text{-}n^{-0.45},1]$. With probability close to 1 there are some, but not more than $n^{0.11}$. If there are points in the corner, then they are moved to an array of size $n^{1/3}$, and the maximal points are determined. In this set, the points with greatest x-coordinate and greatest y-coordinate are put in prespecified locations. Every PE i reads them and determines if the i^{th} point is dominated by them. If so the point is marked as not maximal, and dominator is set.

For those remaining there are two groups: those in section A and those in section B of Figure 1. These are treated similarly, so only A will be discussed. Now it is determined if there are any points in A with x-coordinates in $[1\text{-}n^{-0.5},1\text{-}n^{-0.45}]$. With high probability there are some, but not more than $n^{0.11}$. These are also moved to an array of size $n^{1/3}$ and the maximal points determined. Then the point with the largest y-coordinate of this set is read by all PEs corresponding to points not yet dominated, and if they are dominated by this point they set maximal and dominator to appropriate values. This leaves a set A' as in Figure 2.

The process is repeated using regions with x-coordinates in the ranges $[1\text{-}n^{-0.55},1\text{-}n^{-0.5}]$, $[1\text{-}n^{-0.6},1\text{-}n^{-0.55}]$, ..., $[0,1\text{-}n^{-0.95}]$. With probability $o(n^{-0.01})$ each step is completed successfully. If any step does not complete successfully (i.e., either there are no points in the region, or else the PEs corresponding to the points cannot be allocated a position in the array of size $n^{1/3}$ in constant time) then all PEs revert to using a deterministic CREW PRAM algorithm taking $\Theta(\log n)$ worst-case time [AtGo]. The total expected time is $\Theta(1)$.

A similar approach can be used to determine extreme points of the convex hull of the points, starting at the corners and working inwards. In case of failure at some step, CREW PRAM algorithms which finish in polylogarithmic worst-case time [ACGOY, AtGo, MiSt] are used.

2.1.3 Theorem On a CRDW PRAM with n processors, given a set of n randomly chosen points, in constant expected time it can be determined which points are extreme points. Further, in constant expected time, for each point which is not extreme, three extreme points which contain the point in the triangle they form can be determined (or two can be determined, if the point is on the boundary of the convex hull).

Let E denote the number of extreme points. One can show that the expected value of E is $\Theta(\log n)$ [ReSu], which implies that, for any integer k, with probability close to 1 the ratio n/E is $\Omega(E^k)$. Using this, in constant expected time one can apply algorithms which examine all possible combinations of k of the extreme points. This easily yields the following.

2.1.4 Corollary On a CRDW PRAM with n PEs, given a set of n randomly chosen points, in constant expected time the maximal distance between any pair of points can be determined, and enclosing rectangles and circles of minimal area can be determined.

2.2 Closest Pair

A significantly different problem is to determine the closest pair of points. For this problem there is no immediate technique to eliminate points, since even if n-1 points are known it is not possible to determine the closest pair without knowing the last point, and the closest pair might consist of the last point and any of the previously examined points. However, one can reduce the expected number of pairs for which the distance must be determined, by partitioning the unit square into subsquares of edgelength L. If it is known that at least one square has two points in it, then the answer is known to be no more than $\sqrt{2}$ L. In this situation, if a point is in square S in Figure 3, then it may be part of the closest pair only if there are points in S or the 20 nearby squares. By choosing L to be $n^{-0.99}$, then with probability close to 1 there is a square with at least 2 points, no square has 4 or more points, and the number of squares with 2 or more points is $o(n^{0.1})$.

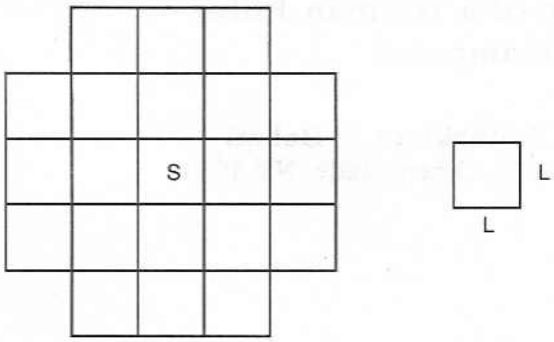

Figure 3. The 20 neighbors with points within $\sqrt{2}$ L of S.

Using this fact, first points are written to their squares. All points in squares with collisions, or which are in one of the 20 squares near a square with collisions, are written to a new array of size $n^{0.2}$. The closest pair within this new array is determined, and also the closest pair involving a square and one of its 20 nearby squares, where all 21 have at most one point in them. The closest pair for the original set is the closer of these two pairs. As before, if any steps cannot be completed properly, then the PEs resort to using a polylogarithmic CREW PRAM algorithm [AtGo].

2.2.1 Theorem On a CRDW PRAM with n processors, given a set of n randomly chosen points, in constant expected time a closest pair can be determined.

3. Final Remarks

This short preliminary version of the paper sketches some ways to accomplish constant time algorithms using random data on a synchronous CRDW PRAM. To the best of my knowledge, these are the first constant expected time algorithms for these problems on any parallel model using a linear number of processors. It seems that the CRDW PRAM is the "weakest" parallel computer which can solve these problems in constant time using only a linear number of processors, but it is not clear how to formalize this properly, let alone prove it.

Many existing algorithms for CRCW PRAMs can be modified to work in the same time on CRDW PRAMs. This occurs because the only concurrent writes they utilize have the property that whenever two PEs are writing to the same memory location at the same time, then they are writing the same value. Such algorithms can often be converted to a scheme as in Lemma 2.0.1, where "conflict" is as useful as the value that was being written. However, not all CRCW PRAM algorithms are of this form, and more work needs to be done to understand which problems can be solved faster on a CRCW PRAM than on a CRDW PRAM.

The algorithms given herein extend quite naturally higher dimensional data. For any fixed dimension d, given n random points chosen from the d-dimensional unit cube, an n processor CRDW PRAM can solve the d-dimensional domination, extreme points, furthest pair, smallest enclosing box, smallest enclosing sphere, and closest pair problems in constant expected time.

The algorithms can also be easily extended to many nonuniform distributions, such as the d-dimensional normal distribution. However, some distributions will cause difficulties because the number of maximal or extreme points may be too large to permit the use of techniques which assume a high processor/point ratio. For example, using the uniform distribution on the unit sphere will result in $\Theta(n^{1/2})$ extreme points, on average [Rayn], rendering the approach of Corollary 2.1.4 invalid.

Acknowledgements

This research was partially supported by Incentive for Excellence Awards from Digital Equipment Corporation, and by National Science Foundation grant DCR-85-07851.

References

[ACGOY] A. Aggarwal, B. Chazelle, L. Guibas, C. O'Dunlaing, and C. Yap, "Parallel computational geometry", *Proc. IEEE Symp. on Found. Comp. Sci.* 26 (1985), pp. 468-477.

[AtGo] M.J. Atallah and M.T. Goodrich, "Efficient parallel solutions to geometric problems", CSD-TR-504, Dept. of Comp. Sci., Purdue Univ., 1985.

[BeSh] J. Bentley and M.I. Shamos, "Divide and conquer for linear expected time", *Info. Proc. Let.* 7 (1978), pp. 87-91.

[BWY] J. Bentley, B.W. Weide, and A.C. Yao, "Optimal expected time algorithms for closest point problems", *ACM Trans. Math. Software* 6 (1980), pp. 563-580.

[MiSt] R. Miller and Q.F. Stout, "Parallel algorithms for convex hulls", *Proc. Comp. Vision and Pat. Recogn. 1988*, to appear.

[Rayn] H. Raynaud, "Sur l'enveloppe convexe des nuages de points aleatoires dans R^n. I", *J. Appl. Prob.* 7 (1970), pp. 35-48.

[ReSu] A. Renyi and R. Sulanke, "Uber die konvexe Hulle von n zufallig gewahlten Punkten, I", *Z. Wahrschein.* 2 (1963), pp. 75-84.

[Val] L. Valiant, "Parallelism in comparison problems", *SIAM J. Computing* 4 (1975), pp. 348-355.

[Yao] A.C. Yao, "A lower bound to finding convex hulls", *J. ACM* 28 (1981), pp. 780-789.