

Time-Optimal Nearest-Neighbor Computations on Enhanced Meshes¹

STEPHAN OLARIU*² AND IVAN STOJMENOVIĆ†

*Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529-0162; and †Department of Computer Science, University of Ottawa, Ottawa, Ontario K1N 9B4, Canada

The All Nearest Neighbor problem (ANN, for short) is stated as follows: given a set S of points in the plane, determine for every point in S , a point that lies closest to it. The ANN problem is central to VLSI design, computer graphics, pattern recognition, and image processing, among others. In this paper we propose time-optimal algorithms to solve the ANN problem for an arbitrary set of points in the plane and also for the special case where the points are vertices of a convex polygon. Both our algorithms run on meshes with multiple broadcasting. Our first main contribution is to establish an $\Omega(\log n)$ time lower bound for the task of solving an arbitrary n -point instance of the ANN problem, even if the points are the vertices of a convex polygon. We obtain our time lower bound results for the CREW-PRAM by using a novel technique involving geometric constructions. These constructions allow us to reduce the well-known OR problem to each of the geometric problems of interest. We then port these time lower bounds to the mesh with multiple broadcasting using simulation results. Our second main contribution is to show that the time lower bound obtained is tight, by exhibiting algorithms solving the problem in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. © 1996 Academic Press, Inc.

1. INTRODUCTION

Recent advances in VLSI have made it possible to build parallel machines featuring tens of thousands of processors. Yet, practice indicates that this increase in raw computational power does not always translate into increased performance of the same order of magnitude. The reason seems to be twofold: first, not all problems are known to admit efficient parallel solutions; second, parallel computation requires interprocessor communications and simultaneous memory accesses which often act as bottlenecks in present-day parallel machines.

In this context, the mesh has emerged as the platform of choice for implementing a variety of tasks in pattern recognition, image processing, computer vision, path planning, and computational geometry. Its regular and intuitive

topology makes the mesh eminently suitable for VLSI implementation, with several models built over the years. Examples include the ILLIAC V, the MPP, and the MasPar, among others [5, 37].

One of the drawbacks of the mesh lies in its large diameter which limits the performance in contexts where the computation involves data spread over distant processing elements. In an attempt to overcome this problem, mesh-connected computers have recently been augmented by the addition of various types of bus systems [1, 12, 22, 25, 27, 28, 30]. For example, Aggarwal [1] and Bokhari [12] have considered meshes enhanced by the addition of k global buses. Yet another such system that is commercially available involves enhancing the mesh by the addition of row and column buses as illustrated in Fig. 1. In [22] an abstraction of such a system is referred to as mesh with multiple broadcasting. The mesh with multiple broadcasting has proven to be feasible to implement in VLSI, and is used in the DAP family of computers [33, 36].

Being of theoretical interest as well as commercially available, the mesh with multiple broadcasting has attracted a great deal of well-deserved attention. Applications ranging from image processing [7, 23, 33, 36], to computational morphology and pattern recognition [6, 8–10, 22, 27, 32], to optimization [15], to other fundamental problems [4, 7, 8, 14] have found efficient solutions on this architecture and some of its variants [27].

The notion of proximity or “closeness” is fundamental in image processing, computer graphics, pattern recognition, and robotics. In image processing, for example, closeness is a simple and important metric for potential similarities of objects in the image space [3]. In pattern recognition, closeness appears in clustering and in computing similarities between sets [18]. One of the recurring problems in computer graphics [34], pattern recognition [13], and image processing [3, 19] involves computing for every point in a given set S of planar points, a point that is closest to it. This problem is known as the All Nearest Neighbor problem (ANN, for short) and has been well studied in both sequential and parallel [2, 11, 16, 18, 21, 29, 31, 35, 38, 39].

In this work we address the problem of devising time-optimal algorithms for the ANN problem on meshes with multiple broadcasting. In this direction, our first main contribution is to establish an $\Omega(\log n)$ time lower bound for the task of solving an arbitrary n -point instance of the

¹ Work supported in part by NATO Grant CRG-950744, by NSF grant CCR-9407180, by ONR grant N00014-91-1-0779, and by NSERC grant OGPIN007; a preliminary version of this paper has appeared in *Proc. of IPSP'94*, Cancun, Mexico, April, 1994.

² E-mail address: olariu@cs.odu.edu.

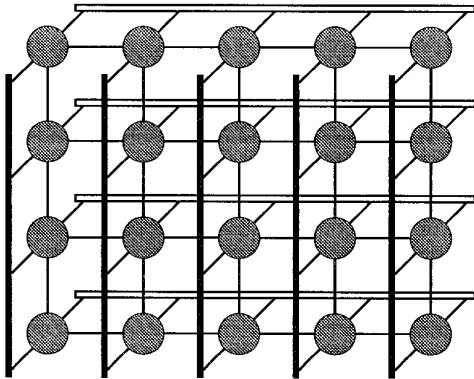


FIG. 1. A mesh with multiple broadcasting of size 4×5 .

ANN problem, even if the points are the vertices of a convex polygon. This lower bound holds for both the CREW-PRAM and for the mesh with multiple broadcasting. We obtain our time lower bound results for the CREW-PRAM by using a novel technique involving geometric constructions. These constructions allow us to reduce the well-known OR problem to each of the geometric problems of interest. We then port these time lower bounds to the mesh with multiple broadcasting using the simulation results of [26]. The only lower bound for the ANN problem known to the authors was obtained recently by Schieber and Vishkin [38] who have established an $\Omega(\log \log n)$ lower bound for the ANN problem for convex n -gons in the CRCW model of computation.

Our second main contribution is to show that the time lower bound we obtained is tight by exhibiting algorithms solving the problem in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Our first algorithm is for an arbitrary point-set, while the second solves the problem in the special case where the points are vertices of a convex polygon.

Our ANN algorithm for a general collection of points in the plane relies on a sampling technique similar to the one described in [7]. In [7] the technique is used to solve a number of visibility-related problems on enhanced meshes. In this paper, we demonstrate that the same technique can be used to solve proximity problems. The resulting algorithm for the ANN problem is significantly simpler than a recent ANN algorithm for the CREW-PRAM reported in [16]. The algorithm in [16] involves a nontrivial new application of the cascading divide and conquer strategy. By contrast, we show that the classical divide and conquer is sufficient in the context of meshes with multiple broadcasting. In essence, the algorithm in [16] proceeds in two stages: in the first stage an approximation of the nearest neighbor for every point in the input set is determined. In the second stage, this choice is refined into the exact solution.

Unlike the algorithm in [16] we do not use Stage 1 to compute an approximate solution for the ANN problem. Instead, we dynamically construct a solution to the prob-

lem, along with data structures that will be used in Stage 2 to construct certain candidate lists, that is, lists of points for which a given point is a possible closest neighbor. It seems to us that this approach is more natural and easier to understand.

The remainder of the paper is organized as follows: Section 2 discusses the computational model used throughout the paper; Section 3 reviews basic algorithms that are key ingredients in our ANN algorithms; Section 4 presents the lower bound arguments; Section 5 discusses the ANN algorithm for arbitrary points in the plane; Section 6 presents the ANN algorithm for convex polygons; finally, Section 7 summarizes our findings and proposes a number of open questions.

2. THE COMPUTATIONAL MODEL

A mesh with multiple broadcasting of size $M \times N$, hereafter referred to as a *mesh* when no confusion is possible, consists of MN identical processors positioned on a rectangular array overlaid with a bus system. In every row of the mesh the processors are connected to a horizontal bus; similarly, in every column the processors are connected to a vertical bus as illustrated in Fig. 1.

Processors $P(i, j)$ is located in row i and column j , $1 \leq i \leq M$; $1 \leq j \leq N$, with $P(1, 1)$ in the north-west corner of the mesh. Each processor is connected to its four neighbors, provided they exist. Throughout this paper we assume that the mesh with multiple broadcasting operates in SIMD mode: in each time unit, the same instruction is broadcast to all processors, which execute it and wait for the next instruction. The processors are assumed to know their coordinates within the mesh and to have a constant number of registers of size $O(\log MN)$; in unit time, each processor performs some arithmetic or boolean operation, communicates with one of its neighbors using a local link, broadcasts a value on a bus, or reads a value from a specified bus. These operations involve handling at most $O(\log MN)$ bits of information. For practical reasons, only one processor is allowed to broadcast on a given bus at any one time. By contrast, all the processors on the bus can simultaneously read the value being broadcast. In accord with other researchers [1, 4, 12, 14, 22, 30, 33], we assume that communications along buses take $O(1)$ time.

A PRAM [20] consists of synchronous processors, all having unit-time access to a shared memory. At each step, every processor performs the same instruction, with a number of processors masked out. In the CREW-PRAM, a memory location can be simultaneously accessed in reading but not in writing.

From a theoretical point of view, a mesh with multiple broadcasting can be perceived as a restricted version of the CREW-PRAM machine: the buses are nothing more than *oblivious* concurrent read-exclusive write registers with the access restricted to certain sets of processors. Indeed, in the presence of p CREW-PRAM processors, groups of \sqrt{p} of these have concurrent read access to a

register whose value is available for one time unit, after which it is lost. Given that the mesh with multiple broadcasting is, in this sense, *weaker* than the CREW-PRAM, it is very often quite a challenge to design algorithms in this model that match the performance of their CREW-PRAM counterparts. Typically, for the same running time, the mesh with multiple broadcasting uses more processors. This phenomenon will appear in our ANN algorithm.

3. BASICS

Data movement operations constitute the basic building blocks that lay the foundations of many efficient algorithms for parallel machines constructed as an interconnection network of processors. The purpose of this section is to review a number of data movement results for the mesh with multiple broadcasting that will be instrumental in the design of our algorithm.

The *prefix sums* problem has turned out to one of the basic techniques in parallel processing, being a key ingredient in many algorithms. The problem is stated as follows: given a sequence a_1, a_2, \dots, a_N of items, compute all the sums of the form $a_1, a_1 + a_2, a_1 + a_2 + a_3, \dots, a_1 + a_2 + \dots + a_N$. Recently, it has been shown that prefix sums can be computed efficiently on meshes with multiple broadcasting. More specifically, the following result has been proved in [23, 32].

PROPOSITION 3.1. *The prefix sums (also maxima or minima) of a sequence of n real numbers stored in one row of a mesh with multiple broadcasting of size $n \times n$ can be computed in $O(\log n)$ time. Furthermore, this is time-optimal in this model.*

Merging two sorted sequences is one of the fundamental operations in computer science. Recently, Olariu *et al.* [32] have proposed a constant time algorithm to merge two sorted sequences of total length n stored in one row of a mesh with multiple broadcasting of size $n \times n$. More precisely, the following result was established in [32].

PROPOSITION 3.2. *Let $S_1 = (a_1, a_2, \dots, a_r)$ and $S_2 = (b_1, b_2, \dots, b_s)$, with $r + s = n$, be sorted sequences stored in the first row of a mesh with multiple broadcasting of size $n \times n$, with $P(1, i)$ holding a_i , $1 \leq i \leq r$, and $P(1, r + i)$ holding b_i , $1 \leq i \leq s$. The two sequences can be merged into a sorted sequence in $O(1)$ time.*

Since merging is an important ingredient in our ANN algorithm, we now give the details of the merging algorithm in [32]. To begin, using vertical buses, the first row is replicated in all rows of the mesh. Next, in every row i , $1 \leq i \leq r$, processor $P(i, i)$ broadcasts a_i horizontally on the corresponding row bus, as illustrated in Fig. 2. It is easy to see that for every i , a unique processor $P(i, j)$, $r + 1 < j \leq n$, will determine that $b_{j-1} < a_i \leq b_j$. Clearly, this unique processor can now use the horizontal bus to broadcast j back to $P(i, i)$. In turn, this processor has enough information to compute the position of a_i in the sorted sequence.

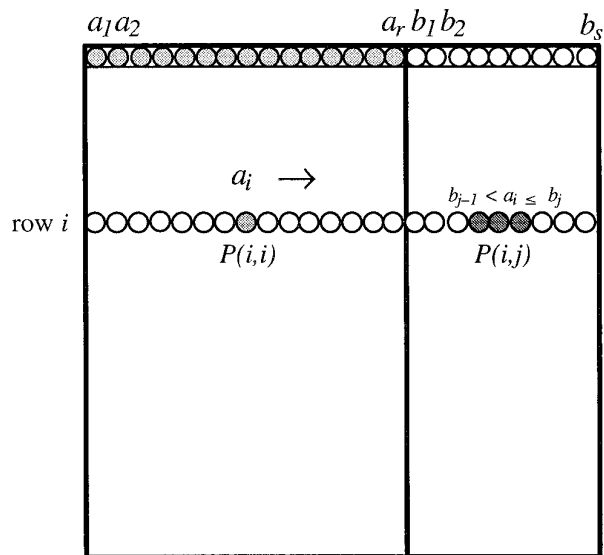


FIG. 2. Illustrating the merging operation.

In exactly the same way, the position of every b_j in the sorted sequence can be computed in $O(1)$ time. Finally, a simple data movement sends every element to its final destination in the first row of the mesh.

The simple merging algorithm that we just sketched is the main stepping stone in a time-optimal sorting algorithm developed in [32]. This algorithm implements the well-known strategy of sorting by merging. Specifically, the following result was established in [32].

PROPOSITION 3.3. *An n -element sequence of items from a totally ordered universe stored one item per processor in the first row of a mesh with multiple broadcasting of size $n \times n$ can be sorted in $O(\log n)$ time. Furthermore, this is time-optimal in this model.*

Again, to make this paper self contained, we briefly sketch the data movement operations performed in the sorting algorithm in [32]. First, the input sequence is divided into a left subsequence containing the first $n/2$ items and a second subsequence containing the remaining $n/2$ items. Further, imagine dividing the original mesh into four equal submeshes of size $(n/2) \times (n/2)$.

In preparation for sorting, the second half of the input sequence is broadcast to the first row of the south-eastern submesh. Note that because of the way we have partitioned the input, no broadcast conflicts can arise when we recursively sort the two halves of the input. Finally, the resulting sorted sequences are merged as discussed in Proposition 3.2. It is easy to see that the overall running time of this simple algorithm is $O(\log n)$. The time-optimality of the sorting algorithm follows from Propositions 4.1 and 4.2 discussed in the next section.

For later reference we now describe the details of a very simple data movement that allows to compact a list by eliminating some of its elements. For definiteness, suppose that the processors in the first row of the mesh store a

sequence a_1, a_2, \dots, a_n of items with some of the items marked. Assume further that every marked item knows its rank among the marked items. We wish to obtain a sublist consisting of the marked elements stored in order in the leftmost positions of the first row of the mesh. This task can be performed as follows. Suppose that a_i is the k th marked element in the sequence; now processor $P(1, i)$ will broadcast a_i vertically to processor $P(k, i)$ which, in turn, will broadcast a_i horizontally to $P(k, k)$. Finally, $P(k, k)$ will broadcast a_i vertically to $P(1, k)$, as desired. Consequently, we have the following result.

LEMMA 3.4. *Consider a sequence a_1, a_2, \dots, a_n of items stored in the first row of a mesh with multiple broadcasting of size $n \times n$, one item per processor, with some of the items marked. If every marked item knows its rank among the marked items, then a sublist consisting of the marked elements stored in order in the leftmost positions of the first row of the mesh can be obtained in $O(1)$ time.*

Suppose that n items a_1, a_2, \dots, a_n are stored, one per processor, in the first row of a mesh with multiple broadcasting of size $n \times n$. Suppose further that these items are partitioned into k disjoint groups of consecutive items, such that for every $j, 1 \leq j \leq k$, group j contains the items $a_{i_{j-1}}, a_{i_{j-1}+1}, a_{i_{j-1}+2}, \dots, a_{i_j}$. To handle boundary conditions we write $a_1 = a_{i_0}$ and $a_n = a_{i_k}$. It is assumed that every item in the input sequence knows the group to which it belongs. The goal is to determine the minimum item in every group.

We begin by informing every item in the sequence of the index of the first item in its own group. This can be done as follows. Every processor holding the first item of some group writes its index in a local register; all others write a 0. After performing the prefix maximum over the resulting sequence, every item has the desired information. Specifically, all the items in the j th group know that the first item in that group is in position a_{i_j} . Furthermore, by Proposition 3.1, this takes at most $O(\log n)$ time.

Next, using vertical buses all the items in the j th group, $1 \leq j \leq k$, are broadcast vertical to processors $P(i_{j-1}, i_{j-1}), \dots, P(i_{j-1}, i_j - 1)$. Clearly, this simple operation takes constant time. Notice that as a result of the previous data movement, the items in group $j, 1 \leq j \leq k$, are in the first row of a submesh of size $(i_j - i_{j-1}) \times (i_j - i_{j-1})$ with its north-west corner at $P(i_{j-1}, i_{j-1})$. In each such submesh, the minimum is computed in parallel. By Proposition 3.1, this operation runs in at most $O(\log n)$ time. Note also that a straightforward application of Propositions 4.1 and 4.2, discussed in the next section, guarantees that our solution is time-optimal. Consequently, we have proved the following result.

LEMMA 3.5. *The (prefix) minimum in every group of consecutive items stored in the first row of a mesh with multiple broadcasting of size $n \times n$ can be computed in $O(\log n)$ time. Furthermore, this is time-optimal in this model.*

4. THE LOWER BOUND

The goal of this section is to derive a time lower bound for the ANN problem. This lower bound holds for both the CREW-PRAM and for the mesh with multiple broadcasting. Our optimality arguments will be stated first in the CREW-PRAM. This approach is motivated by a recent result of Lin *et al.* [26] that allows us to extend many lower bound results from the CREW-PRAM to meshes with multiple broadcasting.

In preparation for the main result of this section we now state a fundamental result of Cook *et al.* [17] that will lay the foundation of our lower bound argument for the ANN problem on meshes with multiple broadcasting.

PROPOSITION 4.1. *The time lower bound for computing the OR of n bits on the CREW-PRAM is $\Omega(\log n)$ no matter how many processors and memory cells are available.*

In addition, we shall rely on the following result of Lin *et al.* [26].

PROPOSITION 4.2. *Any computation that takes $O(t(n))$ computational steps on an n -processor mesh with multiple broadcasting can be performed in $O(t(n))$ computational steps on an n -processor CREW-PRAM with $O(n)$ extra memory.*

It is important to note that Proposition 4.2 guarantees that if $T_M(n)$ is the execution time of an algorithm for solving a given problem on an n -processor mesh with multiple broadcasting, then there exists a CREW-PRAM algorithm to solve the same problem in $T_P(n) = T_M(n)$ time using n processors and $O(n)$ extra memory. In other words, “too fast” an algorithm on the mesh with multiple broadcasting implies “too fast” an algorithm for the CREW-PRAM. This observation is exploited in [26] to transfer known computational lower bounds for the PRAM to the mesh with multiple broadcasting.

Our first goal in this direction is to establish an $\Omega(\log n)$ lower bound for the CONVEX ANN problem in the CREW-PRAM model of computation. From there, the conclusion follows by Propositions 4.1 and 4.2. The CONVEX ANN problem is defined as follows:

CONVEX ANN: Given an n -vertex convex polygon, for each vertex find its nearest neighbor.

Our plan is to reduce OR to CONVEX ANN. For this purpose, let b_1, b_2, \dots, b_n be an arbitrary input to OR. To construct an instance of the CONVEX ANN problem, let C be the unit circle centered at ω and let u_1, u_2, \dots, u_n be equally spaced points on the first quadrant of C such that, for every $i, 1 \leq i \leq n$, the coordinates of u_i in polar coordinates are given by

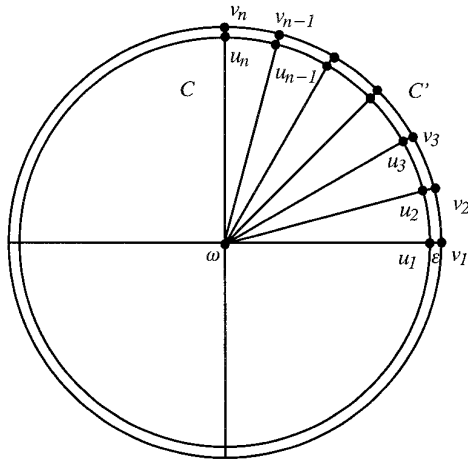
$$u_i = \left(1, \frac{(i-1)\pi}{2(n-1)}\right). \quad (1)$$

Let δ be a number satisfying $\cos \pi/2(n - 1) = 1/(1 + \delta)$ and let ε be a positive number less than δ . Further, let C' be the circle of radius $1 + \varepsilon$ centered at ω , as illustrated in Fig. 3(a). For every $i, 1 \leq i \leq n$, let v_i be the intersection between the first quadrant of C' with the extension of the ray $\overrightarrow{\omega u_i}$. Finally, let P be the polygon with vertices p_0, p_1, \dots, p_n such that $p_0 = \omega$, and for every $i, 1 \leq i \leq n$,

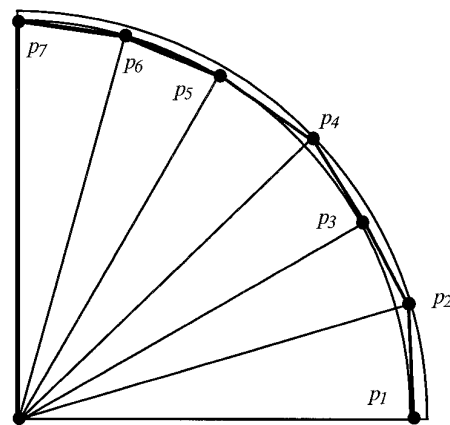
$$p_i = \begin{cases} u_i & \text{if } b_i = 1 \\ v_i & \text{if } b_i = 0. \end{cases} \quad (2)$$

The resulting polygon for the input sequence 1, 0, 1, 0, 1, 1, 1 is featured in Fig. 3(b).

At this point, the reader may wonder whether the polygon P specified in (2) above is always convex, regardless of the input sequence b_1, b_2, \dots, b_n . Our next result shows that this is the case.



(a)



(b)

FIG. 3. Illustrating the construction in the proof of the lower bound.

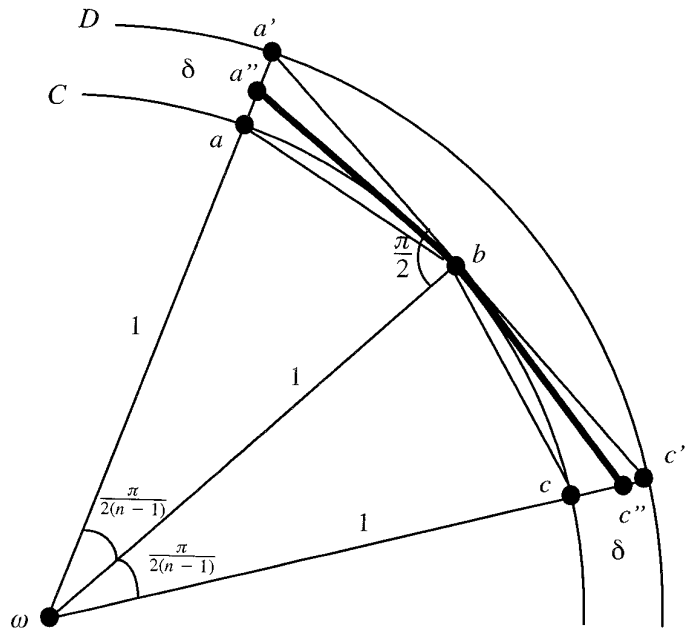


FIG. 4. Illustrating the proof of Lemma 4.3.

LEMMA 4.3. *The polygon P specified by (2) is convex.*

Proof. Consider the unit circle C centered at ω and refer to Fig. 4. Take three points a, b, c on C such that the angles $\angle a\omega b$ and $\angle b\omega c$ equal $\pi/2(n - 1)$.

Draw a line segment through b perpendicular to $\overline{\omega b}$ and let it intersect the rays $\overline{\omega a}$ and $\overline{\omega c}$ at a' and c' respectively. Further, let δ denote the common length of the segments $\overline{aa'}$ and $\overline{cc'}$ and let D be the circle of radius $1 + \delta$ centered at ω . Moreover, the right triangle $\omega b a'$ yields

$$\cos \frac{\pi}{2(n - 1)} = \frac{1}{1 + \delta}. \quad (3)$$

To understand the motivation for the construction above, consider arbitrary points a'' and c'' on the half-open line segments $[a, a')$ and $[c, c')$. We claim that

$$\angle a''bc'' \text{ is convex.} \quad (4)$$

To justify (4), observe that, by construction, the angle $\angle a''bc''$ is contained within the angle $\angle a'bc'$ whose measure is π . Consequently, the angle $a''bc''$ (as seen from ω) must be convex, as claimed.

It is now easy to extend this idea: partition the first quadrant of C into n equal arcs by equally spaced points u_1, u_2, \dots, u_n satisfying (1). Next, pick an arbitrary ε in the range $0 < \varepsilon < \delta$ and place the points v_1, v_2, \dots, v_n at the intersection of the rays $\overrightarrow{\omega u_i}, 1 \leq i \leq n$, with C' . Now, let the vertices of P be placed according to (2). By (4), all the internal angles of P are convex, and the proof of the lemma is complete. ■

Thus, by virtue of Lemma 4.3 the polygon P is always convex, regardless of the sequence b_1, b_2, \dots, b_n . Now, consider any algorithm that correctly solves the CONVEX ANN problem with P as input. Clearly, the answer to the OR problem is 1 if and only if the distance between p_0 and its corresponding nearest neighbor in P is 1.

Since the reduction can be performed in parallel in $O(1)$ time, the conclusion follows by Proposition 4.1. To summarize our discussion we state the following result.

LEMMA 4.4. *CONVEX ANN has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are available.*

It is now clear that Lemma 4.4 and Proposition 4.2 combined imply the following result.

COROLLARY 4.5. *CONVEX ANN has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$.*

It is a simple observation that the result of Corollary 4.5 applies to the ANN problem as well. Thus, we have the following important result.

COROLLARY 4.6. *ANN has a time lower bound of $\Omega(\log n)$ on the a mesh with multiple broadcasting of size $n \times n$.*

5. THE ANN ALGORITHM

The purpose of this section is to show that the lower bound of Corollary 4.6 is tight by devising an algorithm that solves an arbitrary instance of size n of the ANN problem in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$.

Consider an arbitrary set S of points in the plane, with every point specified by its cartesian coordinates. The set is assumed stored, one point per processor, in the first row of a mesh with multiple broadcasting of size $n \times n$. To avoid tedious details we assume, without loss of generality, that the points in S are in *general* position, with no two having the same x or y coordinates, and that n is a power of 2. Should this not be the case, an easy modification detailed in [16] allows to address boundary cases within the same complexity. We borrow most of our definitions, notation, and terminology from [16]. For the sake of completeness we now review the main definitions that will be used in the remainder of the algorithm.

Given an n -leaf complete binary tree T , and an arbitrary node v of T , we let $\text{desc}(v)$ stand for the set of leaf-descendants of v ; we let $Y(v)$ denote the sequence of points in $\text{desc}(v)$ sorted by increasing y -coordinate. We assume that the nodes in T are numbered level by level in left-to-right order. As illustrated in Fig. 5, with every point q in $Y(v)$ we associate four regions of the plane $NW(q)$, $NE(q)$, $SW(q)$, $SE(q)$ defined as follows:

- $NW(q)$ consists of the points p in the plane for which $x(p) < x(q)$ and $y(p) > y(q)$;

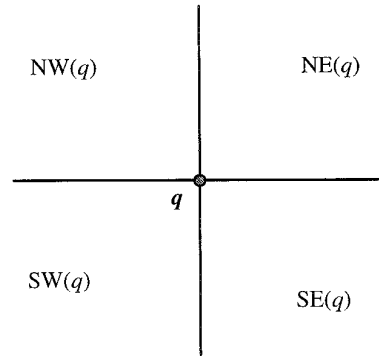


FIG. 5. Illustrating the regions $NW(q)$, $NE(q)$, $SW(q)$, $SE(q)$.

- $NE(q)$ consists of the points p in the plane for which $x(p) > x(q)$ and $y(p) > y(q)$;
- $SW(q)$ consists of the points p in the plane for which $x(p) < x(q)$ and $y(p) < y(q)$;
- $SE(q)$ consists of the points p in the plane for which $x(p) > x(q)$ and $y(p) < y(q)$;

We proceed in two stages. Our stages are different from those in [16] since we compute different items in each of our stages.

Stage 1. Begin by sorting the points in S by increasing x -coordinate and let $S = (q_1, q_2, \dots, q_n)$ be the resulting sequence. Imagine planting a complete binary tree T on S with the leaves corresponding, in left-to-right order, to the points in S as illustrated in Fig. 6. Stage 1 of the algorithm proceeds essentially by divide and conquer, with the computation being guided by the complete binary tree T .

Since the points in S are assumed to have distinct x -coordinates, we separate the leaves of T by placing vertical dividing lines between the corresponding points in S . With every node v in T , we associate the slab $\Pi(v)$ which is the region delimited by the left and right vertical lines that separate the points in $\text{desc}(v)$ from the remaining points in S . For definiteness, we let $\text{left}(v)$ and $\text{right}(v)$ stand for the left and right boundaries of $\Pi(v)$. As an illustration, referring to Fig. 6, the slab $\Pi(e)$ is the planar region delimited in the obvious way by the vertical dividing lines $\text{left}(e)$ and $\text{right}(e)$.

To make the exposition more transparent we shall consider a generic node v in T with left and right children u and w , respectively. Most of the remaining part of Stage 1 involves describing the tasks performed in the transition from u and w to v . First, $Y(v)$ is obtained by simply merging $Y(u)$ and $Y(w)$. By Proposition 3.2, this task is carried out in $O(1)$ time.

Next, for every point q in $Y(u)$, we let $\text{pred}(q, Y(u))$ and $\text{succ}(q, Y(u))$ stand for the points that precede and succeed q in $Y(u)$, respectively. In addition, we let $\text{pred}(q, Y(w))$ and $\text{succ}(q, Y(w))$ stand for the points that precede and succeed q in $Y(w)$. Of course, a similar definition holds for a point q in $Y(w)$. For an illustration, the reader is referred to Fig. 7. These points are said to be “encoun-

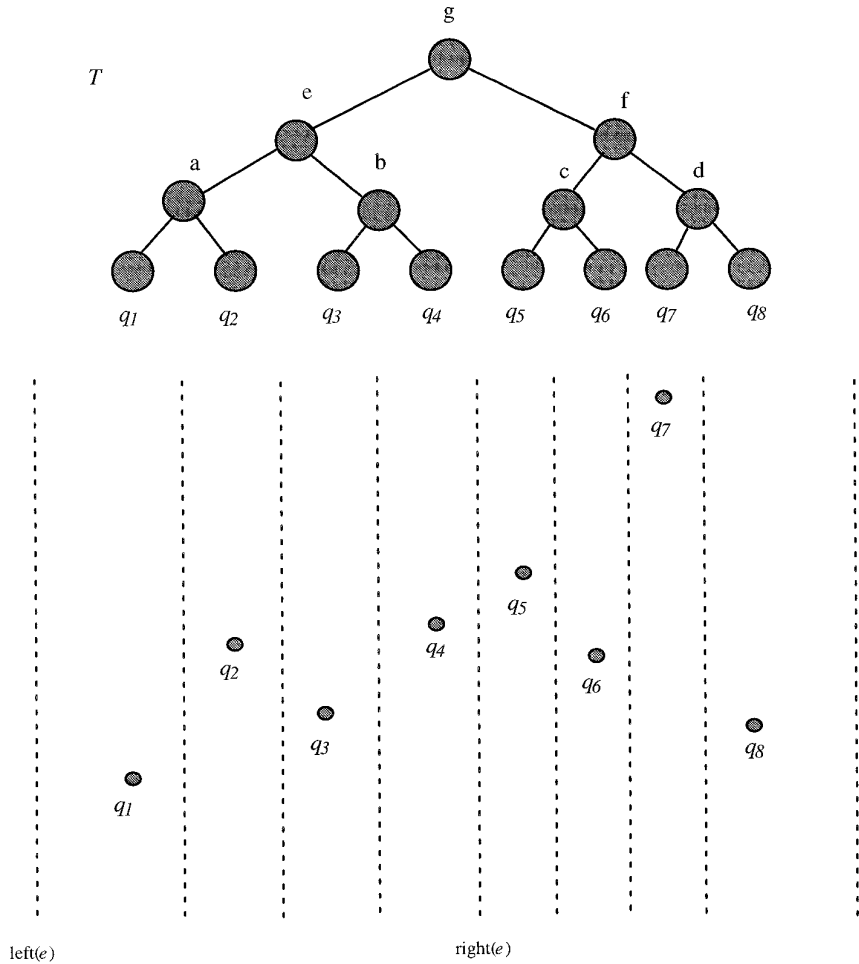


FIG. 6. Illustrating the tree T associated with the set S .

tered” by q at v . To keep track of the closest point encountered in Stage 1 of the algorithm, a label $\beta(q)$ is maintained for every point q in S . For each q_i in S , $\beta(q_i)$ is initialized to the point in $\{q_{i-1}, q_{i+1}\}$ that is closest to q_i (boundary cases are handled in the obvious way). Throughout Stage 1 of the algorithm, $\beta(q)$ contains the identity of the closest point to q encountered thus far.

Once q knows the four points defined above, it will

update $\beta(q)$ to be the point closest to q among $\{\beta(q), \text{pred}(q, Y(u)), \text{succ}(q, Y(u)), \text{pred}(q, Y(w)), \text{succ}(q, Y(w))\}$. It is important to note that this update operation can be carried out in constant time. To see this, note that the task of determining the four points encountered by q at v is easy: the first two are immediately available from $Y(u)$, while the last two are determined in the process of merging $Y(u)$ and $Y(w)$. Note that by Proposition 3.2 the merge operation takes constant time and so updating $\beta(q)$ takes constant time.

Corresponding to $\beta(q)$, we define the region $B(q)$ of the plane consisting of all the points whose distance to q is no larger than $d(q, \beta(q))$. Just as $\beta(q)$, the value of $B(q)$ dynamically changes during the execution of Stage 1.

With every node v of T we associate two subsets of $Y(v)$ defined as follows:

- $L(v)$: the set of points q in $Y(v)$ for which $B(q) \cap \text{left}(v) \neq \emptyset$, and
- $R(v)$: the set of points q in $Y(v)$ for which $B(q) \cap \text{right}(v) \neq \emptyset$.

Intuitively, $L(v)$ and $R(v)$ contain points in $Y(v)$ that may find their nearest neighbor outside the slab $\Pi(v)$. For an

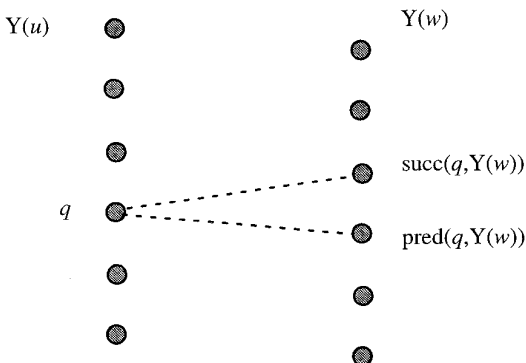


FIG. 7. Illustrating $\text{pred}(q, Y(u))$ and $\text{succ}(q, Y(u))$.

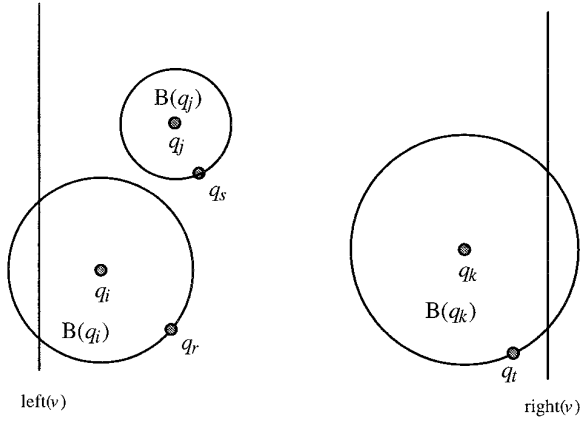


FIG. 8. Illustrating the sets $L(v)$ and $R(v)$.

illustration, in Fig. 8, the point q_i belongs to $L(v)$, the point q_k belongs to $R(v)$, while the point q_j belongs to neither of these sets.

Further, with every point q in $Y(v)$ we associate the following labels (refer to Fig. 9 for an illustration):

- $nw(q)$ consists of the point with minimum y coordinate in $Y(v) \cap NW(q)$;
- $ne(q)$ consists of the point with minimum y coordinate in $Y(v) \cap NE(q)$;
- $sw(q)$ consists of the point with maximum y coordinate in $Y(v) \cap SW(q)$;
- $se(q)$ consists of the point with maximum y coordinate in $Y(v) \cap SE(q)$.

The motivation for defining these labels is provided by the following technical result which is somewhat stronger than Lemma 3.2 in [16]. Also, our proof is different.

LEMMA 5.1. *Let v be a node in T with left and right children u and w , respectively, and let q be a point in $Y(v)$. If q belongs to $Y(u)$, then the only points in $Y(w)$ for which q is a possible nearest neighbor are $pred(q, L(w))$, $sw(pred(q, L(w)))$, $succ(q, L(w))$, and $nw(succ(q, L(w)))$. Similarly, if q belongs to $Y(w)$, then the only points in $Y(u)$ for which q is a possible nearest neighbor are $pred(q, R(u))$, $se(pred(q, R(u)))$, $succ(q, R(u))$, and $ne(succ(q, R(u)))$.*

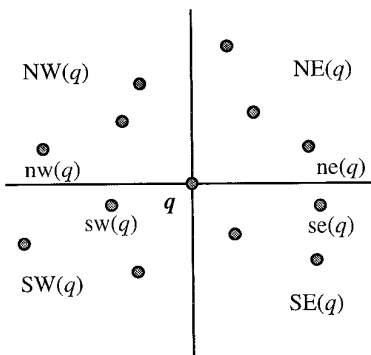


FIG. 9. Illustrating $nw(q)$, $ne(q)$, $sw(q)$, and $se(q)$.

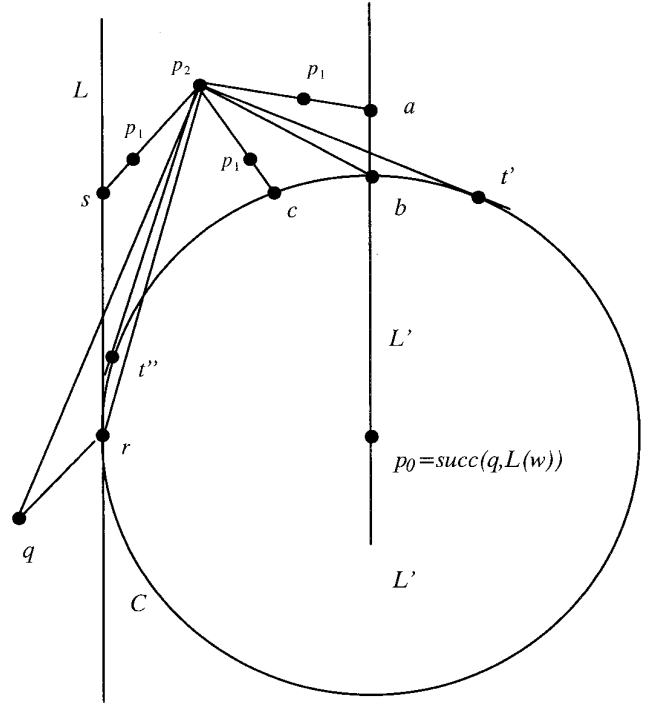


FIG. 10. Illustrating the proof of Lemma 5.1.

Proof. Let q be a point in $Y(u)$. We shall prove that the only points in $Y(w)$, having y coordinates larger than $y(q)$, for which q is a nearest neighbor are $succ(q, L(w))$, and $nw(succ(q, L(w)))$.

For this purpose, referring to Fig. 10, let L be the vertical line separating $Y(u)$ and $Y(w)$ and let p_0 and p_1 stand for $succ(q, L(w))$ and $nw(succ(q, L(w)))$, respectively. Let C be the circle centered at p_0 with radius equal to the distance from p_0 to the line L . Write $\{r\} = C \cap L$. It is easy to see that p_1 is outside C , for otherwise p_0 would not be in $L(w)$.

If the statement is false, we find a point p_2 in $Y(w)$, with $y(p_2) > y(p_0)$, closer to q than to either of p_0 and p_1 . We propose to show that this will lead to a contradiction. For this purpose, let L' be the vertical line through p_0 . Let t' and t'' be the points where the two tangents from p_2 touch C . Clearly, every point to the right of L' is closer to p_0 than to q . Consider a ray R originating at p_2 and going through p_1 . Consider the first intersection point of R with either L , L' , or C . We distinguish between the following cases (refer to Fig. 10).

Case 1. R intersects L first.

Write $\{s\} = R \cap L$. Note that in this case we have $d(p_2, p_1) < d(p_2, s) < d(p_2, r) < d(p_2, q)$, and so Case 1 cannot occur.

Case 2. R intersects L' first.

Write $\{a\} = R \cap L'$ and $\{b\} = C \cap L'$. Note that in this case we have $d(p_2, p_1) < d(p_2, a) < d(p_2, b) < d(p_2, t') = d(p_2, t'') < d(p_2, r) < d(p_2, q)$, confirming that Case 2 cannot occur.

Case 3. R intersects C first.

Write $\{c\} = R \cap C$. Note that in this case we have $d(p_2, p_1) < d(p_2, c) < d(p_2, t') < d(p_2, q)$, and so Case 3 cannot occur.

Therefore, the existence of p_2 leads to a contradiction, completing the proof of Lemma 5.1. ■

As pointed out in [16], the task of updating the labels $nw(q)$, $ne(q)$, $sw(q)$, and $se(q)$ in the transition from u and w to v is easy. For example, to update the label $sw(q)$ one proceeds as follows: if q belongs to $Y(u)$, then no update is necessary; if q belongs to $Y(w)$, then $sw(q)$ is updated to the point with largest y coordinate between the old value of $sw(q)$ and $\text{pred}(q, Y(u))$. Clearly, the corresponding computation takes constant time once $\text{pred}(q, Y(w))$ is known.

Updating the sets $L(u)$ and $L(w)$ into $L(v)$ is a bit trickier and the corresponding task will span both Stage 1 and Stage 2 of our algorithm. We now detail the operations supportive of this update that are performed in Stage 1. First, every point q in $Y(v)$ that belongs to $L(u)$ or $L(w)$ checks to see whether it belongs to $L(v)$. Observe that the definition of the slabs corresponding to nodes of T guarantees that once a point ceases to belong to $L(v)$ or $R(v)$, this property will hold for all the nodes from v to the root of T . In preparation for the update, every point q will remember the identity of the node α in T where, for the first time, it does not belong to $L(\alpha)$. In this context, we shall say that q “dies” at α . Every point q that belongs to $L(\alpha)$ is said to “survive” at α . (A similar operation takes place for $R(\alpha)$.) To anticipate, the dying time of every point in S will be used in the next stage to correctly compute the sets $L(v)$ and $R(v)$ for all nodes in the tree.

Stage 2. The main goal of this stage is to build for every point q in S a candidate list $C(q)$ of points for which q is a possible nearest neighbor. As pointed out in Lemma 5.1, to maintain the list $C(q)$, we need to have the lists $L(v)$ and $R(v)$ available at every node v in T . In preparation for this, we begin by sorting the points in S by their dying time (i.e., by the index of the node in T at which they died), in increasing y coordinate. Clearly, Proposition 3.3 guarantees that the sort operation runs in $O(\log n)$ time.

Once this information is available, we traverse T again from the leaves to the root, computing for every node v in T the lists $L(v)$ and $R(v)$, as well as maintaining the candidate list $C(q)$ for every point in S , as we go. For definiteness, we show how the list $L(v)$ is obtained from $L(u)$ and $L(w)$, when a node v with left and right children u and w is processed. Begin by merging $L(u)$ and $L(w)$ into a list $L'(v)$. From $L'(v)$ we need to delete those points that have died at v . For this purpose, we merge $L'(v)$ with the list $D(v)$ of points that have died at v : note that this list is readily available by virtue of the sorting step described above. Again, by Proposition 3.2, the merging operation runs in constant time.

It is important to note that in the process of merging

$L'(v)$ and $D(v)$, every point that survives at v finds its predecessor and successor (i.e., its own rank) in $D(v)$. This information will allow the surviving points to compute their ranks in $L(v)$. Now Lemma 3.4 guarantees that we can obtain a compact version of $L(v)$ in constant time.

We use the lists $L(v)$ and $R(v)$ to compute the candidate list $C(q)$ for every point q in S . It is an easy observation that the list $C(q)$ need never contain more than six points. Now the construction of $C(q)$ takes place as follows. At every node v in T , we can update the candidate list $C(q)$ of a point q in $Y(v)$, by letting one processor solve an instance of at most 10 points of ANN. Specifically, by virtue of Lemma 5.1, these points are those contained in the old $C(q)$ along with at most four new points. Clearly, this update operation takes $O(1)$ time, since the number of points involved is a small constant. We have proved the following.

LEMMA 5.2. *Given a set S of points in the plane stored one per processor in the first row of a mesh with multiple broadcasting of size $n \times n$, for every point q in S , the candidate list $C(q)$ can be computed in $O(\log n)$ time.*

We complete our algorithm by a post-processing operation. The purpose of this operation is to remove from each $C(q)$ those points which are not closest neighbors of q . The task specific to the postprocessing is carried out as follows. Assume that for every q , $C(q)$ contains ordered pairs of the form (q', q) . Since there are at most $6n$ ordered pairs in all the candidate lists combined, we can sort these ordered pairs by their first component. By Proposition 3.3, this takes $O(\log n)$ time. The resulting sequence is stored by the processors in the first row of the mesh, with each processor storing (at most) six consecutive ordered pairs.

For simplicity of exposition, we assume that every processor in the first row stores ordered pairs having the same first component. As a first step, each processor computes $d(q', q)$ for all the pairs it stores and retains the pair that achieves the minimum. Note that after this step, we are left with an instance of the problem discussed in Lemma 3.5. Therefore, the post-processing takes $O(\log n)$ time. To summarize our findings we state the following result.

THEOREM 5.3. *Given a set S of points in the plane stored one per processor in the first row of a mesh with multiple broadcasting of size $n \times n$, for every point in S , its closest neighbor in S can be computed in $O(\log n)$ time. Furthermore, this is time-optimal on this architecture.*

6. SOLVING THE CONVEX ANN PROBLEM

Clearly, the algorithm detailed in the previous section applies to any instance of the CONVEX ANN problem as well. Furthermore, by Lemma 4.3, this will result in a time-optimal algorithm. However, the CONVEX ANN problem is much more constrained than the general ANN problem and, intuitively, one would expect that a correspondingly simpler algorithm can be devised. The purpose

of the remainder of this work is to show that such an algorithm actually exists. Our algorithm is a nontrivial parallelization of a variant of the algorithm of Lee and Preparata [24].

To make our presentation easier to follow we now introduce some terminology. An n -vertex polygon P is specified by enumerating its vertices in counterclockwise order as p_0, p_1, \dots, p_{n-1} in such a way that with subscript arithmetic modulo n , the edges of P are $p_i p_{i+1}$ ($0 \leq i \leq n-1$). A convex polygon P has the *semicircle* property if the following two conditions are satisfied:

- (i) the diameter $D(P)$ of P is achieved by an edge;
- (ii) all vertices of P lie inside a circle with diameter $D(P)$.

The following technical result [24] is one of the fundamental ingredients in our algorithm.

PROPOSITION 6.1. *Given a convex polygon $P = p_0, p_1, \dots, p_{n-1}$ with the semicircle property, for any vertex p_i , its nearest neighbor is either p_{i-1} or p_{i+1} .*

Throughout the remainder of this work we assume an n -vertex convex polygon $P = p_0, p_1, \dots, p_{n-1}$ stored in the first row of a mesh with multiple broadcasting of size $n \times n$, with $P(1, i)$ storing p_{i-1} for all $i, 1 \leq i \leq n$. We shall say that P is well-stored. We note that Proposition 6.1 has the following consequence.

OBSERVATION 6.2. *Let P be an n -vertex convex polygon well-stored in a mesh with multiple broadcasting of size $n \times n$. In case P satisfies the semicircle property, the nearest neighbor of every vertex in P can be determined in $O(1)$ time.*

Proof. All we need do is mandate every processor to find the smallest distance to its neighbors. This is trivially achieved in $O(1)$ time. ■

The second fundamental ingredient is a parallelization of Lemma 3 in [24]. Let $P = p_0, p_1, \dots, p_{n-1}$ be a convex polygon satisfying property (i) stated above. We shall assume, without loss of generality, that the diameter of P is achieved by the edge $p_0 p_{n-1}$ and that this edge is collinear with the x -axis, as illustrated in Fig. 11.

Let p_j be the vertex with the largest perpendicular distance to $p_0 p_{n-1}$. It is now easy to see that both polygons $P_1 = p_0, p_1, \dots, p_j$ and $P_2 = p_j, p_{j+1}, \dots, p_{n-1}$ are convex and satisfy the semicircle property. Further, let L be the vertical line through p_j , and let p_k be a generic point in P_1 . Let $\delta(p_k)$ be the distance separating p_k from its nearest neighbor in P_1 and consider the circle C_k centered at p_k and with radius $\delta(p_k)$. Clearly, if C_k does not intersect L then p_k has already found its nearest neighbor in P . Otherwise, let u_k and l_k be the intersection points between L and C_k , with u_k above l_k . It is easy to see [24] that the only candidates that need to be considered to correctly update the nearest neighbor information of p_k are those points in P_2 whose projection onto L lie between u_k and l_k .

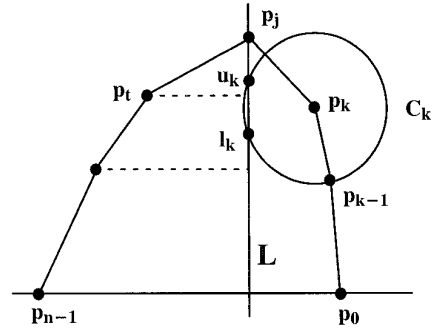


FIG. 11. Updating nearest neighbor information.

What makes the problem easy to solve on meshes with multiple broadcasting is the observation that the projection of a point p_i in P_2 can only belong to at most four circles [24] corresponding to points in P_1 . Furthermore, convexity guarantees that the points corresponding to these circles occur consecutively in P_1 . Let P be an n -vertex convex polygon satisfying property (i). We assume that P is well stored in the first row of a mesh with multiple broadcasting of size $n \times n$. Note that, by Observation 6.2, we can solve the all nearest neighbor problem for P_1 and P_2 in $O(1)$ time. We now propose to show that the all nearest neighbor problem for P can be solved in $O(1)$ time.

To begin, using the vertical buses, we replicate the information in the first row throughout the mesh. Consider a generic row i of the mesh with processor $P(i, i)$ storing a point p_{i-1} in P_1 . The following tasks are being performed in row i . Naturally, similar tasks are carried out, in parallel, in all other rows of the mesh. Processor $P(i, i)$ computes $\delta(p_{i-1})$ (i.e., the distance from p_{i-1} to its nearest neighbor in P_1), as well as the intersection points of C_{i-1} with L . In case the intersection points u_{i-1} and l_{i-1} exist, processor $P(i, i)$ broadcasts on the horizontal bus a packet consisting of $\delta(p_{i-1})$, u_{i-1} , and l_{i-1} . Every processor in row i storing a point in P_2 detects whether the projection onto L of the point it stores lies between u_{i-1} and l_{i-1} . If so, the processor marks itself.

The convexity of P guarantees that in every row of the mesh, the marked processors occur consecutively. Therefore, for every i , ($1 \leq i \leq n$), detecting the leftmost and the rightmost marked processor, along with the number of marked processors in row i can be done in $O(1)$ time. In one broadcast operation this information is sent to $P(i, 1)$. Once this information has been gathered in the first column of the mesh, Proposition 3.1 guarantees that it takes $O(\log n)$ time to compute the prefix sums of these items. The corresponding value of the prefix sum is then broadcast throughout every row. As a consequence, every marked processor knows its rank among the marked processors in the mesh.

By a previous observation, there are at most four marked processors and they occur in consecutive rows. Therefore, for every j , $1 \leq j \leq n$, if $P(i, j)$ is the first marked processor

in column j in top-down order, then at most $P(i + 1, j)$, $P(i + 2, j)$, and $P(i + 3, j)$ can be marked. We shall refer to the top-most marked processor in every column as belonging to the first generation, the second marked processor in every column is said to be of the second generation, and so on. We bring the marked processors to the first row of the mesh, at most four items per processor: note that since the total number of marked processors does not exceed $4n$, this is possible. The details of this data movement follow. In a first step a generic processor $P(i, j)$ of the first generation holding an item of rank c , sends the item vertically to the diagonal processors $P(j, j)$. In turn, $P(j, j)$ broadcasts the item to $P(j, c)$ which broadcasts the item to $P(1, c)$. The same data movement is then repeated three more times to move the items stored by second, third, and fourth generation processors to the first row. Next, proceeding as in the postprocessing stage of our ANN algorithm for arbitrary points we can update the nearest neighbor information for every point in P . To summarize our findings, we state the following result.

LEMMA 6.3. *The CONVEX ANN problem for an n -vertex polygon satisfying property (i) can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$.*

Finally, let $P = p_0, p_1, \dots, p_{n-1}$ be an n -vertex convex polygon well stored in the first row of a mesh with multiple broadcasting of size $n \times n$ with $P(1, i)$. We now show that the CONVEX ANN problem for P can be solved in $O(\log n)$ time. To begin, using the algorithm in [10] compute the diameter $D(P)$ of P . Assume, without loss of generality that the diameter is achieved by the pair p_u and p_v . Clearly, the chord $p_u p_v$ partitions the polygon into two polygons $P_1 = p_u, p_{u+1}, \dots, p_v$ and $P_2 = p_v, p_{v+1}, \dots, p_u$, both satisfying property (i). By Lemma 6.3, we can solve the CONVEX ANN problem for both P_1 and P_2 separately in $O(\log n)$ time. Furthermore, applying one more time the technique of Lemma 6.3 allows one to correctly update the nearest neighbor information for every point in P . Therefore, we have the following result.

THEOREM 6.4. *The CONVEX ANN problem for an n -vertex polygon can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal on this architecture.*

7. CONCLUSIONS AND OPEN PROBLEMS

The mesh-connected computer architecture has emerged as one of the most natural choices for solving a large number of computational tasks in image processing, pattern recognition, robotics, and computer vision. Its regular structure and simple interconnection topology makes the mesh particularly well suited for VLSI implementation. However, due to its large communication diameter, the mesh tends to be slow when it comes to handling data transfer operations over long distances. In an attempt to overcome this problem, mesh-connected computers have

been augmented by the addition of various types of bus systems. Among these, the mesh with multiple broadcasting is of a particular interest being commercially available.

In this paper we have presented a time-optimal algorithm to solve the All-Nearest Neighbor (ANN) problem for a set of n planar points on a mesh with multiple broadcasting of size $n \times n$ as well as for n -vertex convex polygons. Our ANN algorithm for a general collection of points in the plane relies on a sampling technique similar to the one described in [7]. In [7] the technique is used to solve a number of visibility-related problems on enhanced meshes. In this paper, we demonstrate that the same technique can be used to solve proximity problems. The resulting algorithm for the ANN problem is significantly simpler than a recent ANN algorithm for the CREW-PRAM reported in [16] since no cascading divide and conquer is used.

Yet another recurring proximity problem, given a set S of points in the plane, asks for computing a pair of points in S that are closest together. The problem is known as the Closest Pair problem (CP, for short). The CP problem has been extensively studied in both sequential and parallel [11, 13, 18, 39]. Clearly, our ANN algorithm can solve the CP problem since we can compute the smallest interpoint distance in $O(\log n)$ time once the solution to the ANN problem is known. It is also easy to show that this solution is time-optimal on meshes with multiple broadcasting. However, intuitively, one would expect a much simpler algorithm for the CP problem. No such algorithm is known. The straightforward implementation of the classical sequential algorithm [35] runs in $O(\log^2 n)$ time which is suboptimal. It would be of interest to devise a simpler time-optimal algorithm for the CP problem.

Recently, several authors [4, 6, 8, 14] have shown that rectangular meshes sometimes yield faster algorithms than square meshes. It would be interesting to see whether this is also the case for the ANN problem. This promises to be an interesting area for further investigation.

ACKNOWLEDGMENTS

We are grateful to two anonymous referees for a number of constructive comments that have made this into a much better paper. We also thank Professors Sahni and Prasanna for their timely and professional way of handling our submission.

REFERENCES

1. Aggarwal, A. Optimal bounds for finding maximum on array of processors with k global buses. *IEEE Trans. Comput.* **C-35** (1986), 62–64.
2. Akl, S. G., and Lyons, K. A. *Parallel Computational Geometry* Prentice-Hall, Englewood Cliffs, NJ, 1993.
3. Ballard, D. H., and Brown, C. M. *Computer Vision*. Prentice-Hall, Englewood Cliffs, NJ, 1982.
4. Bar-Noy, A., and Peleg, D. Square meshes are not always optimal. *IEEE Trans. Comput.* **C-40** (1991), 196–204.
5. Batcher, K. E. Design of massively parallel processor, *IEEE Trans. Comput.* **C-29** (1980), 836–840.

6. Bhagavathi, D., Gurla, H., Olariu, S., Schwing, J. L., and Zhang, J. Square meshes are not optimal for convex hull computation. *IEEE Trans. Parallel Distrib. Systems*, to appear.
7. Bhagavathi, D., Bokka, V., Gurla, H., Olariu, S., Schwing, J. L., and Stojmenović, I. Time-optimal visibility-related problems on meshes with multiple broadcasting. *IEEE Trans. Parallel Distrib. Systems* **6** (1995), 687–703.
8. Bhagavathi, D., Looges, P. J., Olariu, S., Schwing, J. L., and Zhang, J. A fast selection algorithm on meshes with multiple broadcasting. *IEEE Trans. Parallel Distrib. Systems* **5** (1994), 772–778.
9. Bhagavathi, D., Olariu, S., Shen, W., and Wilson, L. A time-optimal multiple search algorithm on enhanced meshes, with applications. *J. Parallel Distrib. Comput.* **22** (1994), 113–120.
10. Bhagavathi, D., Olariu, S., Schwing, J. L., Shen, W., Wilson, L., and Zhang, J. Convexity problems on meshes with multiple broadcasting. *J. Parallel Distrib. Comput.* **27** (1995), 142–156.
11. Blanz, W.-E., Petkovic, D., and Sanz, J. L. C. Algorithms and architectures for machine vision. In Chen, C. H. (Ed.). *Signal Processing Handbook*. Dekker, New York, 1989.
12. Bokhari, S. H. Finding maximum on an array processor with a global bus. *IEEE Trans. Comput.* **C-33** (1984), 133–139.
13. Cahn, R., Poulsen, R., and Toussaint, G. Segmentation of cervical cell images. *J. Histochem. Cytochem.* **25** (1977), 681–688.
14. Chen, Y. C., Chen, W. T., Chen, G. H., and Sheu, J. P. Designing efficient parallel algorithms on mesh connected computers with multiple broadcasting. *IEEE Trans. Parallel Distrib. Systems* **1** (1990), 241–246.
15. Chen, Y. C., Chen, W. T., and Chen, G.-H. Efficient median finding and its application to two-variable linear programming on mesh-connected computers with multiple broadcasting. *J. Parallel Distrib. Comput.* **15** (1992), 79–84.
16. Cole, R., and Goodrich, M. T. Optimal parallel algorithms for point-set and polygon problems. *Algorithmica* **7** (1992), 3–23.
17. Cook, S. A., Dwork, C., and Reischuk, R. Upper and lower time bounds for parallel random access machines without simultaneous writes. *SIAM J. Comput.* **15** (1986), 87–97.
18. Duda, R. O., and Hart, P. E. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.
19. Gualtieri, J. A., LeMoigne, J., and Packer, C. V. Distance between images. *Proc. 4th Symposium on Frontiers of Massively Parallel Computation*. McLean, VA, 1992, pp. 216–223.
20. JáJá, J. *An Introduction to Parallel Algorithms*. Addison–Wesley, Reading, MA, 1991.
21. Jeong, C. S., and Lee, D. T. Parallel geometric algorithms on a mesh-connected computer. *Algorithmica* **5** (1990), 155–177.
22. Kumar, V. P., and Raghavendra, C. S. Array processor with multiple broadcasting. *J. Parallel Distrib. Comput.* **2** (1987), 173–190.
23. Kumar, V. P., and Reisis, D. I. Image computations on meshes with multiple broadcast. *IEEE Trans. Pattern Anal. Mach. Intelligence PAMI-11* (1989), 1194–1201.
24. Lee, D. T., and Preparata, F. P. The all nearest neighbor problem for convex polygons. *Inform. Process. Lett.* **7** (1978), 189–192.
25. Li, H., and Maresca, M. Polymorphic-torus network. *IEEE Trans. Comput.* **C-38** (1989) 1345–1351.
26. Lin, R., Olariu, S., Schwing, J. L., and Zhang, J. Simulating enhanced meshes, with applications. *Parallel Process. Lett.* **3** (1993), 59–70.
27. Lin, R., Olariu, S., and Schwing, J. L. An efficient VLSI architecture for digital geometry. *Proc. of International IEEE Conference on Application-Specific Array Processors*. San Francisco, 1994, pp. 392–403.
28. Li, H., and Stout, Q. F. (Eds.). *Reconfigurable Massively Parallel Computers*. Prentice–Hall, Englewood Cliffs, NJ, 1991.
29. Lu, M. Constructing the Voronoi diagram on a mesh-connected computer. *Proc. of the International Conference on Parallel Processing* (1985), 806–811.
30. Maresca, M., and Li, H. Connection autonomy and SIMD computers: A VLSI implementation. *J. Parallel Distrib. Comput.* **7** (1989), 302–320.
31. Miller, R., and Stout, Q. Mesh computer algorithms for computational geometry. *IEEE Trans. Comput.* **38** (1989), 321–340.
32. Olariu, S., Schwing, J. L., and Zhang, J. Optimal convex hull algorithms on enhanced meshes. *BIT* **33** (1993), 396–410.
33. Parkinson, D., Hunt, D. J., and MacQueen, K. S. The AMT DAP 500. *33rd IEEE Comp. Soc. International Conf.* 1988, pp. 196–199.
34. Pavlidis, T. *Computer Graphics*. Comput. Sci. Press, Potomac, MD, 1978.
35. Preparata, F. P., and Shamos, M. I. *Computational Geometry—An Introduction*. Springer-Verlag, Berlin, 1988.
36. Reddaway, S. F., Wilson, A., and Horn, A. Fractal graphics and image compression on a SIMD processor. *Proc. 2nd Symp. on Frontiers of Massively Parallel Computation*. Fairfax, VA, 1988, pp. 265–274.
37. Quinn, M. J. *Parallel Computing: Theory and Practice*. McGraw–Hill, New York, 1994.
38. Schieber, B., and Vishkin, U. Finding all nearest neighbors for convex polygons in parallel: A new lower bound technique and a matching algorithm. *Discrete Appl. Math.* **39** (1990), 97–111.
39. Toussaint, G. T. (Ed.). *Computational Geometry*. Elsevier, North-Holland, Amsterdam, 1985.

STEPHAN OLARIU received the M.Sc. and Ph.D. degrees in computer science from McGill University, Montreal in 1983 and 1986, respectively. In 1986 he joined the Computer Science Department at Old Dominion University. Dr. Olariu has published more than 130 papers in various journals, book chapters, and conference proceedings. His research interests include image processing and machine vision, parallel architectures, design and analysis of parallel algorithms, computational graph theory, computational geometry, and computational morphology. Dr. Olariu is an associate editor of *International Journal of Computer Mathematics and Applications* and serves on the editorial board of *VLSI Design and Parallel Algorithms and Applications*. He has been on the program committee of several conferences and workshops.

IVAN STOJMENOVIC was born in 1957, in Yugoslavia. He received the B.S. and M.S. degrees in 1979 and 1983 from the University of Novi Sad (Yugoslavia) and the Ph.D. Math. degree in 1985 from the University of Zagreb (Croatia). In 1980 he joined the Institute of Mathematics, University of Novi Sad. During the winter of 1985/1986 he was a visiting researcher at the Electrotechnical Laboratory, Tsukuba, Japan. In the fall of 1987 and spring of 1988 he was a visiting assistant professor at the Computer Science Department, Washington State University (Pulman, WA) and Department of Mathematics and Computer Science, University of Miami (Miami, FL), respectively. In the fall of 1988, he joined the faculty of the Computer Science Department at the University of Ottawa (Ottawa, Canada), where currently he holds the position of an associate professor. He published three books and about 130 different papers in journals and conference proceedings. His research interests are computational geometry, parallel computing, combinatorial algorithms, multiple-valued logic, and graph theory. He is currently an editor of the following journals: *Parallel Processing Letters*, *Parallel Algorithms and Applications*, *Multiple-Valued Logic*, *Journal of Computing and Information*, and *Tangenta*.