Convex Hulls and Triangulations of Planar Point Sets on the Congested Clique

Jesper Jansson* Christos Levcopoulos† Andrzej Lingas‡

Abstract

We consider geometric problems on planar n^2-point sets in the congested clique model. Initially, each node in the n-clique network holds a batch of n distinct points in the Euclidean plane given by $O(\log n)$-bit coordinates. In each round, each node can send a distinct $O(\log n)$-bit message to each other node in the clique and perform unlimited local computations. We show that the convex hull of the input n^2-point set can be constructed in $O(\min\{h, \log n\})$ rounds, where h is the size of the hull, on the congested clique. We also show that a triangulation of the input n^2-point set can be constructed in $O(\log^2 n)$ rounds on the congested clique.

1 Introduction

The communication/computation model of congested clique focuses on the communication cost and ignores that of local computation. It can be seen as a reaction to the criticized model of Parallel Random Access Machine (PRAM), studied in the 80s and 90s, which focuses on the computation cost and ignores the communication cost [1].

In recent decades, the complexity of dense graph problems has been intensively studied in the congested clique model. Typically, each node of the clique network initially represents a distinct vertex of the input graph and knows that vertex's neighborhood in the input graph. Then, in each round, each of the n nodes can send a distinct message of $O(\log n)$ bits to each other node and can perform unlimited local computation. Several dense graph problems, for example, the minimum spanning tree problem, have been shown to admit $O(1)$-round algorithms in the congested clique model [10]. Note that when the input graph is of bounded degree, each node can send its whole information to a distinguished node in $O(1)$ rounds. The distinguished node can then solve the graph problem locally. However, when the input graph is dense such a trivial solution requires $\Omega(n)$ rounds.

*Graduate School of Informatics, Kyoto University, Kyoto, Japan, jj@i.kyoto-u.ac.jp

†Department of Computer Science, Lund University, 22100 Lund, Sweden, Christos.Levcopoulos@cs.lth.se

‡Department of Computer Science, Lund University, 22100 Lund, Sweden, Andrzej.Lingas@cs.lth.se

Researchers have also successfully studied problems not falling in the category of graph problems, like matrix multiplication [3] or sorting and routing [6], in the congested clique model. In both cases, one assumes that the basic items, i.e., matrix entries or keys, respectively, have $O(\log n)$ bit representations and that each node initially has a batch of n such items. As in the graph case, each node can send a distinct $O(\log n)$-bit message to each other node and perform unlimited computation in every round. Significantly, it has been shown that matrix multiplication admits an $O(n^{1-2/\omega})$-round algorithm [3], where ω is the exponent of fast matrix multiplication, while sorting and routing admit $O(1)$-round algorithms [6] under the aforementioned assumptions.

We extend this approach to include basic geometric problems on planar point sets. These problems are generally known to admit polylogarithmic time solutions on PRAMs with a polynomial number of processors [1]. Initially, each node of the n-clique network holds a batch of n points belonging to the input set S of n^2 points with $O(\log n)$-bit coordinates in the Euclidean plane. As in the graph, matrix, sorting, and routing cases, in each round, each node can send a distinct $O(\log n)$-bit message to each other node and perform unlimited local computations. Analogously, trivial solutions consisting in gathering the whole data in a distinguished node require $\Omega(n)$ rounds.

First, we provide a simple implementation of the Quick Convex Hull algorithm [9], showing that the convex hull of S can be constructed in $O(h)$ rounds on the congested clique, where h is the size of the hull. Then, we present and analyze a more refined algorithm for the convex hull of S on the congested clique running in $O(\log n)$ rounds. Finally, we present a divide-and-conquer method for constructing a triangulation of S in $O(\log^2 n)$ rounds on the congested clique.

2 Preliminaries

For a positive integer r, $[r]$ stands for the set of positive integers not exceeding r.

Let $S = \{p_1, ..., p_n\}$ be a set of n distinct points in the Euclidean plane such that the x-coordinate of each point is not smaller than that of p_1 and not greater than that of p_n. The *upper hull* of S (with respect to (p_1, p_n)) is the part of the convex hull of S beginning in

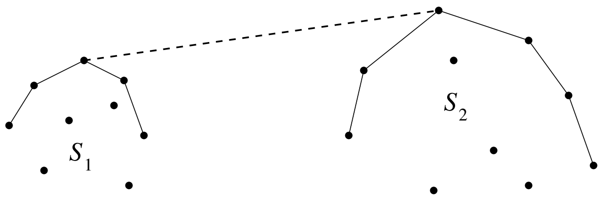

Figure 1: An example of the bridge between the upper hulls of S_1 and S_2.

p_1 and ending in p_n in clockwise order. Symmetrically, the *lower hull* of S (with respect to (p_1, p_n)) is the part of the convex hull of S beginning in p_n and ending in p_1 in clockwise order. A *supporting line* for the convex hull or upper hull or lower hull of a finite point set in the Euclidean plane is a straight line that touches the hull without crossing it properly.

Let S_1, S_2 be two finite sets of points in the Euclidean plane separated by a vertical line. The *bridge* between the upper (or lower) hull of S_1 and the upper (or, lower, respectively) hull of S_2 is a straight line that is a supporting line for the both upper (lower, respectively) hulls. See Figure 1 for an illustration.

3 Quick Convex Hull Algorithm on Congested Clique

The Quick Convex Hull Algorithm is well known in the literature, see, e.g, [9]. Roughly, we shall implement it as follows in the congested clique model. First, the set S of n^2 input points with $O(\log n)$-bit coordinates is sorted by their x-coordinates [6]. As a result, each consecutive clique node gets a consecutive n-point fragment of the sorted S. Next, each node informs all other nodes about its two extreme points along the x axis. By using this information, each node can determine the same pair of extreme points p_{min}, p_{max} in S along the x axis. Using this extreme pair, each node can decompose its subsequence of S into the upper-hull subsequence consisting of the points that lie above or on the segment (p_{min}, p_{max}) and the lower-hull subsequence consisting of points that lie below or on (p_{min}, p_{max}). From now on, the upper hull of S and the lower hull of S are computed separately by calling the procedures $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$, respectively. The former procedure proceeds as follows. Each node sends a point q of highest y-coordinate among those in its upper-hull subsequence different from p_{min} and p_{max} to all other nodes. Then, each node selects the same point q of maximum y-coordinate among all points in the whole upper-hull subsequence different from p_{min} and p_{max}. Note that q must be a vertex of the upper hull of S. Two recursive calls $QuickUpperHull(p_{min}, q)$ and $QuickUpperHull(q, p_{max})$ follow, etc. The procedure $QuickLowerHull$ is defined symmetrically. As each non-leaf call of these two procedures results in a new vertex of the convex hull, and each step of these procedures but for the recursive calls takes $O(1)$ rounds, the total number of rounds necessary to implement the outlined variant of Quick Convex Hull algorithm, specified in the procedure $QuickConvexHull(S)$, is proportional to the size of the convex hull of S.

procedure $QuickConvexHull(S)$
Input: A set of n^2 points in the Euclidean plane with $O(\log n)$ bit coordinates, each node holds a batch of n input points.
Output: The vertices of the convex hull of S held in clockwise order in consecutive nodes in batches of at most n vertices.

1. Sort the points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S, in the sorted order.
2. Each node sends the first point and the last point in its subsequence to the other nodes.
3. Each node computes the same point p_{max} of the maximum x-coordinate and the same point p_{min} of the minimum x-coordinate in the whole input sequence S based on the gathered information. Next, it decomposes its sorted subsequence into the upper hull subsequence consisting of points above or on the segment connecting p_{max} and p_{min} and the lower hull subsequence consisting of the points lying below or on this segment. In particular, the points p_{min} and p_{max} are assigned to both upper and lower hull subsequences of the subsequences they belong to.
4. Each node sends its first and last point in its upper hull subsequence as well as its first and last point in its lower hull subsequence to all other nodes.
5. $QuickUpperHull(p_{min}, p_{max})$
6. $QuickLowerHull(p_{min}, p_{max})$
7. By the previous steps, each node keeps consecutive pieces (if any) of the upper hull as well as the lower hull. However, some nodes can keep empty pieces. In order to obtain a more compact output representation in batches of n consecutive vertices of the hull (but for the last batch) assigned to consecutive nodes of the clique, the nodes can count the number of vertices on the upper and lower hull they hold and send the information to the other nodes. Using the global information, they can design destination addresses for their vertices on both hulls. Then, the routing protocol from [6] can be applied.

procedure $QuickUpperHull(p, r)$
Input: The upper-hull subsequence of the input point set S held in consecutive nodes in batches of at most n points and two distinguished points p, r in the subsequence , where the x-coordinate of p is smaller than that of r.
Output: The vertices of the upper hull of S with x-coordinates between those of p and r held in clockwise order in consecutive nodes, between those holding p and r respectively, in batches of at most n points.

1. Each node u determines the set S_u of points in its upper-hull subsequence that have x-coordinates between those of p and r and lie above or on the segment between p and r. If S_u is not empty then the node sends a point in S_u having the largest y-coordinate to the clique node holding p, from here on referred to as the *master node*.

2. If the master node has not received any point satisfying the requirements from the previous step then it proclaims p and r to be vertices of the upper hull by sending this information to the nodes holding p and/or q, respectively. (In fact one of the vertices p and r has been marked as being on the upper hull earlier.) Next, it pops a call of $QuickUpperHull$ from the top of a stack of recursive calls held in a prefix of the clique nodes numbered 1, 2, ... In case the stack is empty it terminates $QuickUpperHull(p_{min}, p_{max})$.

3. If the master has received some points satisfying the requirements from Step 1 than it determines a point q of maximum y-coordinate among them. Next, it puts the call of $QuickUpperHull(q, r)$ on the top of the stack and then activates $QuickUpperHull(p, q)$.

The procedure $QuickLowerHull(p, r)$ is defined analogously.

Each step of the procedure $QuickConvexHull(S)$, but for the calls to $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$ can be done in $O(1)$ rounds on the congested clique on n nodes. In particular, the sorting and the routing steps in $QuickConvexHull(S)$ can be done in $O(1)$ rounds by [6]. Similarly, each step of $QuickUpperHull(p, r)$, and symmetrically each step of $QuickLowerHull(p, r)$, but for recursive calls, can be done in $O(1)$ rounds. Since each non-leaf (in the recursion tree) call of $QuickUpperHull(p, r)$ and $QuickLowerHull(p, r)$ results in a new vertex of the convex hull, their total number does not exceed h. Hence, we obtain the following theorem.

Theorem 1 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. Let h be the number of vertices on the convex hull of the set S of the n^2 points. The convex hull of S can be computed by the procedure $QuickConvexHull(S)$ in $O(h)$ rounds on the congested clique.*

4 An $O(\log n)$-round Algorithm for Convex Hull on Congested Clique

Our refined algorithm for the convex hull of the input point set S analogously as $QuickConvexHull(S)$ starts by sorting the points in S by their x-coordinates and then splitting the sorted sequence of points in S into an upper-hull subsequence and lower-hull subsequence. Next, it computes the upper hull of S and the lower hull of S by calling the procedures $NewUpperHull(s)$ and $NewLowerHull(S)$, respectively. The procedure $NewUpperHull(S)$ lets each node ℓ construct the upper hull H_ℓ of its batch of at most n points in the upper-hull subsequence locally. The crucial step of $NewUpperHull(S)$ is a parallel computation of bridges between all pairs H_ℓ, H_m, $\ell \neq m$, of the constructed upper hulls by parallel calls to the procedure $Bridge(H_\ell, H_m)$. Based on the bridges between H_ℓ and the other upper hulls H_m, each node ℓ can determine which of the vertices of H_ℓ belong to the upper hull of S (see Lemma 1). The procedure $Bridge$ has recursion depth $O(\log n)$ and the parallel implementation of the crucial step of $NewUpperHull(s)$ takes $O(\log n)$ rounds. The procedure $NewLowerHull(s)$ is defined symmetrically. Consequently, the refined algorithm for the convex hull of S specified by the procedure $NewConvexHull(S)$ can be implemented in $O(\log n)$ rounds.

The procedure $NewConvexHull(S)$ is defined in exactly the same way as $QuickConvexHull(S)$, except that the calls to $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$ are replaced by calls to $NewUpperHull(S)$ and $NewLowerHull(S)$, respectively.

procedure $NewUpperHull(S)$
Input: The upper-hull subsequence of the input point set S held in consecutive nodes in batches of at most n points.
Output: The vertices of the upper hull of S held in clockwise order in consecutive nodes in batches of at most n vertices.

1. Each node ℓ computes the upper hull H_ℓ of its upper-hull subsequence locally.

2. In parallel, for each pair ℓ, m of nodes, the procedure $Bridge(H_\ell, H_m)$ computing the bridge between H_ℓ and H_m is called. (The procedure uses the two nodes in $O(\log n)$ rounds, exchanging at most two messages between the nodes in each of these rounds.)

3. Each node ℓ checks if it has a single point p not marked as not qualifying for the upper hull of S such that there are bridges between H_k and H_ℓ and H_ℓ and H_m, where $k < \ell < m$, p is an endpoint of both bridges, and the angle formed by the two bridges is smaller than 180 degrees. If so, p is also marked as not qualifying for the upper hull of S.

4. Each node ℓ prunes the set of vertices of H_l, leaving only those vertices that have not been marked in the previous steps (including calls to the procedure *Bridge*) as not qualifying for the upper hull of S.

The following lemmata enable the implementation of the n^2 calls to $Bridge(H_\ell, H_m)$ in the second step of $NewUpperHull(S)$ in $O(\log n)$ rounds on the congested clique.

Lemma 2 *For $\ell \in [n]$, let H_ℓ be the upper hull of the upper-hull subsequence of S assigned to the node ℓ. A vertex v of H_ℓ is not a vertex of the upper hull of S if and only if it lies below a bridge between H_ℓ and H_m, where $\ell \neq m$, or there are two bridges between H_ℓ and H_s, H_t, respectively, where $s < \ell < t$, such that they touch v and form an angle of less than 180 degrees at v.*

Proof. Clearly, if at least one of the two conditions on the right side of "if and only if" is satisfied then v cannot be a vertex of the upper hull of S. Suppose that v is not a vertex of the upper hull of S. Then, since it is a vertex of H_l, there must be an edge e of the upper hull of S connecting H_k with H_m for some $k \leq \ell \leq m$, $k \neq m$, that lies above v. We may assume without loss of generality that v does not lie below any bridge between H_ℓ and H_q, $\ell \neq q$. It follows that $s < \ell < t$. Let b_k be the bridge between H_k and H_ℓ, and let b_m be the bridge between H_ℓ and h_m. It also follows that both b_k and b_m are placed below e and the endpoint of b_k at H_ℓ is v or a vertex of H_ℓ to the left of v while the endpoint of b_m at H_ℓ is v or a vertex to the right of v. Let C be the convex chain that is a part of H_ℓ between the endpoints of b_k and b_m on H_ℓ. Suppose that C includes at least one edge. The bridge b_k has to form an angle not less than 180 degrees with the leftmost edge of C and symmetrically the bridge b_m has to form an angle not less than 180 degrees with the rightmost edge of C. However, this is impossible because the bridges b_k and b_m are below the edge e of the upper hull of S with endpoints on H_k and H_m so they form an angle less than 180 degrees. We conclude that C consists solely of v and consequently v is an endpoint of both b_k and b_m. See Figure 2. □

The following folklore lemma follows easily by a standard case analysis (cf. [5, 7, 8]). It implies that the recursive depth of the procedure *Bridge* is $O(\log n)$.

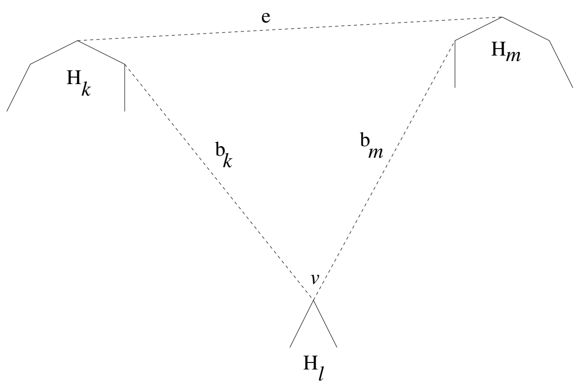

Figure 2: The final case in the proof of Lemma 2.

Lemma 3 *Let S_1, S_2 be two n-point sets in the Euclidean plane separated by a vertical line. Let H_1, H_2 be the upper hulls of S_1, S_2, respectively. Suppose that each of H_1 and H_2 has at least three vertices. Next, let m_1, m_2 be the median vertices of H_1, H_2, respectively. Suppose that the segment connecting m_1 with m_2 is not the bridge between H_1 and H_2. Then, the vertices on H_1 either to the left or to the right of m_1, or the vertices on H_2 either to the left or to the right of m_2 cannot be an endpoint of the bridge between H_1 or H_2.*

procedure $Bridge(H'_\ell, H'_m)$
Input: A continuous fragment H'_ℓ of the upper hull H_ℓ of the upper-hull subsequence assigned to a node ℓ and a continuous fragment H'_m of the upper hull H_m of the upper-hull subsequence assigned to the node m.
Output: The bridge between H'_ℓ and H'_m. Moreover, all points in the upper-hull subsequence held in the nodes ℓ and m placed under the bridge are marked as not qualifying for the convex hull of S.

1. If H'_ℓ or H'_m has at most two vertices then compute the bridge between H'_ℓ and H'_m by binary search. Next, mark all the points in the upper-hull subsequence between the endpoints of the found bridge that are assigned to the nodes ℓ or m as not qualifying for vertices of the upper hull of S and stop.

2. Find a median m_1 of H'_ℓ and a median m_2 of H'_m.

3. If the straight line passing through m_1 and m_2 is a supporting line for both H'_ℓ and H'_m then mark all the points in the upper-hull subsequence between m_1 and m_2 that are assigned to the nodes ℓ or m as not qualifying for vertices of the upper hull of S and stop.

4. Otherwise, call $Bridge(H''_\ell, H''_m)$, where either $H'_\ell = H''_\ell$ and H''_m is obtained from H'_m by removing vertices on the appropriate side of the median of H'_m or *vice versa*, according to Lemma 2.

The procedure $NewLowerHull(H'_\ell, H'_m)$ is defined analogously.

As in case of the procedure $QuickConvexHull(S)$, each step of $NewConvexHull(S)$, but for the calls to $NewUpperHull(S)$ and $NewLowerHull(S)$, can be done in $O(1)$ rounds on the congested clique by [6]. Furthermore, the first, next to the last, and last steps of $NewUpperHull(S)$ require $O(1)$ rounds. By Lemma 2, the recursion depth of the procedure $Bridge$ is logarithmic in n. The crucial observation is now that consequently the nodes ℓ and m need to exchange $O(\log n)$ messages in order to implement $Bridge(H_\ell, H_m)$. In particular, they need to inform each other about the current medians and in case H'_ℓ or H'_m contains at most two vertices, the node ℓ or m needs to inform about the situation and the two vertices the other node. In consequence, by Lemma 1, these two nodes can implement $Bridge(H_\ell, H_m)$ by sending a single message to each other in each round in a sequence of $O(\log n)$ consecutive rounds. It follows that all the n^2 calls of $Bridge(H_\ell, H_m)$ can be implemented in parallel in $O(\log n)$ rounds. Note that in each of the $O(\log n)$ rounds, each clique node sends at most one message to each other clique node, so in total, each node sends at most $n-1$ messages to the other nodes in each of these rounds. It follows that $NewUpperHull(S)$ and symmetrically $NewLowerHull(S)$ can be implemented in $O(\log n)$ rounds on the congested clique. We conclude that $NewConvexHull(S)$ can be done in $O(\log n)$ rounds on the congested clique.

Theorem 4 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. The convex hull of the set S of the n^2 input points can be computed by the procedure $NewConvexHull(S)$ in $O(\log n)$ rounds on the congested clique.*

5 Point Set Triangulation in $O(\log^2 n)$ Rounds on Congested Clique

Our method of triangulating a set of n^2 points in the congested n-clique model initially resembles that of constructing the convex hull of the points. That is, first the input point set is sorted by x-coordinates. Then, each node triangulates its sorted batch of n points locally. Next, the triangulations are pairwise merged and extended to triangulations of doubled point sets by using the procedure $Merge$ in parallel in $O(\log n)$ phases. In the general case, the procedure $Merge$ calls the procedure $Triangulate$ in order to triangulate the area between the sides of the convex hulls of the two input triangulations, facing each other.

The main idea of the procedure $Triangulate$ is to pick a median vertex on the longer of the convex hulls sides and send its coordinates and the coordinates of its neighbors to the nodes holding the facing side of the other hull. The latter nodes send back candidates (if any) for a mate of the median vertex so that the segment between the median vertex and the mate can be an edge of a triangulation extending the input ones. The segment is used to split the area to triangulate into two that are triangulated by two recursive calls of $Triangulate$ in parallel. Before the recursive calls the edges of the two polygons surrounding the two areas are moved to new node destinations so each of the polygons is held by a sequence of consecutive clique nodes. This is done by a global routing in $O(1)$ rounds serving all parallel calls of $Triangulate$ on a given recursion level, for a given phase of $Merge$ (its first argument).

Since the recursion depth $Triangulate$ is $O(\log n)$ and $Merge$ is run in $O(\log n)$ phases, the total number of required rounds becomes $O(\log^2 n)$.

To simplify the presentation, we shall assume that the size n of the clique network is a power of 2.

procedure $Triangulation(S)$

1. Sort the points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S, in the sorted order.
2. Each node sends the first point and the last point in its subsequence to the other nodes.
3. Each node q constructs a triangulation $T_{q,q}$ of the points in its sorted subsequence locally.
4. For $1 \le p < q \le n$, $T_{p,q}$ will denote the already computed triangulation of the points in the sorted subsequence held in the nodes p through q. For $i = 0, \log n - 1$, in parallel, for $j = 1, 1 + 2^{i+1}, 1 + 22^{i+1}, 1 + 32^{i+1}, ...$ the union of the triangulations $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$ is transformed to a triangulation $T_{j,j+2^{i+1}-1}$ of the sorted subsequence held in the nodes j through $j + 2^{i+1} - 1$ by calling the procedure $Merge(i, j)$.

procedure $Merge(i, j)$
Input: A triangulation $T_{j,j+2^i-1}$ of the subsequence held in the nodes j through $j + 2^i - 1$ and a triangulation $T_{j+2^i,j+2^{i+1}-1}$ of the subsequence held in the nodes $j+2^i$ through $j + 2^{i+1} - 1$,.
Output: A triangulation $T_{j,j+2j+1-1}$ of the subsequence held in the nodes j through $j + 2^{j+1} - 1$.

1. Compute the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$. Determine the polygon P formed by the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$, the right side of the convex hull of $T_{j,j+2^i-1}$, and the left side of the convex hull of $T_{j+2^i,j+2^{i+1}-1}$ between the bridges.
2. $Triangulate(P, j, j + 2^{i+1} - 1)$

procedure $Triangulate(P, p, q)$
Input: A simple polygon P composed of two convex chains facing each other on opposite sides of a vertical line and two edges crossing the line, held in nodes p through q, with $p < q$.
Output: A triangulation of P held in nodes p through q.

1. If $p = q$ then the p node triangulates P locally and terminates the call of the procedure.

2. The nodes p through q determine the lengths of the convex chains on the border of P and the node holding the median vertex v of the longest chain (in case of ties, the left chain) sends the coordinates of v and the adjacent vertices on the chain to the other nodes p through q.

3. The nodes holding vertices of the convex chain that is opposite to the convex chain containing v determine if they hold vertices u that could be connected by a segment with v within P. They verify if the segment (v, u) is within the intersection of the union of the half-planes induced by the edges adjacent to v on the side of P with the union of the half-planes induced by the edges adjacent to u on the side of P. If so, they send one such a candidate vertex u to the node holding v.

4. The node holding v selects one of the received candidate vertices u as the mate and sends its coordinates to the other nodes p through q.

5. The nodes p through q split the polygon P into two subpolygons P_1 and P_2 by the edge (v, u) and by exchanging messages in $O(1)$ rounds compute the new destinations for the edges of the polygons P_1 and P_2 so P_1 can be held in nodes p through r_1 and P_2 in the nodes r_2 through q, where $p \leq r_1 \leq r_2 \leq q$ and $r_1 = r_2$ or $r_2 = r_1 + 1$.

6. A synchronized global routing in $O(1)$ rounds corresponding to the current phase of the calls to the procedure $Merge$ (given by its first argument) and all parallel calls of the procedure $Triangulate$ on the same recursion level is implemented. In particular, the edges of P_1 and P_2 are moved to the new consecutive destinations among nodes p through q.

7. In parallel, $Triangulate(P_1, p, r_1)$ and $Triangulate(P_2, r_2, q)$ are performed.

At the beginning, we have outlined our triangulation method, in particular the procedures forming it, in a top-down fashion. We now complement this outline with a bottom-up analysis. All steps of the procedure $Triangulate(P, p, q)$ but for the recursive calls in the last step and the next to the last step can be implemented in $O(1)$ rounds, using only the nodes p through q. The next to the last step is a part of the global routing. It serves all calls of the procedure $Triangulate$ on the same recursion level for a given phase of the parallel calls of procedure $Merge(i, \ )$, i.e., for given i. Since each node is involved in at most two of the aforementioned calls of $Triangulate$ that cannot be handled locally, the global routing, implementing the next to the last step of $Triangulate$, requires $O(1)$ rounds. Since the recursion depth of $Triangulate$ is $O(\log n)$, $Triangulate$ takes $O(\log n)$ rounds. The first step of the procedure $Merge(i, j)$, i.e., constructing the bridges between the convex hulls, can be implemented in $O(\log n)$ rounds by using the convex hull algorithm from Section 4 on nodes i through $i + 2^{i+1} - 1$. The second step can easily be implemented in $O(1)$ rounds using the aforementioned nodes. Finally, the call to $Triangulate$ in the last step of $Merge$ requires $O(\log n)$ rounds by our analysis of this procedure. Again, it can be done by nodes j through $j + 2^{i+1} - 1$ but for the last steps of calls to $Triangulate$ that are served by the discussed, synchronized global routing in $O(1)$ rounds. We conclude that $Merge(i, j)$ can be implemented in $O(\log n)$ rounds. Finally, all steps in $Triangulation(S)$ except the one involving parallel calls to $Merge(i, j)$ in $O(\log n)$ phases can be done in $O(1)$ rounds. For a given phase, i.e., given i, each node is involved in $O(1)$ calls of $Merge(i, j)$ but for the next to the last steps in $Triangulate$ that for a given recursion level of $Triangulate$ are implemented by the join global routing in $O(1)$ rounds. It follows from our analysis of $Merge(i, j)$ and $i = O(\log n)$ that $Triangulate(S)$ can be implemented in $O(\log^2 n)$ rounds.

Theorem 5 *Consider a congested n-clique network, where each node holds a batch of n points in the Euclidean plane specified by $O(\log n)$-bit coordinates. A triangulation of the set S of the n^2 input points can be computed by the procedure $Triangulation(S)$ in $O(\log^2 n)$ rounds on the congested clique.*

6 Remarks

The primary difficulty in the design of efficient parallel algorithms for the Voronoi diagram of a planar point set using a divide-and-conquer approach is the efficient parallel merging of Voronoi diagrams [1, 11]. In the full version of this paper [4], we show that when the n^2 input points with $O(\log n)$-bit coordinates are drawn uniformly at random from a unit square then the expected number of rounds required to build their Voronoi diagram on the congested clique is $O(1)$.

Acknowledgments

This research was partially supported by Swedish Research Council grants 621-2017-03750 and 2018-04001, and JSPS KAKENHI JP20H05964.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. Yap. Parallel Computational Geometry. *Algorithmica*, 3: 293–327, 1988. Preliminary version in *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, FOCS 1985, pp. 468–477, 1985.

[2] S.G. Akl. Optimal parallel algorithms for computing convex hulls and for sorting. *Computing*, 33(1): 1–11, 1984.

[3] K. Censor-Hillel, P. Kaski, J.H. Korhonen, C. Lenzen, C., A. Paz, and J. Suomela. Algebraic Methods in the Congested Clique. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC 2015, pp. 143–152, 2015.

[4] J. Jansson, C. Levcopoulos, and A. Lingas. Convex Hulls and Triangulations of Planar Point Sets on the Congested Clique. arXiv:2305.09987, 2023.

[5] D.G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1): 287–299, 1986.

[6] C. Lenzen. Optimal Deterministic Routing and Sorting on the Congested Clique. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC 2013, pp. 42–50, 2013.

[7] M.H. Overmars and J. Van Leeuwen. Maintenance of Configurations in the Plane. *Journal of Computer and System Sciences*, 23(2): 166–204, 1981.

[8] F. Preparata. An optimal real-time algorithm for planar convex hulls. *Communications of the ACM*, 22(7): 402–405, 1979.

[9] J. Ramesh and S. Suresha. Convex Hull - Parallel and Distributed Algorithms. Technical Report, Stanford University, U.S.A., 2016.

[10] P. Robinson. What Can We Compute in a Single Round of the Congested Clique? arXiv:2210.02638, 2022.

[11] B. C. Vemuri, R. Varadarajan and N. Mayya. An Efficient Expected Time Parallel Algorithm for Voronoi Construction. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, SPAA 1992, pp. 392–401, 1992.