An $O((\log\log n)^2)$ Time Convex Hull Algorithm on Reconfigurable Meshes*

Tatsuya Hayashi† Koji Nakano† Stephan Olariu‡

Abstract

It was open for more than eight years to obtain an algorithm for computing the convex hull of a set of n sorted points in sub-logarithmic time on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. Our main contribution is to provide the first breakthrough: we propose an almost optimal algorithm running in $O((\log\log n)^2)$ time on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. With slight modifications this algorithm can be implemented to run in $O((\log\log n)^2)$ time on a reconfigurable mesh of size $\frac{\sqrt{n}}{\log\log n} \times \frac{\sqrt{n}}{\log\log n}$. Clearly, the latter algorithm is work-optimal. We also show that any algorithm that computes the convex hull of a set of n sorted points on an n-processor reconfigurable mesh must take $\Omega(\log\log n)$ time. Our result opens the door to efficient convex-hull-based algorithms on reconfigurable meshes.
Keywords: *convex hulls, reconfigurable meshes, pattern recognition, morphology, image processing.*

1 Introduction

In essence, a reconfigurable mesh (RM) consists of a mesh augmented by the addition of a dynamic bus system whose configuration changes in response to computational and communication needs. More precisely, a RM of size $n \times m$ consists of nm identical SIMD processors positioned on a rectangular array with n rows and m columns. As usual, it is assumed that every processor knows its own coordinates within the mesh: we let $PE(i,j)$ denote the processor in row i and column j, with $PE(1,1)$ in the north-west corner of the mesh.

Each processor $PE(i,j)$ is connected to its four neighbors $P(i-1,j)$, $P(i+1,j)$, $P(i,j-1)$, and $P(i,j+1)$, provided they exist, and has 4 ports denoted by N, S, E, and W in Figure 1. Local connections between these ports can be established, under program control, creating a powerful bus system that changes dynamically to accommodate various computational needs. We note that some RM models proposed in the literature allow processors to fuse an arbitrary number of ports. Throughout this paper we assume a strictly *weaker* model that allows at most two pairs of connections to be set in each processor at any one time. Furthermore, these two connections must involve disjoint pairs of ports as illustrated in Figure 1. From a practical standpoint, our assumption precludes the setting of branch connections within processors whose net effect is to split and weaken the signal being broadcast. We also assume that broadcasts along subbuses take O(1) time. Although inexact, recent experiments with reconfigurable multiprocessor systems seem to indicate that this is a reasonable working hypothesis.

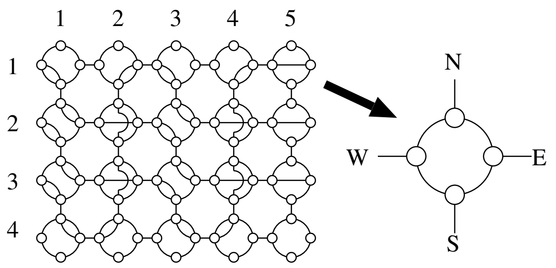

Figure 1. *A reconfigurable mesh of size 4 × 5 and several subbuses*

Given its importance, and its far-reaching applications, the convex hull problem has been studied extensively in the literature, both sequentially and in parallel [2, 3, 8]. On the reconfigurable mesh, the convex hull problem has been addressed in two different contexts: for *sparse input* and for *dense input*. While the sparse case allows one to use more processors than input points, in the dense case the number of processors and the number of input points are, essentially, the same. For sparse input, Olariu *et al.* [5] and Jang *et al.* [2] proposed $O(1)$ time algorithms to compute the convex hull of a set of $\sqrt{n}$ points on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. Nakano [4] showed that, if the $\sqrt{n}$ points are

*Work supported in part by NSF grant CCR-9522093, by ONR grant N00014-97-1-0526, by Grant-in-Aid for Encouragement of Young Scientists (09780262) from Ministry of Education, Science, Sports, and Culture of Japan, and by a grant from the Casio Science Promotion Foundation.
†Department of Electrical and Computer Engineering, Nagoya Institute of Technology, Showa-ku, Nagoya 466, JAPAN

‡Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529, USA

1	4	5	6	59	60	61	64
2	3	8	7	58	57	62	63
15	14	9	10	55	56	51	50
16	13	12	11	54	53	52	49
17	18	31	32	33	34	47	48
20	19	30	29	36	35	46	45
21	24	25	28	37	40	41	44
22	23	26	27	38	39	42	43

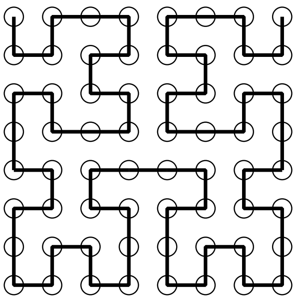

Figure 2. *Proximity order and the corresponding bus embedding*

sorted beforehand, then, for every fixed $\epsilon > 0$, the convex hull can be computed in $O(1)$ time on a reconfigurable mesh of size $\sqrt{n} \times n^{\epsilon}$.

In the dense case, Miller and Stout [3] proposed an $O(\log^2 n)$ time algorithm computing the convex hull of a sorted set of n points, pretiled in proximity order on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. Olariu *et al.* [6] using a different approach proposed algorithms featuring, essentially, the same $O(\log^2 n)$ performance. Recently, Nakano [4] showed that the convex hull of a sorted set of $\sqrt{mn}$ points can be computed in $O(\frac{\log^2 n}{\log m} + \log^2 m)$ time on a reconfigurable mesh of size $\sqrt{m} \times \sqrt{n}$. In particular, for $m = 2^{\log^{\frac{2}{3}} n}$ the computing time becomes $O(\log^{\frac{4}{3}} n)$, which is the fastest known for dense input.

For more than eight years it was open to obtain a convex hull algorithm running in sub-logarithmic time on dense input. Our main contribution is to provide the first breakthrough: we propose an almost optimal algorithm running in $O((\log \log n)^2)$ time on a reconfigurable mesh of size $\sqrt{n} \times \sqrt{n}$. We then go on to show that with minor changes this algorithm can be implemented to run in $O((\log \log n)^2)$ time on a reconfigurable mesh of size $\frac{\sqrt{n}}{\log \log n} \times \frac{\sqrt{n}}{\log \log n}$. Our second algorithm features a processor-time product of $O(n)$, being work-optimal.

We also show that our algorithms are close to time-optimality. Indeed, we prove that any algorithm that computes the convex hull of a set of n sorted points on an n-processor reconfigurable mesh must take $\Omega(\log \log n)$ time. In fact, this lower bound holds even for the strongest reconfigurable meshes that allow port fusion.

As in [3], we assume that the input is pretiled onto the platform in *proximity order*, as illustrated in Figure 2. With a bit of thought, it is easy to confirm that processors adjacent in proximity order are physically adjacent in the reconfigurable mesh. As an important consequence, we can *embed* the

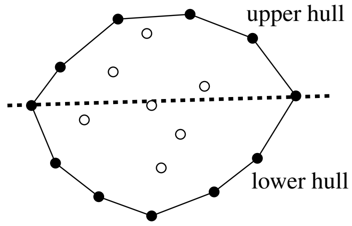

Figure 3. *Illustrating the upper and lower hull*

proximity order into a bus traversing the entire platform, as shown in Figure 2.

It is well known that the convex hull of a set P of points in the plane is a convex polygon having some of the points in P as vertices. As illustrated in Figure 3, the points of minimum and maximum x-coordinate partition the convex hull of P into the *upper hull* and the *lower hull*. For obvious reasons, we only focus on computing the upper hull.

2 The lower bound

In order to establish a time-lower bound for the problem of computing the upper hull of a set of n points on a reconfigurable mesh with n processors we rely on the following fundamental result of Valiant [9].

Lemma 2.1 *The problem of finding the maximum of n items requires $\Omega(\log \log n)$ time on the parallel comparison model, provided that the number of processors is at most $n \log^{O(1)} n$.*

We observe that, for comparison-based problems, the parallel comparison model with n processors can simulate without loss of time any parallel machine with n processors including the reconfigurable mesh. Thus, Lemma 2.1 has the following consequence.

Corollary 2.2 *The problem of finding the maximum of n items requires $\Omega(\log \log n)$ time on the reconfigurable mesh, provided that the number of processors is at most $n \log^{O(1)} n$.*

We now reduce the problem of finding the maximum of $\frac{n}{2}$ items to that of computing the upper hull of n sorted points in the plane. For this purpose, let $A = \{a_1, a_2, \ldots, a_{\frac{n}{2}}\}$ be an arbitrary input to the maximum problem. From this input we generate a set P of n sorted points in the plane by writing $P = \{(i, a_i), (i + \frac{n}{2}, a_i) \mid 1 \leq i \leq \frac{n}{2}\}$. Notice that the points (j, a_j) and $(j + \frac{n}{2}, a_j)$ are both vertices of the upper hull of P if and only if a_j is the maximum of A. It follows that any algorithm that computes the upper hull of P also computes the maximum of A. Thus, we have

Theorem 2.3 *The problem of computing the upper hull of n sorted points in the plane requires $\Omega(\log \log n)$ time on an n-processor reconfigurable mesh.*

3 Our geometric machinery

In this section we develop novel geometric results that will lay the foundation of our convex hull algorithms. Let $P_1, P_2, \ldots, P_m$ be disjoint upper hulls, each containing n points, such that for all i, $(1 \le i \le m-1)$, P_i is to the left of P_{i+1}. We are interested in computing the upper hull $U(P)$ of $P = P_1 \cup P_2 \cup \cdots \cup P_m$. For every i, $(1 \le i \le m)$, enumerate the points of P_i in increasing x-coordinate as $p(i,1), p(i,2), p(i,3), \ldots, p(i,n)$. Observe that $U(P)$ is a convex polygonal chain with left endpoint $p(1,1)$ and right endpoint $p(m,n)$. It is easy to see that the points belonging to $U(P) \cap P_i$ are consecutive in $U(P)$ and can be specified by the interval $[g_i, h_i]$: in other words, $U(P) \cap P_i = \{p(i,g_i), p(i,g_i+1), \ldots, p(i,h_i)\}$. The two points $L_i = p(i,g_i)$ and $R_i = p(i,h_i)$ are termed, respectively, the *left contact point* and the *right contact point* of P_i with respect to $U(P)$. Note that each P_i may have, with respect to $U(P)$, one, two, or no contact points. Clearly, P_i has exactly one contact point only if $L_i = R_i$.

The line segment $p(i,h_i)p(j,g_j)$, $(i < j)$, is said to be an *upper hull tangent* of $U(P)$ if for every k, $(i < k < j)$, $U(P) \cap P_k$ is empty. In other words, all the points in $P_{i+1} \cup P_{i+2} \cup \cdots \cup P_{j-1}$ lie below the line segment $p(i,h_i)p(j,g_j)$.

Samples will play a crucial role in the sequel of this work. For our purpose, a *sample* is just a subset of a given set. In particular, let $S(P_i)$ be a sample of P_i including the points $p(i,1)$ and $p(i,n)$. Let $U(S(P))$ be the upper hull of $S(P) = S(P_1) \cup S(P_2) \cup \cdots \cup S(P_m)$. It should be clear that $U(S(P))$ is also a convex polygonal chain with left endpoint $p(1,1)$ and right endpoint $p(m,n)$. As noted, $S(P_i)$ may have one, two, or no contact points with respect to $U(S(P))$. If $S(P_i)$ has no contact points with respect to $U(S(P))$, then it must be the case that all the points in $S(P_i)$ lie below some upper hull tangent of $U(S(P))$. In this case, the closest point $p(i,g'_i)$ to this upper hull tangent over all the points in $S(P_i)$ is termed a *pseudo contact point* of $S(P_i)$ with respect to $U(S(P))$. We refer the reader to Figure 4 for an illustration of these concepts. Here, the upper hulls are represented as parabolic curves, while the points in $S(P)$ are denoted by dark circles.

For a point $p(i,f)$ of $S(P_i)$ we let $p(i,l(f))$ and $p(i,r(f))$ denote, respectively, its left and right neighbor in $S(P_i)$. Let $p(i,g'_i)$ and $p(i,h'_i)$ be, respectively, the left and right contact points of $S(P_i)$ with respect to $U(S(P))$.

Lemma 3.1 *Let $p(i,g_i)$ and $p(i,h_i)$ be the left and right contact points (if any) of P_i with respect to $U(P)$. If the points $p(i,g'_i)$ and $p(i,h'_i)$ are distinct then $l(g'_i) < g_i \le h_i < r(h'_i)$. If $p(i,g'_i)$ and $p(i,h'_i)$ coincide or if $S(P_i)$ has a pseudo contact point $p(i,g'_i)$ with respect to $U(S(P))$, then $l(g'_i) < g_i \le h_i < r(g'_i)$.*

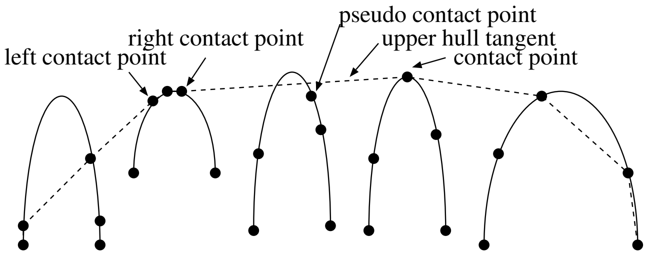

Figure 4. *Illustrating contact points and upper hull tangents of $U(S(P))$*

Proof. First, assume that the points $p(i,g'_i)$ and $p(i,h'_i)$ exist and are distinct and refer to Figure 5. Let $p(i,h'_i)p(j,g'_j)$, $(i < j)$, be an upper hull tangent of $U(S(P))$. Clearly, the points $p(i,k)$ with $r(h'_i) \le k$ lie below the upper hull tangent $p(i,h'_i)p(j,g'_j)$. Hence, $p(i,h_i)$ cannot be among them. Therefore, $h_i < r(h'_i)$ must hold. A symmetric argument shows that $l(g'_i) < g_i$. Thus, we have $l(g'_i) < g_i \le h_i < r(h'_i)$.

The case where the points $p(i,g'_i)$ and $p(i,h'_i)$ coincide is handled in a perfectly similar way and will not be repeated.

Finally, assume that $S(P_i)$ has a pseudo contact point $p(i,g'_i)$ and let λ be the upper hull tangent of $U(S(P))$ confirming that $p(i,g'_i)$ is a pseudo contact point. Clearly, the points $p(i,k)$ with $k \le l(i,g'_i)$ or $r(i,g'_i) \le k$ must lie below λ. Thus, neither $p(i,g_i)$ nor $p(i,h_i)$ can be among them and $l(g'_i) < g_i \le h_i < r(g'_i)$. □

Let $p(i,h_i)p(j,g_j)$, $(i < j)$, be an upper hull tangent of $U(P)$. As we noted before, $p(i,h_i)$ is the right contact point of P_i, $p(j,g_j)$ is the left contact point of P_j, and all the points in $P_{i+1} \cup P_{i+2} \cup \cdots \cup P_{j-1}$ lie below the line determined by $p(i,h_i)$ and $p(j,g_j)$.

Lemma 3.2 *Let $p(i,h'_i)$ be the right contact point or the pseudo contact point of $S(P_i)$ with respect to $U(S(P))$, and let $p(j,g'_j)$ be the left contact point or the pseudo contact point of $S(P_j)$ with respect to $U(S(P))$. At least one of $l(h'_i) < h_i < r(h'_i)$ or $l(g'_j) < g_j < r(g'_j)$ is satisfied.*

Proof. Let $p(i,h_i)p(j,g_j)$, $(i < j)$, be an upper hull tangent of $U(P)$ and refer to Figure 6. By Lemma 3.1 we must have

$$h_i < r(h'_i) \text{ and } l(g'_j) < g_j. \tag{1}$$

If both $h_i \le l(h'_i)$ and $r(g'_j) \le g_j$ are satisfied, then at least one of the points $p(i,h_i)$ and $p(j,g_j)$ must lie be-

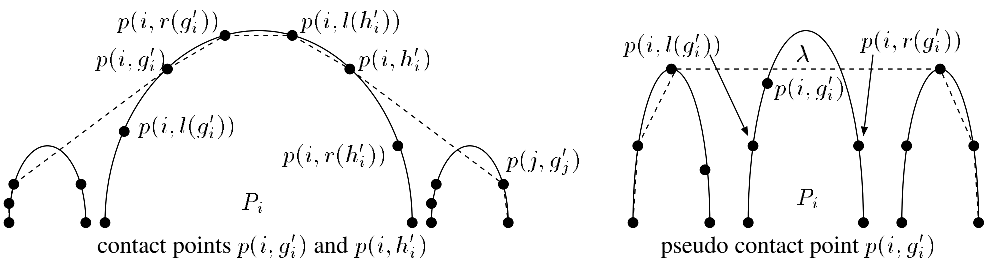

Figure 5. *Illustrating the proof of Lemma 3.1*

low the line segment determined by $p(i,h_i')$ and $p(j,g_j')$, a contradiction. Therefore, we must have

$$l(h_i') < h_i \text{ or } g_j < r(g_j'). \tag{2}$$

Now, equations (1) and (2), combined, imply that $l(h_i') < h_i < r(h_i')$ or $l(g_j') < g_j < r(g_j')$ is satisfied, completing the proof of the lemma. □

Let $p(i,h_i)p(j,g_j)$, $(i<j)$, be an upper hull tangent of $U(P)$ and assume that one of the points $p(i,h_i)$ or $p(j,g_j)$ is known.

Lemma 3.3 *Let $p(i,h_i')$ be the right contact point of $S(P_i)$ and let $p(j,g_j')$ be the left contact point of $S(P_j)$ with respect to $U(S(P))$. If $p(j,g_j')$ is also the right contact point of P_j with respect to $U(P)$, i.e. $g_j' = g_j$, then we must have $l(h_i') < h_i < r(h_i')$.*

Proof. Referring to Figure 7, note that among the points in P_i, only those lying on or above the line $p(i,h_i')p(j,g_j)$ can be the right contact point $p(i,h_i)$. □

We shall say that $S(P)$ is a *good sample* of P if all the contact points of P with respect to $U(P)$ belong to $S(P)$. Clearly, in this case, the contact points of $S(P)$ with respect to $U(S(P))$ are precisely the contact points of P with respect to $U(P)$. Thus, once a good sample $S(P)$ is available, the task of computing all the contact points of P with respect to $U(P)$ and, therefore, $U(P)$ itself, reduces to the task of computing $U(S(P))$.

4 Obtaining a good sample efficiently

We are now in a position to show how the geometric machinery developed in Section 3 can be exploited to obtain a good sample $S(P)$ of P. For this purpose, we shall find it convenient to import the relevant notation and terminology developed in the context of Section 3.

We begin by selecting for every i, $(1 \le i \le m)$, a sample $S(P_i)$ by retaining every $n^{\frac{1}{2}}$-th item in P_i. Note that the first and last point of P_i are also included in $S(P_i)$. Write $S(P) = S(P_1) \cup S(P_2) \cup \cdots \cup S(P_m)$. Having computed the convex hull $U(S(P))$ of $S(P)$, we determine for every $S(P_i)$ its left and right contact points, $p(i,g_i')$ and $p(i,h_i')$ (if any) with respect to $U(S(P))$. Lemma 3.2 guarantees that the left and right contact points $p(i,g_i)$ and $p(i,h_i)$ of P_i with respect to $U(P)$ satisfy $l(g_i') < g_i < r(g_i')$ or $l(h_i') < h_i < r(h_i')$. This motivates us to refine the sample $S(P_i)$ by adding more sample points from the sequence

$$p(i,l(g_i')),\ldots,p(i,g_i'),\ldots,p(i,r(g_i'))$$

and, similarly, from the sequence

$$p(i,l(h_i')),\ldots,p(i,h_i'),\ldots,p(i,r(h_i')).$$

The initial sample $S(P_i)$ is refined, in the way described, $2\log\log n - 2$ times. Somewhat surprisingly, what results is a good sample $S(P)$ of P. The details follow.

Algorithm `Find-good-sample`

Step 1 For every i, $(1 \le i \le m)$, build sample $S(P_i)$ by retaining every $n^{\frac{1}{2}}$-th item in P_i.

Step 2 Repeat the following substeps $2\log\log n - 2$ times.

Step 2.1 Compute the upper hull $U(S(P))$ of $S(P)$ and determine the left and right contact points $p(i,g_i')$ and $p(i,h_i')$ of $S(P_i)$ with respect to $U(S(P))$. If $S(P_i)$ has no contact point, determine the pseudo contact point $p(i,g_i')$ of $S(P_i)$.

Step 2.2 Let $p(i,l(g_i'))$ and $p(i,r(g_i'))$ be, respectively, the left and right neighbor of $p(i,g_i')$ in $S(P_i)$; similarly, let $p(i,l(h_i'))$ and $p(i,r(h_i'))$ be the left and right neighbor of $p(i,h_i')$ in $S(P_i)$. Update $S(P_i)$ by the addition of:

(1) every $(g_i' - l(g_i'))^{\frac{1}{2}}$-th item from the sequence $\{p(i,l(g_i')), p(i,l(g_i')+1),\ldots,p(i,g_i')\}$,

(2) every $(r(g_i') - g_i')^{\frac{1}{2}}$-th item from the sequence $\{p(i,g_i'), p(i,g_i'+1),\ldots,p(i,r(g_i'))\}$,

(3) every $(h_i' - l(h_i'))^{\frac{1}{2}}$-th item from the sequence $\{p(i,l(h_i')), p(i,l(h_i')+1),\ldots,p(i,h_i')\}$,

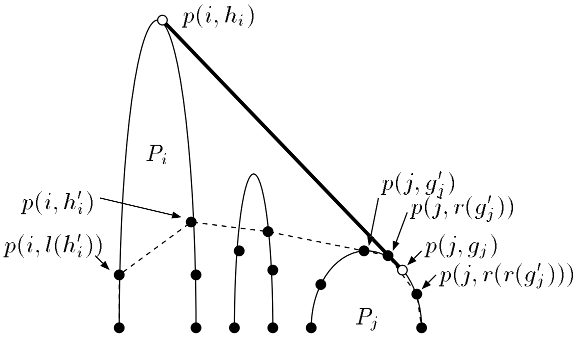

Figure 6. *Illustrating the proof of Lemma 3.2*

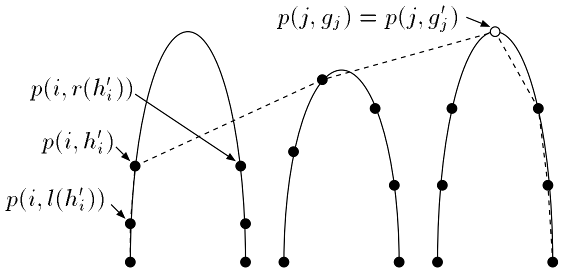

Figure 7. *Illustrating the proof of Lemma 3.3*

(4) every $(r(h'_i) - h'_i)^{\frac{1}{2}}$-th item from the sequence $\{p(i,h'_i), p(i,h'_i+1), \ldots, p(i,r(h'_i))\}$.

If $S(P_i)$ does not have distinct contact points, then only (1) and (2) are executed for the unique contact point or for the pseudo contact point $p(i,g'_i)$.

Let $p(i,g_i)$ and $p(i,h_i)$ be the left and right contact points (if any) of P_i with respect to $U(P)$. The *span* of $p(i,g_i)$ with respect to $S(P_i)$ is defined to be $\min\{g_i - l(g_i), r(g_i) - g_i\}$ if $p(i,g_i)$ belongs to $S(P_i)$; otherwise, with $p(i,f)$ standing for the (unique) point in $S(P_i)$ for which $l(f) < g_i < f$, the span is $f - l(f)$. The span of $p(i,h_i)$ is defined analogously. Intuitively, the span of point $p(i,g_i)$ measures the "density" of $S(P_i)$ around $p(i,g_i)$. Notice that $S(P)$ is a good sample whenever the span of all the contact points of P is 1.

Clearly, the span of a contact point of P_i with respect to $U(P)$ does not increase during the execution of the above algorithm, because no sample point is ever removed from $S(P)$. Somewhat surprisingly, the spans decrease quite fast. As we are about to show, when `Find-good-sample` terminates, the span of all the contact points is reduced to 1.

Fix an arbitrary upper hull tangent $p(i,h_i)p(j,g_j)$, ($i < j$), of $U(P)$. At the end of Step 1 the span of $p(i,h_i)$ and $p(j,g_j)$ is $n^{\frac{1}{2}}$. Consider what happens in the first iteration of Step 2. In Step 2.1 we determine the right contact point $p(i,h'_i)$ of $S(P_i)$ and the left contact point $p(j,g'_j)$ of $S(P_j)$ with respect to $U(S(P))$. Step 2.2 retains every $n^{\frac{1}{4}}$-th item from $\{p(i,l(h'_i)), p(i,l(h'_i)+1), \ldots, p(i,r(h'_i))\}$ and every $n^{\frac{1}{4}}$-th item from $\{p(j,l(g'_j)), p(j,l(g'_j)+1), \ldots, p(j,r(g'_j))\}$ and adds them to $S(P_i)$ and $S(P_j)$, respectively. Lemma 3.2 guarantees that at least one of $l(h'_i) < h_i < r(h'_i)$ or $l(g'_j) < g_j < r(g'_j)$ holds. Thus, at least one of the spans of $p(i,h_i)$ or $p(j,g_j)$ decreases from $n^{\frac{1}{2}}$ to $n^{\frac{1}{4}}$. More generally, we have the following result.

Lemma 4.1 *In Step 2.1 of each of the* $2\log\log n - 2$ *iterations of Step 2, at least one of the following conditions is satisfied:*

(a) *the span* s, ($s > 1$), *of* $p(i,h_i)$ *decreases to* $s^{1/2}$,

(b) *the span* s', ($s' > 1$), *of* $p(j,g_j)$ *decreases to* ${s'}^{1/2}$,

(c) *the span of both* $p(i,h_i)$ *and* $p(j,g_j)$ *is 1.*

Proof. If $s > 1$ and $s' > 1$ hold, then Lemma 3.2 ensures that either (a) or (b) is satisfied. If exactly one of $s > 1$ or $s' > 1$ holds, say, $s = 1$ and $s' > 1$ holds, then Lemma 3.3 ensures that (b) occurs. If both $s = 1$ and $s' = 1$, then clearly (c) holds. Hence, in each iteration, Step 2.1 decreases at least one of the spans of $p(i,h_i)$ or $p(j,h_j)$. □

Lemma 4.1 implies the main result of this section.

Theorem 4.2 *When algorithm* `Find-good-sample` *terminates,* $S(P)$ *is a good sample of* P.

Proof. Notice that condition (a) in Lemma 4.1 cannot hold true for more than $\log\log n - 1$ iterations. Similarly, condition (b) cannot hold true for more than $\log\log n - 1$ iterations. Therefore, at the end of $2\log\log n - 2$ iterations condition (c) must hold, as claimed. □

As it turns out, algorithm `Find-good-sample` has a number of additional properties that, collectively, allow us to implement each iteration in $O(1)$ time on the reconfigurable mesh. The proofs will be given in Section 6.

- In each iteration at most $4n^{1/4}$ new sample points are added to $S(P_i)$,
- The new sample points are "sparse" in the sense that from any sequence $p(i,f), p(i,f+1), \ldots,$ $p(i,f+s-1)$ of s points, at most $4s^{1/2}$ points are added to $S(P_i)$.
- $S(P)$ contains at most $mn^{1/2} + (2\log\log n - 2) \times 4mn^{1/4} \le 2mn^{1/2}$ sample points during the execution of algorithm.

5 Data movement and basic techniques

We begin by discussing a data movement technique that will be used time and again in the remainder of this work. Consider a set $A = \{a(1), a(2), \ldots, a(n)\}$ of $n = 2^d \times 2^d$ items, with some of the items marked. The set A is pretiled, in proximity order, on a reconfigurable mesh $\mathcal{M}$ of size $2^d \times 2^d$. We are interested in identifying, for every item in A, the *next marked item* in proximity order. More precisely, we wish to identify a set $A' = \{a'(1), a'(2), \ldots, a'(n)\}$ such that $a'(i) = a(j)$ with $j = \min\{k \mid k > i \text{ and } a(k) \text{ is marked}\}$. If no $a(k)$ with $k > i$ is marked, then $a'(i)$ is $\emptyset$. The task at hand involves (1) establishing a bus embedding of the proximity order as shown in Figure 2, (2) having each marked item segment the bus, and (3) mandating each marked item to broadcast its identity on the subbus containing items preceding it in proximity order. Thus, we have

Lemma 5.1 *Given a set of $n = 2^d \times 2^d$ items pretiled on a reconfigurable mesh of the same size with some of the items marked, the task of identifying for each item the next marked item in proximity order can be performed in $O(1)$ time.*

In essentially the same way one can identify the *previous marked item*, and the *leftmost marked item*, that is, the first marked item in proximity order.

Next, suppose that the set A is partitioned into $2^{2(d-d')}$ subsets $A(1), A(2), \ldots, A(2^{2(d-d')})$ of size $2^{d'} \times 2^{d'}$ each. Partition $\mathcal{M}$ into horizontal and vertical strips of size $2^{d'} \times 2^d$ and $2^d \times 2^{d'}$, respectively. For $1 \le i, j \le 2^{d-d'}$, let $M_{i,j}$ be the submesh of $\mathcal{M}$ determined by the intersection of the i-th horizontal and the j-th vertical strip. Each $A(k)$ is stored in one of the submeshes $M_{i,j}$.

The problem is to move the leftmost marked item in each $M_{i,j}$ to the first row of $\mathcal{M}$. To ensure that we have enough bandwidth to complete the data movement in $O(1)$ time, we insist that $2^d \ge 2^{2(d-d')}$ or, equivalently, that $d \le 2d'$. The details of the data movement follow.

Algorithm `Move-leftmost-items`

Step 1 Identify the leftmost marked item $a_{i,j}$ in each submesh $M_{i,j}$ and move $a_{i,j}$ vertically to the processor in row j of $M_{i,j}$;

Step 2 Move each item $a_{i,j}$ horizontally to the processor in column $(i-1)2^{d'} + j$ of $\mathcal{M}$;

Step 3 Move each item $a_{i,j}$ vertically to the first row of $\mathcal{M}$.

It is easy to confirm that `Move-leftmost-items` correctly moves all the marked items to the top row of $\mathcal{M}$. Since each step can be performed in $O(1)$ time, we have,

Lemma 5.2 *Given a reconfigurable mesh of size $2^d \times 2^d$ partitioned into non-overlapping submeshes of size $2^{d'} \times 2^{d'}$, with $\frac{d}{2} \le d' < d$, the task of moving the leftmost marked item in each submesh to the first row of the platform can be performed in $O(1)$ time.*

Suppose that each submesh $M_{i,j}$ of size $2^{d'} \times 2^{d'}$ contains $2^{d''}$ items stored in the leftmost $2^{d''}$ positions of its top row. The goal is to move all these items to the top row of $\mathcal{M}$. Note that the total number of items is $2^{d-d'} \times 2^{d-d'} \times 2^{d''} = 2^{2d-2d'+d''}$ and the number of processors in the top row is 2^d. In order to ensure that no processor receives more than one item (i.e. $2^{2d-2d'+d''} \le 2^d$), we insist that $d \le 2d' - d''$. The details of the data movement follow.

Algorithm `Move-to-top-row`

Step 1 In each submesh $M_{i,j}$ shift the items $i \cdot 2^{d''}$ positions to the right. In other words, the items now reside in positions $i \cdot 2^{d''} + 1$ through $(i+1) \cdot 2^{d''}$ in the top row of $M_{i,j}$.

Step 2 Move each item to the top row of $\mathcal{M}$.

Step 1 involves three broadcasts. It is easy to see that we have enough bandwidth to perform this step in $O(1)$ time. Step 2 involves one broadcast operation. Thus, we have

Lemma 5.3 *Consider a reconfigurable mesh of size $2^d \times 2^d$ partitioned into non-overlapping submeshes of size $2^{d'} \times 2^{d'}$ and assume that each submesh has at most $2^{d''}$ items in its top row. The task of moving all the items to the first row of the platform can be performed in $O(1)$ time, provided that $d + d'' \le 2d'$.*

6 An $O((\log\log n)^2)$-time algorithm

The main goal of this section is to show how all the pieces fit together to yield an $O((\log\log n)^2)$ time upper hull algorithm on a reconfigurable mesh $\mathcal{M}$ of size $2^d \times 2^d$, where $d \ge 2$.

Let $P = \{p(1), p(2), \ldots, p(n)\}$ be a set of $n = 2^{2d}$ points sorted by x-coordinate and pretiled on $\mathcal{M}$ in proximity order. Write $d_0 = \lceil \frac{2d}{3} \rceil$ and let $d_s = \max\{\lceil d_{s-1} - \frac{d}{8} \rceil, \lceil \frac{d_{s-1}}{2} \rceil\}$, for all $s \ge 1$. It is easy to see that both $d_{s-1} - d_s \le \lfloor \frac{d}{8} \rfloor$ and $d_s \ge \lceil \frac{d_{s-1}}{2} \rceil$ hold. Further, let T be the smallest integer for which $d_T = 1$. Clearly, $T \in O(\log d) = O(\log\log n)$. Partition P into $2^{2(d-d_0)}$ subsets $A(1), A(2), \ldots, A(2^{2(d-d_0)})$, such that each $A(i)$ involves 2^{2d_0} points consecutive in proximity order stored in a submesh $M(i)$ of $\mathcal{M}$ of size $2^{d_0} \times 2^{d_0}$.

For each i_0, $(1 \le i_0 \le 2^{2(d-d_0)})$, the set $A(i_0)$ is partitioned into $2^{2(d_0-d_1)}$ subsets $A(i_0, 1), A(i_0, 2), \ldots, A(i_0, 2^{2(d_0-d_1)})$, with each $A(i_0, j)$, $(1 \le j \le 2^{2(d_0-d_1)})$, consisting of 2^{2d_1} points consecutive in proximity order, stored in the submesh $M(i_0, j)$ of $M(i_0)$ of size $2^{d_1} \times 2^{d_1}$. Proceeding in this way, each set $A(i_0, i_1, \ldots, i_{s-1})$, $(1 \leq s \leq T)$, containing $2^{2d_{s-1}}$ points and stored in proximity order in the submesh $M(i_0, i_1, \ldots, i_{s-1})$ of size $2^{d_{s-1}} \times 2^{d_{s-1}}$ is partitioned into $2^{2(d_{s-1}-d_s)}$ subsets $A(i_1, i_2, \ldots, i_{s-1}, j)$, $(1 \leq j \leq 2^{2(d_{s-1}-d_s)})$, with each $A(i_1, i_2, \ldots, i_{s-1}, j)$ involving 2^{2d_s} points consecutive in proximity order, stored in a submesh $M(i_1, i_2, \ldots, i_{s-1}, j)$ of size $2^{d_s} \times 2^{d_s}$. To simplify the exposition, in the remainder of this section we shall use sets of points and the submeshes containing them interchangeably. This should create no ambiguity.

Algorithm `Compute-upper-hull`

Step 1 Recursively, determine the upper hull $P(i_0)$ of the points in each submesh $M(i_0)$, $(1 \leq i_0 \leq 2^{2(d-d_0)})$, and mark all the points in $P(i_0)$;

Step 2 In each submesh $M(i_0)$, $(1 \leq i_0 \leq 2^{2(d-d_0)})$, extract a sample $S(P(i_0))$ of $P(i_0)$ by retaining the leftmost marked point in each submesh $M(i_0, i_1)$, $(1 \leq i_1 \leq 2^{2(d_0-d_1)})$. Add to $S(P(i_0))$ the rightmost marked point in $M(i_0)$. Move the points in $S(P) = S(P(1)) \cup S(P(2)) \cup \cdots \cup S(P(2^{2(d-d_0)}))$ to the top row of $\mathcal{M}$. Using an $O(1)$-time prefix-sums algorithm [7] move $S(P)$ to the leftmost $|S(P)|$ positions of the top row of $\mathcal{M}$;

Step 3 Repeat the following substeps $2T$ times:

Step 3.1 Using one of the existing algorithms for the sparse case [2, 5], compute the upper hull $U(S(P))$ of $S(P)$ and determine the left and right contact points $p(g_i)$ and $p(h_i)$, if any, of $S(P(i))$ with respect to $U(S(P))$. If $S(P(i))$ does not have contact points, then find the pseudo contact point $p(g_i)$ of $S(P(i))$, using an $O(1)$-time minimum finding algorithm;

Step 3.2 Let $p(l(g_i))$ and $p(r(g_i))$ be, respectively, the left and right neighbors of $p(g_i)$ in $S(P(i))$. If the two points $p(l(g_i))$ and $p(g_i)$ are in the same submesh $M(i_0, i_1, \ldots, i_T)$ of size $2^1 \times 2^1$, then add to $S(P(i))$ the remaining two points $P(i_0, i_1, \ldots, i_T) - \{p(l(g_i)), p(g_i)\}$ in $M(i_0, i_1, \ldots, i_T)$. Otherwise, let $M(i_0, i_1, \ldots, i_{s-1}, i_s)$, $(s > T)$, be the smallest submesh containing both $p(g_i)$ and $p(l(g_i))$. Add to $S(P(i))$ the leftmost marked point in each of the submeshes $M(i_0, i_1, \ldots, i_{s-1}, i_s, j)$, $(1 \leq j \leq 2^{2(d_s-d_{s+1})})$. Repeat the procedure for the three pairs of points $p(g_i)$ and $p(r(g_i))$, $p(l(h_i))$ and $p(h_i)$, and $p(h_i)$ and $p(r(h_i))$. Move all newly selected sample points to the top row of $\mathcal{M}$. Using prefix sums in the obvious way, the updated $S(P)$ is stored, one item per processor, in the leftmost $|S(P)|$ positions of the top row of $\mathcal{M}$;

Step 4 Compute the upper hull $U(S(P))$ of $S(P)$ and determine the left and the right contact points $p(g_i)$ and $p(h_i)$ of $S(P(i))$, $(1 \leq i \leq 2^{2(d-d_0)})$, with respect to $U(S(P))$. Retain all the points in $P(i)$ between $p(g_i)$ and $p(h_i)$ (inclusive) as the upper hull points of P.

The correctness of the algorithm follows immediately by the induction hypothesis along with Theorem 4.2. To argue for the complexity, note that Step 2 as well as each iteration of Step 3 of the algorithm can be implemented in $O(1)$ time. In Step 2, `Move-leftmost-items` is performed in every submesh $M(i_0)$, $(1 \leq i_0 \leq 2^{2(d-d_0)})$, to move the leftmost marked point in $M(i_0, i_1)$ $(1 \leq i_1 \leq 2^{2(d_0-d_1)})$, to the top row of $M(i_0)$. Since $d_1 \geq \lceil \frac{d_0}{2} \rceil$, Lemma 5.2 guarantees that this can be done in $O(1)$ time. Since $d_0 - d_1 \leq \lfloor \frac{d}{8} \rfloor$, the top row of $M(i_0)$ contains at most $2^{2(d_0-d_1)} \leq 2^{\lfloor \frac{d}{4} \rfloor} + 1$ marked points. Further, the points thus obtained are moved to the top row of $\mathcal{M}$. Since $2d_0 - 2(d_0 - d_1) = 2\lceil \frac{2d}{3} \rceil - \lfloor \frac{d}{4} \rfloor \geq d$, Lemma 5.3 guarantees that this data movement operation can be performed in $O(1)$ time. Further, the rightmost marked point of each $M(i_0)$ can be moved to the top row of M in $O(1)$ time by executing `Move-leftmost-items`.

In the first iteration of Step 3.1, $S(P)$ contains at most $2^{2(d-d_0)} \cdot (2^{\lfloor \frac{d}{4} \rfloor}+1) \leq 2^{2\lfloor \frac{d}{3} \rfloor} \cdot (2^{\lfloor \frac{d}{4} \rfloor}+1) < 2^{\lfloor \frac{11d}{12} \rfloor} + 2^{\lfloor \frac{2d}{3} \rfloor} \leq 2^d$ points. Thus, the upper hull of $S(P)$ can be computed in $O(1)$ time. In Step 3.2, in case the points $p(l(g_i))$ and $p(g_i)$ belong to the same submesh $M(i_0, i_1, \ldots, i_T)$, the remaining two points in the submesh are selected and moved to the top row of $M(i_0)$. Clearly, this can be done in $O(1)$ time. Otherwise, `Move-leftmost-items` is executed, in parallel, in each submesh $M(i_0, i_1, \ldots, i_{s-1}, i_s)$ to move the leftmost marked item in $M(i_0, i_1, \ldots, i_{s-1}, i_s, i_{s+1})$, $(1 \leq i_{s+1} \leq 2^{2d_{s+1}})$, to the top row of $M(i_0, i_1, \ldots, i_{s-1}, i_s)$. Since $d_{s+1} \geq \lceil \frac{d_s}{2} \rceil$, Lemma 5.2 guarantees that this can be done in $O(1)$ time. Further, the leftmost marked items are moved to the top row of $M(i_0)$. The same task for the other three pairs $p(g_i)$ and $p(r(g_i))$, $p(l(g_i))$ and $p(g_i)$, and $p(h_i)$ and $p(r(h_i))$ is performed in $O(1)$ time. Next, these newly selected sample points are moved to the top row of $\mathcal{M}$ by using procedure `Move-to-top-row`. Since $d_s - d_{s+1} \leq \lfloor \frac{d}{8} \rfloor$, at most $4 \cdot 2^{2(d_s-d_{s+1})} \leq 2^{\lfloor \frac{d}{4} \rfloor + 2}$ points are moved. Since $2d_0 - (\lfloor \frac{d}{4} \rfloor + 2) = 2\lceil \frac{2d}{3} \rceil - \lfloor \frac{d}{4} \rfloor - 2 \geq d$ for sufficiently large d, Lemma 5.3 guarantees that this data movement can be performed in $O(1)$ time.

In each iteration, at most $4 \cdot 2^{\lfloor \frac{d}{4} \rfloor}$ sample points are added to each $S(P_i)$. Therefore, at most $2^{2(d-d_0)} \cdot 2^{\lfloor \frac{d}{4} \rfloor} \leq 4 \cdot 2^{\lfloor \frac{11d}{12} \rfloor}$ points are added to $S(P)$ in each execution of Step 3.2. Since Step 3.2 is executed $2T$ times, $S(P)$ has at most $2^{\lfloor \frac{11d}{12} \rfloor} + 2^{\lfloor \frac{2d}{3} \rfloor} + 2T \cdot 4 \cdot 2^{\lfloor \frac{11d}{12} \rfloor} < 2^d$ points, for sufficiently large d. Thus, the upper hull of $S(P)$ can be computed in $O(1)$ time.

Since each substep of Step 3 can be performed in $O(1)$ time, it follows that Step 3 takes, altogether,

$O(T) = O(\log\log n)$ time. Since the depth of recursion is $O(\log\log n)$, the overall running time of algorithm `Compute-upper-hull` is bounded by $O((\log\log n)^2)$. Thus we have

Lemma 6.1 *The upper hull of n points sorted by x-coordinate and stored in proximity order on a reconfigurable mesh of size $n = 2^d \times 2^d$ can be computed in $O((\log\log n)^2)$ time.*

7 A work-optimal upper hull algorithm

The goal of this section is to develop a work-optimal algorithm for computing the upper hull of n sorted points in the plane. Due to stringent page limitations, we only outline the idea that allows us to attain work-optimality.

The input to the algorithm is a set P of n points sorted by x-coordinate, pretiled on a reconfigurable mesh of size $\frac{\sqrt{n}}{\log\log n} \times \frac{\sqrt{n}}{\log\log n}$ in proximity order. In this scenario, each processor of the mesh stores $(\log\log n)^2$ consecutive input points in its local memory. We show that the upper hull of the n points can be computed in $O((\log\log n)^2)$ time by slightly modifying the algorithm `Compute-upper-hull`. To begin, using an optimal sequential algorithm[8], each processor determines in $O((\log\log n)^2)$ time the upper hull of the points it stores. The resulting upper hull is stored in the array $U[1..(\log\log n)^2]$ in its local memory. Then, as in algorithm `Compute-upper-hull`, the upper hulls stored by the processors are merged. During this merging operation, some of the points in the array $U[1..(\log\log n)^2]$ may cease to be upper hull points. However, it is clear that at any point in the merging operation the subset of points that are still candidates for being upper hull points of P form an interval, say $U[\alpha_i..\beta_i]$, with $1 \leq \alpha_i \leq \beta_i \leq (\log\log n)^2$. Notice that at all times the two local variables α_i and β_i specify the current upper hull points stored by processor $PE(i)$.

We now demonstrate the modifications to algorithm `Compute-upper-hull`. Assume that the processor $PE(i)$ stored the unique point $p(i)$ in the original `Compute-upper-hull` algorithm. Now, suppose that the point $p(i)$ was added to the sample $S(P)$. Instead, processor $PE(i)$ adds to $S(P)$ the points $U[\alpha_i]$ and $U[\beta_i]$. If in the first iteration of Step 3.2, one of the points $U[\alpha_i]$ or $U[\beta_i]$ is a contact point or pseudo contact point of $S(P)$, then $PE(i)$ adds the point $U[\frac{\alpha_i+\beta_i}{2}]$ to $S(P)$ as well. As a consequence α_i and β_i will be updated in the merging process, as appropriate. Moreover, the reader should be able to confirm that this update can be performed in $O(\log\log n)$ time. Thus we have,

Theorem 7.1 *The upper hull of n points sorted by x-coordinate and stored in proximity order, $(\log\log n)^2$ points per processor, in a reconfigurable mesh of size $\frac{\sqrt{n}}{\log\log n} \times \frac{\sqrt{n}}{\log\log n}$ can be computed in $O((\log\log n)^2)$ time.*

8 Concluding remarks

To date, the vast majority of the $O(1)$ time geometric algorithms on the reconfigurable mesh were developed for sparse input. A notable exception is the work of Bokka *et al.* [1] who showed that a number of geometric tasks can be solved in $O(1)$ time even in the dense scenario, that is, where the number of processors essentially matches the size of the input. The current effort is, in our view, a step in the same direction. We feel that in order to justify using reconfigurable meshes, it is extremely important to develop $O(1)$ time or near-$O(1)$ time algorithms for as many fundamental geometric problems as possible.

References

[1] V. Bokka, H. Gurla, S. Olariu, and J. L. Schwing, Constant-time convexity problems on reconfigurable meshes, *Journal of Parallel and Distributed Computing*, 27, (1995), 86-99.

[2] J. Jang, M. Nigam, V. K. Prasanna, and S. Sahni, Constant time algorithms for computational geometry on the reconfigurable mesh, *IEEE Transactions on Parallel and Distributed Systems*, 8, (1997), 1–12.

[3] R. Miller and Q. F. Stout, Efficient parallel convex hull algorithms, *IEEE Transactions on Computers*, C-37, (1988), 1605–1618.

[4] K. Nakano, Computing the convex hull of a sorted set of points on a reconfigurable mesh, *Parallel Algorithms and Applications*, 8, (1996), 243–250.

[5] S. Olariu, J. L. Schwing, and J. Zhang, Time-optimal convex hull algorithms on enhanced meshes, *BIT*, 33, (1993), 396–410.

[6] S. Olariu, J. L. Schwing, and J. Zhang, Fast component labeling and convex hull computation on reconfigurable meshes, *Image and Vision Computing Journal*, 11, (1993), 447–455.

[7] S. Olariu, J. L. Schwing, and J. Zhang, Fundamental algorithms on reconfigurable meshes, *Proc. Allerton Conf. on Communications, Control and Computing*, 1991, 811-820.

[8] F. P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer, 1985.

[9] L. G. Valiant, Parallelism in comparison problems, *SIAM Journal on Computing*, 4, (1975), 348–355.