Computing the diameter of a point set: sequential and parallel algorithms

Michiel Smid*

November 4, 2003

1 Introduction

Let S be a set of n points in the plane. We want to compute the *diameter* $D(S)$ of S, which is defined as the largest distance between any two points of S. That is, if $d(p,q)$ denotes the Euclidean distance between the points p and q, then we want to compute

$$D(S) = \max\{d(p,q) : p,q \in S\},$$

together with two points a and b of S such that $d(a,b) = D(S)$. Such a pair is called a *diametral pair*.

A trivial way to solve this problem is to compute the distance between every pair of points and select the largest distance. Since one distance computation takes constant time, this gives an algorithm with running time $O(n^2)$.

Is there a faster algorithm? The answer is "yes": In these notes, we will give an $O(n \log n)$-time algorithm for computing the diameter of a set of n points in the plane. It can be shown that this is optimal in the algebraic computation tree model. In the second part of these notes, we will design an optimal *parallel* algorithm for the diameter problem.

*School of Computer Science, Carleton University, Ottawa, Ontario, Canada K1S 5B6. E-mail: michiel@scs.carleton.ca.

2 Some properties of the diameter

In order to avoid looking at each pair of points explicitly, we have to find some properties of diametral pairs.

Recall the definition of the convex hull of a point set S: It is the smallest convex set that contains all points of S. The convex hull consists of vertices and edges. An edge is bounded by two vertices, which are points of S. If S contains a point p that is in the interior of a convex hull edge, then, by definition, p is *not* a convex hull vertex.

Lemma 1 *Let S be a finite set of points in the plane and let S' be the set of points of S that are on its convex hull. The diameter of S is equal to the diameter of S'.*

Proof. Let p and q be two distinct points of S that are not both contained in S'. Assume without loss of generality that $q \notin S'$. We prove that there is a point r in S' such that $d(p,q) < d(p,r)$.

Assume without loss of generality that the line through p and q is horizontal and that p is to the left of q. Since q is not on the convex hull of S, there must be a convex hull vertex r that is on or to the right of the vertical line through q. It is easy to see that $d(p,q) < d(p,r)$.

Hence, we have shown the following: If p and q are two distinct points of S that are not both contained in S', then $d(p,q) < D(S)$. It follows that the diameter of S is determined by two points of S'. This proves that $D(S) = D(S')$. ∎

This lemma suggests the following approach for computing the diameter of S:

1. Compute the vertex set S' of the convex hull of S.

2. Compute the diameter of S'.

We know how the set S' can be computed in $O(n \log n)$ time. How do we compute the diameter of S'? As we will see, the fact that the points of S' are the vertices of a *convex polygon* allows us to compute the diameter of S' in $O(|S'|) = O(n)$ time.

3 Computing the diameter of a convex polygon

In order to simplify the notation, we will write S instead of S'. Hence, $S = \{p_1, p_2, \ldots, p_n\}$ is a set of n points in the plane that form the vertices of a convex polygon, given in, say, counterclockwise order. Our goal is to compute the diameter of S.

Exercise 1 Draw a convex polygon with n vertices and look at the sequence of distances

$$d(p_1, p_2), d(p_1, p_3), d(p_1, p_4), \ldots, d(p_1, p_n).$$

By looking at your polygon, it seems reasonable to conjecture that this sequence is *uni-modal*, i.e., it consists of one (possibly empty) increasing part followed by one (possibly empty) decreasing part. Prove that this is *not* true in general.

(If this conjecture would have been true, then the following algorithm *would have* computed the diameter of S: Using binary search, compute a point p_i that is furthest away from p_1. In a similar way, for each k with $1 \leq k \leq n$, use binary search to compute a point of S that is furthest away from the point p_k. This gives n pairs of points and the pair having the largest distance determines the diameter of S.)

This exercise sounds like bad news. It turns out, however, that a variation of it is true:

Lemma 2 *Let ℓ be the line through p_1 and p_2. The sequence*

$$d(p_3, \ell), d(p_4, \ell), \ldots, d(p_n, \ell)$$

is uni-modal.

Proof. This follows immediately from the fact that the points of S form the vertices of a convex polygon. ∎

If e is an edge of our convex polygon, then we denote by ℓ_e the line through e. Let L be the set of all pairs (p, q) for which there is an edge e on our convex polygon having p as one of its endpoints, such that

$$d(q, \ell_e) = \max\{d(r, \ell_e) : r \in S\}.$$

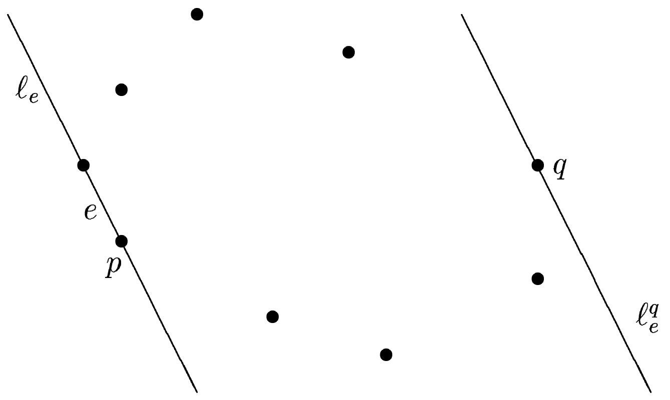

Figure 1: *Illustrating the definition of the set L. Edge e has p as one of its endpoints and q is furthest away from ℓ_e. Hence, L contains the pair (p, q).*

See Figure 1 for an illustration. In this figure, point q is at maximum distance from the line ℓ_e. In other words, if ℓ_e^q is the line through q that is parallel to ℓ_e, then all points of S are on or between ℓ_e and ℓ_e^q. (Because of convexity, there can be only one other point on ℓ_e^q.) Hence, the pair (p, q) is contained in the set L.

Lemma 3 *The diameter of S is equal to the largest distance determined by any element of L.*

Proof. Let a and b be two points of S such that $d(a, b) = D(S)$. We will show that the pair (a, b) or (b, a) is contained in L. This will prove the lemma.

Assume without loss of generality that the line through a and b is horizontal. All points of $S \setminus \{a, b\}$ are strictly between the two vertical lines through a and b. (Otherwise, the diameter of S would be larger than $d(a, b)$.) Rotate these two lines *simultaneously* in clockwise order around a and b, and stop as soon as one of them hits an edge of the convex hull of S. (Hence, while rotating, the two lines remain parallel.) See Figure 2.

Let e be the edge that is hit first. If a is an endpoint of this edge, then it is not hard to see that b has maximum distance to the line through e. Therefore, the pair (a, b) is contained in L. By a symmetric argument, if b is an endpoint of e, then the pair (b, a) is contained in L. ∎

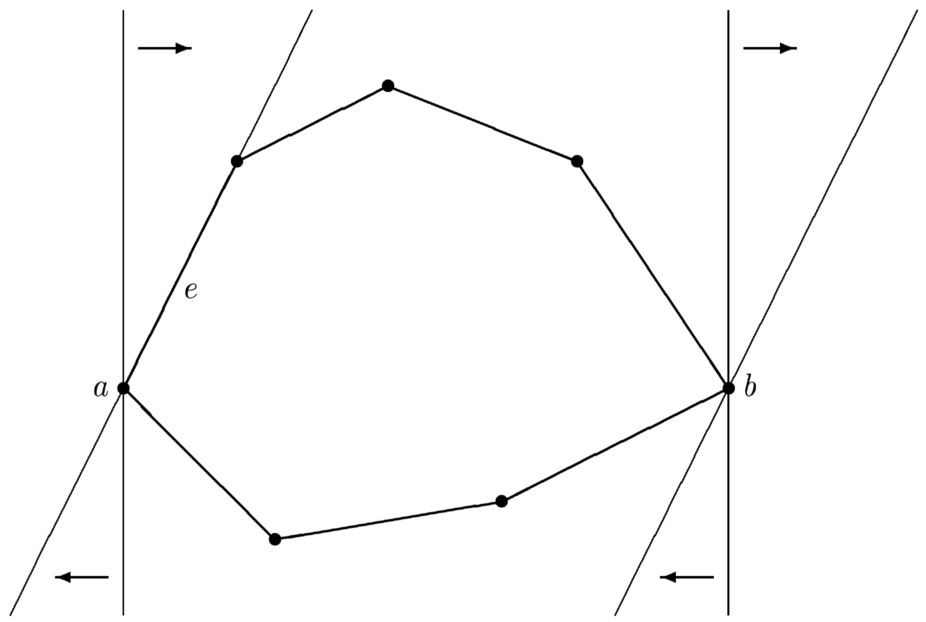

Figure 2: *Illustrating the proof of Lemma 3. The line through a is the first one that hits an edge of the convex hull.*

3.1 The algorithm

Lemma 3 tells us that we can compute the diameter of S in the following way:

Step 1: Compute the set L.

Step 2: Compute all distances determined by the elements of L, and select the largest one.

Step 2 is easy. First observe that L contains at most $4n$ elements: The convex polygon S contains n edges. Consider one such edge e. It is bounded by two points. There are at most two points of S at maximum distance from the line ℓ_e through e. Hence, this edge "delivers" at most four pairs to L.

As a result, Step 2 takes only $O(n)$ time. Of course, the problem is how to compute the set L. We will show that this can also be done in $O(n)$ time.

Recall that we numbered the points of S as $p_1, p_2, \ldots, p_n$, in counterclockwise order. Let e be the edge bounded by p_1 and p_2, and let ℓ_e be the line that contains e. We want to find the (at most two) points of S that are at maximum distance from ℓ_e. Lemma 2 implies that there is an easy way to find these points:

```
    $j := 3$;
    **while** $d(p_{j+1}, \ell_e) > d(p_j, \ell_e)$
    **do** $j := j + 1$
    **endwhile**
```

In this procedure, indices are to be read modulo n, i.e., $p_{n+1} = p_1$. At the end of this while-loop, we have

$$d(p_3, \ell_e) < d(p_4, \ell_e) < \ldots < d(p_{j-1}, \ell_e) < d(p_j, \ell_e)$$

and

$$d(p_{j+1}, \ell_e) \leq d(p_j, \ell_e).$$

Then it follows from Lemma 2 that

$$d(p_n, \ell_e) < d(p_{n-1}, \ell_e) < \ldots < d(p_{j+2}, \ell_e) < d(p_{j+1}, \ell_e).$$

Hence, if $d(p_{j+1}, \ell_e) < d(p_j, \ell_e)$, then p_j is the only point having maximum distance to ℓ_e, and we insert the two pairs (p_1, p_j) and (p_2, p_j) into L. Otherwise, we have $d(p_{j+1}, \ell_e) = d(p_j, \ell_e)$, and we insert the four pairs (p_1, p_j), (p_1, p_{j+1}), (p_2, p_j) and (p_2, p_{j+1}) into L.

Hence, we now have "handled" the first edge e. The total time spent is $O(n)$, because the index j may be as large as n.

Let e' be the next convex hull edge, having endpoints p_2 and p_3. How do we find the (at most two) points of S that are at maximum distance from $\ell_{e'}$? Of course, we can use the same procedure as for edge e, starting with $j = 4$. There is, however, a better way:

Lemma 4 *Let j be an index such that p_j has maximum distance to line ℓ_e. Similarly, let i be an index such that the point p_i has maximum distance to $\ell_{e'}$. Then $j \leq i \leq n$ or $i = 1$.*

Proof. Assume without loss of generality that the line through p_1 and p_2 is horizontal. Observe that p_1 is to the left of p_2. Let ℓ_j (resp. ℓ'_j) be the line through p_j that is parallel to ℓ_e (resp. $\ell_{e'}$). See Figure 3.

Since p_j has maximum distance to ℓ_e, we know that $p_4, p_5, \ldots, p_{j-1}$ are all on or below ℓ_j. Then, convexity implies that these points must be to the

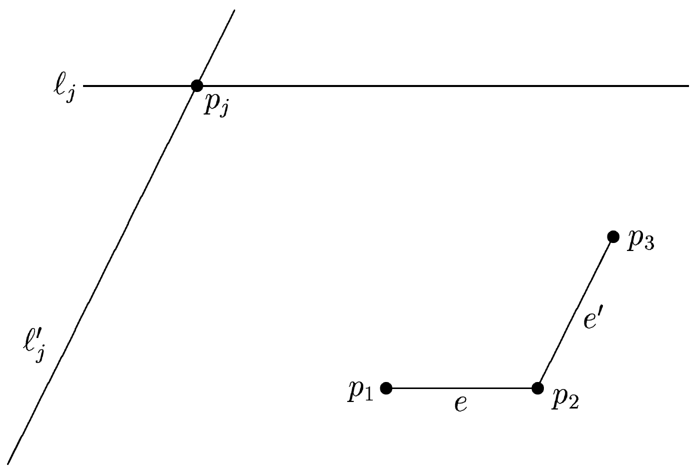

Figure 3: *Illustrating the proof of Lemma 4.*

right of line ℓ'_j. (Convince yourself that this is also true if the angle at p_2 is less than $\pi/2$.) As a result, $d(p_j, \ell_{e'}) > d(p_k, \ell_{e'})$, $k = 4, 5, \ldots, j-1$. This proves that $i \notin \{4, 5, \ldots, j-1\}$. It is clear that $i \neq 2$ and $i \neq 3$. ∎

This lemma has the following consequence. In order to find the index i, we do *not* have to start at p_4 while walking along the convex polygon; rather, we can start at p_j.

In general, let e_k be the convex hull edge with endpoints p_k and p_{k+1}, and let p_u be a point that is at maximum distance from the line ℓ_k through e_k. Then, in order to find the point that has maximum distance to the line ℓ_{k+1} that contains the next edge e_{k+1}, we scan the convex polygon, starting at p_u, and consider

$$p_u, p_{u+1}, \ldots, p_n, p_1, p_2, \ldots, p_k,$$

until we have found the maximal element of the uni-modal sequence

$$d(p_u, \ell_{k+1}), d(p_{u+1}, \ell_{k+1}), \ldots, d(p_n, \ell_{k+1}), d(p_1, \ell_{k+1}), \ldots, d(p_k, \ell_{k+1}).$$

In this way, we compute the entire set L, by walking at most twice around the convex polygon. This proves that this set can be computed in $O(n)$ time.

We summarize our result:

Theorem 1 *Let S be a set of n points in the plane. The diameter of S can be computed in $O(n \log n)$ time. If the points of S are given as the vertices* *of a convex polygon, sorted along its boundary, then the diameter of S can be computed in $O(n)$ time.*

Remark 1 There can be more than one pair of points that determines the diameter. All these pairs, however, must be contained in the set L. (This follows from the proof of Lemma 3.) As a result, there are at most $4n$ pairs of points that are at distance $D(S)$, and our algorithm finds them all.

In the analysis of our algorithm, we assumed that the following operation can be performed in constant time:

> Given a line ℓ through points a and b, that is directed from a to b, and given two points x and y that are both to the left of ℓ, decide whether $d(x,\ell) > d(y,\ell)$, $d(x,\ell) < d(y,\ell)$, or $d(x,\ell) = d(y,\ell)$.

How do we implement this operation? Here is the answer. Let ℓ' be the directed line through x that is parallel to ℓ and that has the same direction as ℓ. Then $d(x,\ell) > d(y,\ell)$ if and only if y is to the right of ℓ'. Observe that ℓ' contains the two points x and $x+b-a$ and that ℓ' is directed from x to $x+b-a$. It follows that $d(x,\ell) > d(y,\ell)$ if and only if the three points x, $x+b-a$, and y form a right-turn.

4 A parallel algorithm

In this section, we consider the problem of designing a parallel algorithm for computing the diameter of a planar point set. The model of computation we use is the CREW PRAM. We will show that in this model, the diameter can be computed in $O(\log n)$ time, using $O(n)$ processors. We will also show that this is optimal.

Let S be a set of n points in the plane. Our general approach is the same as before: First, we compute the convex hull of S. Then, we compute the diameter of the convex hull.

Let us look at the second step first. Hence, $S = \{p_1, p_2, \ldots, p_n\}$ is given as the set of vertices of a convex polygon, sorted in counterclockwise order along its boundary. The algorithm for computing the diameter of S consists of two steps.

Step 1: We use Lemma 2 to compute the set L of Lemma 3. This is done using n processors. For each i with $1 \leq i \leq n$, the i-th processor handles the i-th edge e_i of our convex polygon. To be more precise, if e_i has p_i and p_{i+1} as its endpoints[1], then the i-th processor makes a binary search in the uni-modal sequence

$$d(p_{i+2}, \ell), d(p_{i+3}, \ell), \ldots, d(p_n, \ell), d(p_1, \ell), d(p_2, \ell), \ldots, d(p_{i-1}, \ell),$$

where ℓ is the line that contains e_i. In this way, this processor finds the at most two maximal elements of this sequence, and these give rise to at most four pairs of points that have to be inserted into the set L. The processor writes these pairs in the locations $4i-3$, $4i-2$, $4i-1$ and $4i$ of an array $L[1 \ldots 4n]$. (Hence, each array entry contains at most one pair of points.)

This concludes Step 1. Convince yourself that this step can be implemented on a CREW PRAM. In particular, where do we need concurrent reads?

Step 2: We know that the diameter of S is equal to the largest distance determined by any element of L. This largest distance is computed using a CREW PRAM algorithm that simulates a knock-out tournament.

This concludes the description of the algorithm. It is easy to analyze the complexity. The number of processors used is bounded by $O(n)$. Consider Step 1. Each processor makes a binary search in a set of size $n-2$. Clearly, this takes $O(\log n)$ time. Regarding Step 2, the maximum of $4n$ numbers—the distances in L—can be computed in $O(\log n)$ time, even using $O(n/\log n)$ processors. Hence, we have proved the following result.

Theorem 2 *Let S be a set of n points in the plane. Assume that the points of S are given as the vertices of a convex polygon, sorted along its boundary. On a CREW PRAM, the diameter of S can be computed in $O(\log n)$ time, using $O(n)$ processors.*

Hence, in order to show how to compute the diameter of an arbitrary point set S, it remains to give a parallel algorithm that computes the convex hull of S.

4.1 A parallel convex hull algorithm

In this section, S is an arbitrary set of n points in the plane. We will give a parallel *multiway divide-and-conquer* algorithm that computes the convex

[1]observe that $p_{n+1} = p_1$

hull of S. This algorithm first sorts the points of S by their x-coordinates. Then it divides the sorted sequence into $\sqrt{n}$ subsequences, each of size $\sqrt{n}$, and recursively computes the convex hull of each subsequence. Finally, it computes the convex hull of the $\sqrt{n}$ subhulls.

We will not give any details about the first step. We just mention that on a CREW PRAM, the points of S can be sorted by their x-coordinates in $O(\log n)$ time, using $O(n)$ processors. Let $S = \{p_1, p_2, \dots, p_n\}$ be the sorted sequence. Assume that these points are stored in an array $S[1 \dots n]$, so that $S[i] = p_i$, $1 \leq i \leq n$.

The convex hull of S consists of an *upper hull* and a *lower hull.* We will show how the upper hull can be computed. The lower hull can be computed in a symmetric way.

We do the following in parallel: For each i with $1 \leq i \leq \sqrt{n}$, using $O(\sqrt{n}\,)$ processors, we recursively compute the upper hull of the points in the subarray $S[(i-1)\sqrt{n}+1 \dots i\sqrt{n}\,]$. (So, in total, $O(n)$ processors are working simultaneously. Also, in each subarray, the points are sorted already.)

Let $q_1^i, q_2^i, \dots, q_{n_i}^i$ be the vertices of the i-th upper hull, sorted from left to right. Then, $n_i \leq \sqrt{n}$. We assume that each such sequence is stored in an array $Q_i[1 \dots n_i]$.

It remains to compute the upper hull of the $\sqrt{n}$ upper hulls. For each i and j with $1 \leq i \leq \sqrt{n}$, $1 \leq j \leq \sqrt{n}$, $i \neq j$, one processor computes the *upper common tangent* of the i-th and j-th upper hulls. (Hence, $\sqrt{n}(\sqrt{n}-1) < n$ processors are working in parallel.) One tangent can be computed in $O(\log n)$ time by a binary search. (We do not give the details here. Draw some figures for yourself, and try to find out how the binary search works.)

Hence, at this moment, for each fixed value of i, we have $\sqrt{n}-1$ upper common tangents. Using $O(\sqrt{n}\,)$ processors, we find the upper tangent V_i that is tangent to the i-th upper hull and an upper hull to the left of it, and that has the largest slope, where slopes are measured by their angles (in $[-\pi/2, \pi/2]$) with the negative x-axis. Then, again using $O(\sqrt{n}\,)$ processors, we find the upper tangent W_i that is tangent to the i-th upper hull and an upper hull to the right of it, and that has largest slope, where now slopes are measured by their angles (in $[-\pi/2, \pi/2]$) with the positive x-axis. Let v_i (resp. w_i) be the point of the i-th upper hull that lies on V_i (resp. W_i). Let α_i be the angle between V_i and W_i. See Figure 4.

If $\alpha_i \leq \pi$, then none of the points of the i-th upper hull is on the upper hull of S. Otherwise, if $\alpha_i > \pi$, then all points on the i-th upper hull between v_i and w_i—both endpoints included—belong to the upper hull of S; moreover,

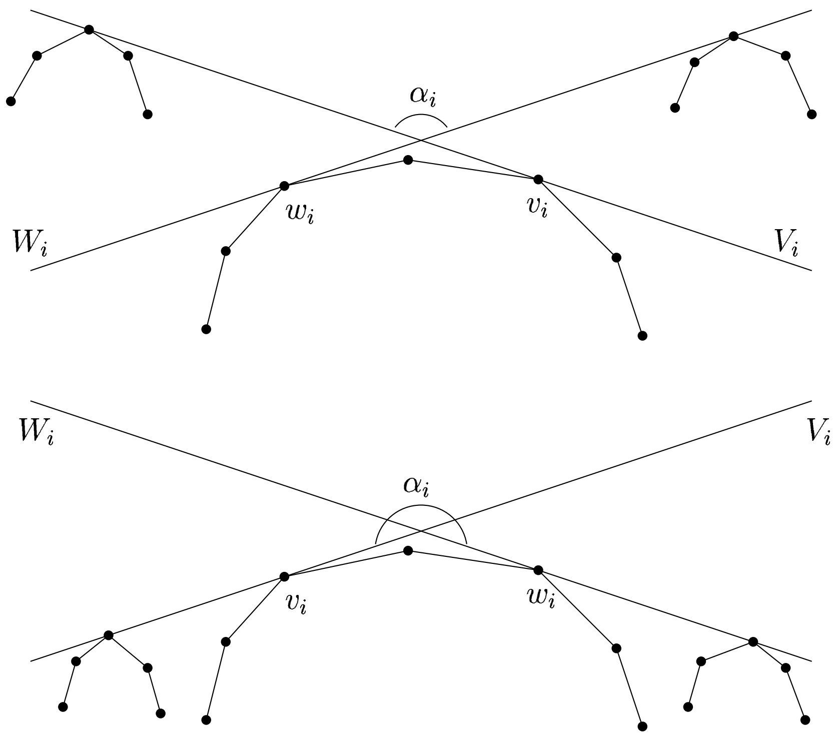

Figure 4: *Illustrating the algorithm that computes the upper hull of upper hulls.*

these are the only points on the upper hull of S that are on the i-th upper hull. (Again, we do not give a proof; by looking at Figure 4, convince yourself that this is true.)

What do we know at this moment? For each i with $1 \leq i \leq \sqrt{n}$, we either know that all points on the i-th upper hull do not belong to the upper hull of S, or we have a chain $v_i \dots w_i$ that contains exactly the points of the i-th upper hull that are on the upper hull of S. Hence, the upper hull of S can be obtained by concatenating all these chains. Here we have to be careful: How do we represent the upper hull of S? After the recursive step, we assumed that each of the $\sqrt{n}$ upper hulls was stored in an array. (We need an array in order to find upper common tangents using binary search.) Therefore, at the end of the current merge step—during which we compute the upper hull of S—we also have to take care that the points on the upper hull of S are stored in an array, sorted from left to right. This array is obtained as follows.

We will use an array $A[1 \dots n]$, and n processors. For each i and j with $1 \leq i \leq \sqrt{n}$ and $1 \leq j \leq \sqrt{n}$, the $(i\sqrt{n}+j)$-th processor checks whether the j-th point on the i-th upper hull, i.e., the point q^i_j, lies on the upper hull of S. (This is done by testing if $\alpha_i \leq \pi$, or if $\alpha_i > \pi$ and the x-coordinate of q^i_j lies in between the x-coordinates of v_i and w_i.) Then this processor writes a one (resp. zero) in $A[i\sqrt{n}+j]$ if q^i_j belongs (resp. does not belong) to the upper hull of S.

At this moment, we know the following: For any two indices i and j, if $A[i\sqrt{n}+j] = 1$, then q^i_j is a point on the upper hull of S, and among all points on this upper hull, it is the one having the $(A[1]+A[2]+\cdots+A[k])$-th smallest x-coordinate, where $k = i\sqrt{n}+j$.

We obtain our final array by computing the values $B[k] := \sum_{l=1}^{k} A[l]$, $1 \leq k \leq n$, and storing each point q^i_j for which $A[i\sqrt{n}+j] = 1$, at the position $B[i\sqrt{n}+j]$ of our final array.

Computing the values $B[k]$, $1 \leq k \leq n$, is known as the *prefix sum problem.* This problem can be solved on a CREW PRAM in $O(\log n)$ time, using $O(n/\log n)$ processors.

This concludes the description of the algorithm. We analyze its complexity. We mentioned already that the initial sorting step takes $O(\log n)$ time, using $O(n)$ processors.

Let $t(n)$ denote the parallel time needed to compute the convex hull of n

points that are sorted by their x-coordinates. Then

$$t(n) = t(\sqrt{n}\,) + O(\log n).$$

It follows that $t(n) = O(\log n)$. (Prove this!) Also, it is not hard to see that the entire algorithm uses $O(n)$ processors. (Of course, in recursive steps, processors are "re-used".) This proves the following result.

Theorem 3 *On a CREW PRAM, the convex hull of a set of n points in the plane can be computed in $O(\log n)$ time, using $O(n)$ processors.*

Corollary 1 *On a CREW PRAM, the diameter of a set of n points in the plane can be computed in $O(\log n)$ time, using $O(n)$ processors.*

At the beginning of Section 4, we mentioned that the results of Theorem 3 and Corollary 1 are optimal. Why is this? We know that the convex hull problem and the diameter problem both have an $\Omega(n \log n)$ lower bound on a *sequential* machine. Can we use this to prove the optimality of our PRAM algorithms? The answer is "yes":

Theorem 4 *Assume there is a PRAM algorithm that solves a problem $\mathcal{P}$ in time $t(n)$, using $s(n)$ processors. Then, problem $\mathcal{P}$ can be solved on a sequential machine in $O(t(n) \cdot s(n))$ time.*

Proof. Simulate the parallel algorithm on a sequential machine. In each parallel step, each of the $s(n)$ processors performs one operation. Hence, each parallel step can be simulated in $O(s(n))$ sequential steps. Since there are $t(n)$ parallel steps, the entire simulation takes $O(t(n) \cdot s(n))$ time. (Of course, during the simulation, we have to store intermediate results of the different processors in an appropriate way.) ■

Corollary 2 *Let A be any parallel algorithm that solves the convex hull problem or the diameter problem. Let $t(n)$ (resp. $s(n)$) be the parallel running time of (resp. the number of processors used by) this algorithm. Then*

$$t(n) \cdot s(n) = \Omega(n \log n).$$

As a result, the algorithms of Theorem 3 and Corollary 1 are optimal.

5 Further reading

For a good introduction to PRAM algorithms, see the book

- T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* MIT Press, Cambridge, Mass., 1990.

The following book is one of the best sources for parallel algorithms.

- J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, Mass., 1992.