Time-Optimal Visibility-Related Algorithms on Meshes with Multiple Broadcasting *

D. Bhagavathi[†] V. Bokka[‡] H. Gurla[‡] S. Olariu[‡]
J. L. Schwing[‡] I. Stojmenović [§] J. Zhang[¶]

Abstract

Given a collection of objects in the plane along with a viewpoint ω, the visibility problem involves determining the portion of each object that is visible to an observer positioned at ω. The visibility problem is central to various application areas including computer graphics, image processing, VLSI design, and robot navigation, among many others. The main contribution of this work is to provide time-optimal solutions to this problem for several classes of objects, namely ordered line segments, disks, and iso-oriented rectangles in the plane. In addition, our visibility algorithm for line segments is at the heart of time-optimal solutions for determining, for each element in a given sequence of real numbers, the position of the nearest larger element within that sequence, triangulating a set of points in the plane, determining the visibility pairs among a set of vertical line segments, and constructing the dominance and visibility graphs of a set of iso-oriented rectangles in the plane. All the algorithms in this paper involve an input of size n and run in O($\log n$) time on a mesh with multiple broadcasting of size $n \times n$. This is the first instance of time-optimal solutions for these problems on this architecture.
Keywords: visibility, triangulation, compaction, dominance graph, visibility graph, robotics, image processing, computer graphics, VLSI design, computational geometry, meshes with multiple broadcasting, parallel algorithms, time-optimal algorithms.

1 Introduction

Being a natural platform for solving a large number of problems in computer graphics, image processing, robotics, and VLSI design, the mesh-connected computer has emerged as one of the most widely investigated parallel models of computation. In addition, due to its simple and regular interconnection topology, the mesh is well suited for VLSI implementation [3]. However, as a result of its large diameter, the mesh does not deliver high performance in applications requiring *nonspatially organized* communications [17] where several hops have to be performed to complete data exchanges between nonadjacent processors.

To overcome this problem, the mesh architecture has been enhanced by various types of bus systems [11, 19, 22, 28, 37, 39]. Early solutions, involving the addition of one or more global buses, shared by all the processors, have been implemented on a number of massively

*Work supported by NASA grant NCC1-99, by NSF grant CCR-8909996, and by NSERC grant OGPIN007. Preliminary versions of this work have appeared in *Proc. of ASAP'93*, Venice, Italy, October 1993 and in *Proc. of IPPS'94*, Cancun, Mexico, April 1994.
[†]Department of Computer Science, Southern Illinois University, Edwardsville, IL 62026, U.S.A.
[‡]Department of Computer Science, Old Dominion University, Norfolk, VA 23529, U.S.A.
[§]Department of Computer Science, University of Ottawa, Ottawa, Ontario, K1N 9B4, Canada
[¶]Dept. of Math. & Computer Science, Elizabeth City State University, Elizabeth City, NC 27909, U.S.A.

parallel machines [11]. Recently, a more powerful architecture, referred to as mesh with multiple broadcasting, has been obtained by adding one bus to every row and to every column of the mesh [19, 33]. The mesh with multiple broadcasting has proven to be feasible to implement in VLSI, and is used in the DAP family of computers [33].

Being of theoretical interest as well as commercially available, the mesh with multiple broadcasting has attracted a great deal of attention. In recent years, efficient algorithms to solve a number of computational problems on meshes with multiple broadcasting have been proposed in the literature. These include image processing [20, 33], computational geometry [7, 8, 10, 19, 30, 31, 32], semigroup computations [2, 9, 13, 19], sorting [5], multiple-searching [7], and selection [6, 13, 19], among others.

A recurring problem in a number of contexts in computer graphics, VLSI design, and robot navigation involves computing the visibility of a collection of objects in the plane from a distinguished point ω. In computer graphics, for example, visibility from a point plays a crucial role in ray tracing and hidden line elimination [16, 34]. The same problem arises in path planning and collision avoidance problems in robotics [25, 41, 42] where a navigational course for a mobile robot is sought in the presence of various obstacles. Yet another fertile field of application is provided by VLSI, where visibility plays a fundamental role in the compaction process of integrated circuit design [24, 27, 29, 35, 36]. In this latter context, it is customary to formulate the compaction problem as a visibility problem involving a collection of iso-oriented, non-overlapping, rectangles in the plane. For simplicity reasons, the compaction process is often one-dimensional, i.e. the components are moved in the x-direction or y-direction only. Hence, it is convenient to abstract rectangles as vertical or horizontal line segments [44]. In this context, the compaction is referred to as "stick" compaction and reduces to a special instance of the visibility problem of vertical line segments [24, 26, 38].

The main contribution of this work is to provide time-optimal solutions to the visibility problem for various classes of objects, namely ordered segments, disks, and iso-oriented rectangles in the plane. The key to our time-optimal visibility algorithms for disks and iso-oriented rectangles is a novel, time-optimal, algorithm to solve the visibility problem for an ordered set of line segments. This problem, referred to as segment visibility, can be described generically as follows. Let a point ω in the plane be given along with an ordered set $S = s_1, s_2, \ldots, s_n$ of non-intersecting line segments in the same plane. We are interested in determining the portions of each segment s_i that are visible to an observer positioned at ω. In addition, our solution to the segment visibility problem will be used, as a basic ingredient, in time-optimal algorithms for a host of problems motivated by, and finding applications to, robotics, computer graphics, and VLSI design. Examples include finding the all nearest larger values for a given sequence of real numbers, triangulating a set of points in the plane, determining the visibility pairs among a given set of vertical segments, and constructing the dominance and visibility graphs of a set of iso-oriented rectangles in the plane. All the algorithms in this paper involve an input of size n and run in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$.

The segment visibility problem and its variants have attracted a good deal of attention in the literature. A solution running in $O(\log n)$ time and using n processors in the CREW-PRAM model is presented in [18]. This algorithm uses the concept of *plane-sweep tree* of Atallah *et al.* [1]. The construction of the plane sweep tree is nontrivial and uses the powerful technique of cascading divide-and-conquer. Yet another solution to the vertical segment visibility problem of the same time and processor complexity and using cascading divide-and-conquer has been reported in [12]. By contrast, our solution to the segment visibility problem is very simple and does not rely on the plane-sweep tree or on cascading divide-and-conquer.

The remainder of the paper is organized as follows: Section 2 briefly describes the model of

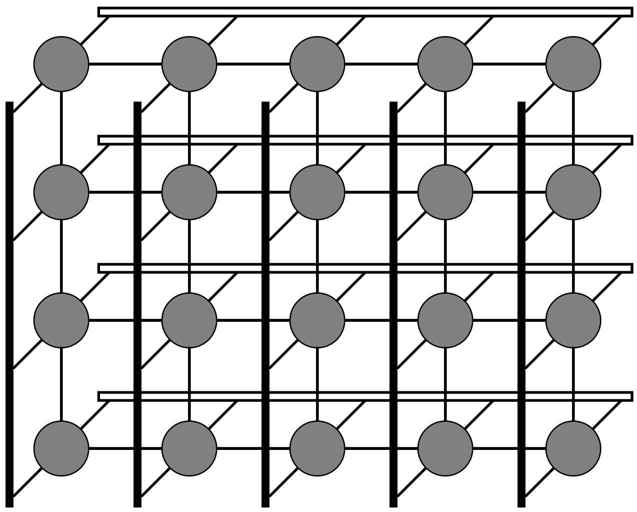

Figure 1: A mesh with multiple broadcasting of size 4×5

computation. Section 3 states some basic results that will be useful in subsequent sections. Section 4 presents lower bound arguments as well as the details of a time-optimal algorithm for the segment visibility problem. Section 5 discusses various applications of the algorithm developed in Section 4 along with corresponding time lower bound arguments. Specifically, Subsection 5.1 discusses the all nearest larger values problem; Subsection 5.2 discusses the triangulation problem; Subsection 5.3 deals with the disk visibility problem; Subsection 5.4 describes the rectangle visibility problem; Subsection 5.5 addresses the problem of determining the visibility pairs among a set of vertical line segments as well as the construction of the visibility graph of a set of iso-oriented rectangles. Finally, Subsection 5.6 presents the construction of the dominance graph of a set of iso-oriented rectangles in the plane. The paper concludes with Section 6 in which the results are summarized and a number of open problems are posed.

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 Figure 1.

Processor $P(i,j)$ 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. Every processor $P(i,j)$ is connected to its four neighbors $P(i-1,j)$, $P(i+1,j)$, $P(i,j-1)$, $P(i,j+1)$, 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. Each processor is assumed to know its own coordinates within the mesh and to have a constant number of registers of size $O(\log MN)$. In unit time, every 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 [2, 11, 13, 19, 20, 22, 28, 33, 37], we assume that communications along buses take $O(1)$ time. Although inexact, recent experiments with the DAP and the YUPPIE multiprocessor array systems seem to indicate that this is a reasonable working hypothesis [22, 28, 33].

A PRAM [18] 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.

A mesh with multiple broadcasting can be perceived as a restricted version of the CREW-PRAM: the buses are nothing more than *oblivious* concurrent read, exclusive write registers with the access restricted to certain sets of processors. Indeed, a square mesh with multiple broadcasting using p processors can be viewed as a CREW-PRAM with p processors where 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 algorithms.

3 Preliminaries

The purpose of this section is to review a number of basic results for the mesh with multiple broadcasting that will be instrumental in the design of our algorithms.

The well-known OR problem, given a sequence of n bits $b_1, b_2, \ldots, b_n$, asks for computing their logical OR. We begin by stating a fundamental result of Cook *et al.* [14] that will be used in all the time lower bound arguments in this paper.

Proposition 3.1. [14] 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 used. □

In addition, our arguments rely on the following result of Lin *et al.* [23].

Proposition 3.2. 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 3.2 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 [23] to transfer known time lower bounds for the PRAM to the mesh with multiple broadcasting.

Merging two sorted sequences is one of the fundamental operations in computer science. Recently, Olariu *et al.* [30] have proposed an $O(1)$ 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 [30].

Proposition 3.3. Let $S_1 = (a_1, a_2, \ldots, a_r)$ and $S_2 = (b_1, b_2, \ldots, b_s)$, 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 $(1 \leq i \leq s)$. These two sequences can be merged into a sorted sequence S in O(1) time. □

Since merging is an important ingredient in our visibility algorithms, we now give the details of the merging algorithm in [30]. 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. It is easy to see that for every i, a unique processor $P(i, r+j)$ $(1 \leq j \leq s)$, will find that $b_{j-1} < a_i \leq b_j$ (b_0 is taken to be $-\infty$). Clearly, this unique processor can now use the horizontal bus to broadcast j back to $P(i,i)$. In turn, $P(i,i)$ has enough information to compute the position of a_i in S. In exactly the same way, the position of every b_j in S 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.

Proposition 3.3 is the main stepping stone for a time-optimal sorting algorithm developed in [30]. This algorithm implements the well-known strategy of sorting by merging. Specifically, in [30] the following result was established.

Proposition 3.4. 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. □

To make this paper self-contained, we briefly sketch the data movement operations performed in the sorting algorithm of [30]. First, the input sequence is divided into a left subsequence containing the first $\frac{n}{2}$ items and a right subsequence containing the remaining $\frac{n}{2}$ items. Further, imagine dividing the original mesh into four equal submeshes of size $\frac{n}{2} \times \frac{n}{2}$. Note that for computational purposes, the north-west and south-east submeshes can be treated as independent meshes with multiple broadcasting.

In preparation for sorting, the right subsequence is broadcast to the first row of the south-eastern submesh. The algorithm then proceeds to recursively sort the data in each submesh. The resulting sorted subsequences are merged using the process described in Proposition 3.3. 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 3.1 and 3.2 by reducing the OR problem to sorting.

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 $a_1, a_2, \ldots, a_n$ 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 an ordered 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; 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 3.5. Consider a sequence $a_1, a_2, \ldots, a_n$ 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 an ordered 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. □

The convex hull of a set of planar points is the smallest convex set containing the given set. Quite recently, Olariu *et al.* [30] have proposed a time-optimal algorithm to compute the convex hull of a set of points in the plane. More precisely, they proved the following result.

Proposition 3.6. The convex hull of an n-element set of points in the plane, stored one item per processor in one row or one column of a mesh with multiple broadcasting of size $n \times n$ can be computed in $O(\log n)$ time. Furthermore, this is time-optimal on this architecture. □

4 Endpoint and Segment Visibility

In this section, we exhibit time lower bounds for the segment visibility problem as well as for a closely related problem termed the endpoint visibility problem, on meshes with multiple broadcasting. In fact, the time lower bound also holds for the CREW-PRAM. We then discuss an algorithm to solve both the segment and endpoint visibility problems whose running time on the mesh with multiple broadcasting matches the lower bound.

Before formally stating the problems that we address, we need to introduce a few terms. Let ω be a distinguished point and let $S = s_1, s_2, \ldots, s_n$ be a collection of non-intersecting line segments in the plane. The collection S is said to be *well ordered* if for every i, j $(1 \le i, j \le n)$, $i < j$ guarantees that any ray that originates at ω and intersects both s_i and s_j, intersects s_i before s_j.

For an endpoint e of a line segment in S, we let $e\omega$ denote the ray originating at e and directed towards ω; similarly, we let $e\overline{\omega}$ be the ray emanating from e, collinear with ω and away from ω. We now state the endpoint visibility problem (EV, for short) which turns out to be intimately related to the segment visibility problem that we have already mentioned informally. Specifically, given a collection S of well ordered line segments, the EV problem asks to determine, for every endpoint e of a segment in S, the closest segments (if any) intersected by the rays $e\omega$ and $e\overline{\omega}$. As an example, in Figure 2, the closest segments intersected by the rays $f_3\omega$ and $f_3\overline{\omega}$ are s_1 and s_6, respectively.

To state the segment visibility problem (SV, for short), we define the *contour* of S from ω to be the ordered sequence of segment portions that are visible to an observer positioned at ω. Now, the SV problem asks to compute the contour of S from ω. For an illustration refer to Figure 2 where the sequence of heavy lines, when traversed in increasing polar angle about ω, yields the contour of the collection of segments.

We establish an $\Omega(\log n)$ lower bound for the EV problem on the CREW-PRAM by reducing OR to EV. Let $b_1, b_2, \ldots, b_n$ be an arbitrary input to the OR problem. Now consider any algorithm that correctly solves the EV problem with ω at $(-\infty, 0)$ and with input $z_0, z_1, z_2, \ldots, z_{n+1}$, where z_i is the vertical segment with endpoints $(i, 0)$ and $(i, 3)$ in case $b_i = 1$, and the segment with endpoints $(i, 0)$ and $(i, 1)$ if $b_i = 0$. To complete the construction, we let z_0 and z_{n+1} be the segments with endpoints $(0, 0)$ and $(0, 2)$, and $(n + 1, 0)$ and $(n + 1, 3)$, respectively. The construction guarantees that the resulting set of segments is well ordered. Clearly, the answer to the OR problem is 0 if, and only if, the ray $z_0\overline{\omega}$ encounters the segment z_{n+1}. The conclusion follows by Proposition 3.1. To summarize our discussion we state the following result.

Lemma 4.1. The task of solving the endpoint visibility problem for a collection of n well ordered line segments in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □

Lemma 4.1 and Proposition 3.2 combined, imply the following result.

Corollary 4.2. The task of solving the endpoint visibility problem for a collection of n well ordered line segments in the plane has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$. □

Next, we show that the same lower bound applies to the SV problem. As before, we shall reduce OR to SV. Let $b_1, b_2, \ldots, b_n$ be an arbitrary input to the OR problem. Now consider any algorithm that correctly solves the SV problem with input $z_1, z_2, \ldots, z_{n+1}$, where z_i is the vertical segment with endpoints $(i, 0)$ and $(i, 1)$ in case $b_i = 1$, and the (degenerate) segment

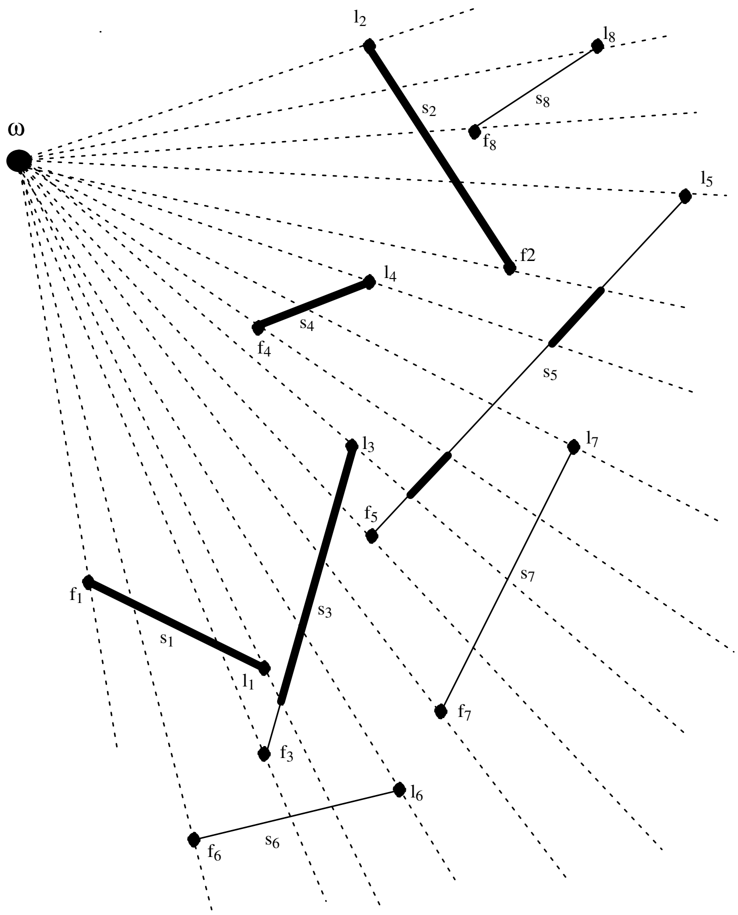

Figure 2: Illustrating the endpoint and segment visibility problems

with endpoints $(i, 0)$ and $(i, 0)$ if $b_i = 0$. To complete the construction, we let z_{n+1} be the segment with endpoints $(n+1, 0)$ and $(n+1, 1)$ and we place the viewpoint ω at $(0, 1)$. The construction guarantees that the resulting set of segments is well ordered. Clearly, the answer to the OR problem is 0 if, and only if, the entire segment z_{n+1} is visible from ω. The conclusion follows by Proposition 3.1. To summarize, we state the following result.

Lemma 4.3. The task of solving the segment visibility problem for a collection of n well ordered line segments in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □

Lemma 4.3 and Proposition 3.2 combined, imply the following result.

Corollary 4.4. The task of solving the segment visibility problem for a collection of n well ordered line segments in the plane has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$. □

Our next goal is to show that the time lower bounds of Corollaries 4.2 and 4.4 are tight, by devising an algorithm that solves an arbitrary instance of size n of the EV and SV problems in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Consider an arbitrary collection $S = s_1, s_2, \ldots, s_n$ of *well ordered* line segments, with every segment being specified by its endpoints. The set S is assumed to be stored, one segment per processor, in the first row of a mesh with multiple broadcasting of size $n \times n$. Without loss of generality, we assume that the viewpoint ω lies to the left of S (i.e. its x-coordinate is smaller than that of any endpoint of a segment in S). The endpoints are specified by their polar coordinates with ω as pole and the vertical ray from ω to $-\infty$ as polar axis. We assume that the segments are in general position, with no two endpoints sharing the same polar angle. The reader will not fail to observe that these assumptions are made for convenience only and are, in fact, non-essential. For example, if ω does not lie to the left of S, the problem can be divided into two subproblems by splitting some of the segments into two parts, if necessary. The solutions to the two subproblems can be easily combined to yield the required solution.

Every line segment s_i in S has its endpoints denoted in increasing polar angle as f_i and l_i, standing for *first* and *last*, respectively. With a generic endpoint e_i of segment s_i we associate the following variables:

- the identity of the segment to which it belongs (i.e. s_i);
- a bit indicating whether e_i is the first or last endpoint of s_i;
- $t(e_i)$, the identity of the first segment, if any, that blocks the ray $e_i\omega$;
- $a(e_i)$, the identity of the first segment, if any, that blocks the ray $e_i\overline{\omega}$.

The notation $t(e_i)$ and $a(e_i)$ is meant to indicate directions "towards" and "away" from the viewpoint ω, respectively. Initially, $t(e_i)$=$a(e_i)$=0; when the algorithm terminates, $t(e_i)$ and $a(e_i)$ will contain the desired solutions. It is perhaps appropriate, before we get into details, to give a brief description of how the problem at hand is solved. The algorithm begins by computing an "approximate" solution to the EV problem: this involves determining for each of the rays $e_i\omega$ and $e_i\overline{\omega}$ whether it is blocked by some segment in S, without specifying the identity of the segment. This approximate solution is then refined into an exact solution.

Let us proceed with a high-level description of the algorithm. Imagine planting a complete binary tree T on S, with the leaves corresponding, in left-to-right order, to the segments in S. Given an arbitrary node v of T, we let $L(v)$ stand for the set of leaf-descendants of v. We further assume that the nodes in T are numbered level by level in left-to-right order. For a generic endpoint e_i of segment s_i, we let:

- t-blocked(e_i) stand for the identity of the first node in T on the path from the leaf storing the segment s_i to the root, at which it is known that the ray $e_i\omega$ is blocked by some segment in S;
- a-blocked(e_i) stand for the identity of the first node in T on the path from the leaf storing s_i

to the root, at which it is known that the ray $e_i\varpi$ is blocked by some segment in S. Both t-blocked(e_i) and a-blocked(e_i) are initialized to 0.

Our algorithm proceeds in two stages. In the first stage, the tree T is traversed, in parallel, from the leaves to the root, computing for every endpoint e_i, t-blocked(e_i) and a-blocked(e_i). As we shall demonstrate, in case t-blocked(e_i) is not 0, we are guaranteed that some segment in S blocks the ray $e_i\omega$. However, the identity of the blocking segment is not known at this stage. Similarly, if a-blocked(e_i) is not 0, then we are guaranteed that some segment in S blocks the ray $e_i\varpi$. As before, the identity of the blocking segment is unknown. In the second stage of the algorithm, the tree T is traversed again, from the leaves to the root. In this process, the information in t-blocked(e_i) and a-blocked(e_i) is refined into t(e_i) and a(e_i).

For convenience, we view the algorithm as a sequence of processing tasks involving nodes of T. A node v of T is said to be "processed" when the subproblem involving segments in $L(v)$ has been processed. Specifically, consider a generic node v of T with left and right children u and w, respectively. The following variables are associated with node v:

- $E(v)$ – the list of endpoints of segments in $L(v)$ sorted by increasing polar angle;
- BT(v) – the set of all endpoints e_i in $L(v)$ for which t-blocked(e_i)=v;
- BA(v) – the set of all endpoints e_i in $L(v)$ for which a-blocked(e_i)=v;
- LC(v) – the set of all endpoints e_i in $L(v)$ for which t-blocked(e_i)=0;
- RC(v) – the set of all endpoints e_i in $L(v)$ for which a-blocked(e_i)=0.

The sets BT(v), BA(v) are initialized to the empty set. For a leaf α of T, $E(\alpha)$, LC(α), and RC(α) contain the two endpoints of the corresponding segment in S, sorted by increasing polar angle.

We are now in a position to give the details of the two stages of our algorithm.

Stage 1. Consider a generic node v in T with left and right children u and w, respectively. We now describe the tasks performed in the transition from u and w to v. First, $E(v)$ is obtained by merging $E(u)$ and $E(w)$. By Proposition 3.1, this task is carried out in O(1) time. For every endpoint e_i in the sorted list $E(u)$, let pred$(e_i, E(w))$ and succ$(e_i, E(w))$ stand for the *predecessor* and *successor* in $E(w)$, that is, the endpoints that precede and succeed e_i in $E(w)$, respectively. For an endpoint e_i in $E(w)$ the predecessor and successor pred$(e_i, E(u))$ and succ$(e_i, E(u))$ in $E(u)$ are defined analogously. Note that in the process of merging $E(u)$ and $E(w)$ into $E(v)$, every endpoint e_i updates its predecessor and successor information in O(1) time, as described in the merging algorithm in Section 3.

Next, we describe how t-blocked(e_i) and a-blocked(e_i) are computed. The well ordering of the segments in S guarantees that if an endpoint e_i in $E(u)$ has t-blocked(e_i)=0 just prior to processing v, then t-blocked(e_i)=0 holds after v has been processed. Similarly, if the endpoint e_i in $E(w)$ has a-blocked(e_i)=0 just prior to processing v, then a-blocked(e_i)=0 after v has been processed. Now, let e_i be an endpoint in $E(u)$ with a-blocked(e_i)=0. Write e_j=pred$(e_i, E(w))$ and e_k=succ$(e_i, E(w))$. After v has been processed, a-blocked(e_i)=0 only if e_k and e_j belong to different segments and t-blocked(e_j)=a-blocked(e_j)=t-blocked(e_k)=a-blocked(e_k)=0; otherwise, a-blocked(e_i) is set to v. Similarly, let e_i be an endpoint in $E(w)$ with t-blocked(e_i)=0, and write e_j=pred$(e_i, E(w))$ and e_k=succ$(e_i, E(w))$. Now t-blocked(e_i)=0 after processing v, only if e_k and e_j belong to different segments and t-blocked(e_j)=a-blocked(e_j)=t-blocked(e_k)=a-blocked(e_k)=0; otherwise, t-blocked(e_i) is set to v.

The correctness of this assignment is guaranteed by the following result.

Lemma 4.5.

(a) Let e_i be an endpoint in $E(u)$ with a-blocked(e_i)=0. If, in the transition from u and w to v, a-blocked(e_i)=v, then the ray $e_i\varpi$ intersects some segment in $L(w)$.

(b) Let e_i be an endpoint in $E(w)$ with t-blocked(e_i)=0. If, in the transition from u and w to

v, t-blocked(e_i)=v, then the ray $e_i\omega$ intersects some segment in $L(u)$.

Proof. The proof is by induction on the level of v in T. The statement is vacuously true at the leaves of T which are at level 0. Assume that both (a) and (b) hold for u and w, and suppose that in the transition from u and w to v, a-blocked(e_i)=v for some endpoint e_i in $E(u)$. As above, write e_j=pred($e_i, E(w)$) and e_k=succ($e_i, E(w)$).

Since a-blocked(e_i)=v, one of the following cases must have occurred.

Case 1. e_j and e_k belong to the same segment.
Let s_p be the segment in $S(w)$ with endpoints e_j and e_k. Since S is well ordered, we must have $i < p$ and, consequently, s_p blocks the ray $e_i\overline{\omega}$, as claimed.

Case 2. a-blocked(e_j)$\neq 0$ or a-blocked(e_k)$\neq 0$.
We only discuss the case a-blocked(e_k)$\neq 0$, the other following by a mirror argument. By the induction hypothesis, a-blocked(e_k)$\neq 0$ guarantees the existence of a segment s_q in $S(w)$ that blocks the ray $e_k\overline{\omega}$. Since S is well ordered, we must have $i < q$. Furthermore, since e_j and e_k are consecutive in $E(w)$, the first endpoint of s_q cannot occur between e_j and e_k and, therefore, s_q blocks the ray $e_i\overline{\omega}$.

Case 3. t-blocked(e_j)$\neq 0$ or t-blocked(e_k)$\neq 0$.
We only discuss the case t-blocked(e_j)$\neq 0$, the other following by a mirror argument. By the induction hypothesis, t-blocked(e_j)$\neq 0$ guarantees the existence of a segment s_p in $S(w)$ that blocks the ray $e_j\omega$. The fact that S is well ordered guarantees that $i < p$. Since e_j and e_k are consecutive in $E(w)$, the last endpoint of s_p cannot occur between e_j and e_k and, therefore, s_p blocks the ray $e_i\overline{\omega}$.

This completes the proof of (a). The proof of (b) is similar and, therefore, omitted. □

By virtue of Lemma 4.5, when *root*(T), the root of T, is reached at the end of Stage 1, all the endpoints e_i having t-blocked(e_i)=0 know that the ray $e_i\omega$ is blocked by no segment in $\mathcal{S}$. All the endpoints e_i with a-blocked(e_i)=0 set a(e_i) = $+\infty$.

Stage 2. The main goal of this stage is to use the information obtained in Stage 1 to compute the actual values of t(e_i) and a(e_i) for every endpoint e_i. A key role in the computation specific to this stage is being played by the sets BT(v), BA(v), LC(v), and RC(v) defined in the preamble to the algorithm.

We begin by sorting the endpoints of segments in S separately, first by a-blocked(e_i) and then by t-blocked(e_i). By Proposition 3.4 this operation can be performed in O($\log n$) time. As a result, we obtain two sorted lists: in the first one, all the endpoints that have the value a-blocked(e_i)=v occur consecutively, and will be referred to as BA(v). In the second one, all the endpoints that have the value t-blocked(e_i)=v occur consecutively, and will be denoted by BT(v). We may assume that both BT(v) and BA(v) feature endpoints sorted in increasing polar angle: this can be easily achieved by using two keys for sorting and the complexity will not be affected.

At this point, we need to explain the intuition for the sets LC(v), and RC(v). As it will become apparent, LC(v) contains a sorted list of endpoints e_i in $E(v)$ whose t-blocked(e_i) = 0 after node v in T has been processed. Put differently, Lemma 4.5 guarantees that LC(v) contains all the endpoints in $E(v)$ for which the ray $e_i\omega$ is blocked by no segment in $L(v)$. For this reason, and since ω lies to the left of S, we shall refer to LC(v) as the *left contour* at v. It is important to note that the left contour LC(v) provides a partial solution to the segment visibility problem. The set RC(v) is defined similarly and will be referred to as the *right contour* at v.

Consider, again, a generic node v in T with left and right children u and w, respectively. We now show how the sets RC(u), RC(w), LC(u), and LC(w) are updated into RC(v) and LC(v)

in the transition from u and w to v. Specifically, with $\cup$ standing for the set-merge operation, we set

$$\mathrm{RC}(v) = (\mathrm{RC}(w) \cup \mathrm{RC}(u)) - \mathrm{BA}(v) \tag{1}$$

and

$$\mathrm{LC}(v) = (\mathrm{LC}(w) \cup \mathrm{LC}(u)) - \mathrm{BT}(v). \tag{2}$$

For definiteness, we show how the list RC(v) in (1) is obtained from RC(u), RC(w), and BA(v). Begin by merging RC(u) and RC(w) into a list $E'(v)$. From $E'(v)$ we need to delete those endpoints e_i that have a-blocked(e_i)=v. For this purpose, we merge $E'(v)$ with the list BA(v) that is readily available by virtue of the sorting step described above. Again, by Proposition 3.3, the merging operation runs in O(1) time. It is important to note that in the process of merging $E'(v)$ and BA(v), every endpoint e_i whose a-blocked(e_i) value is 0 after node v has been processed, finds its predecessor and successor (i.e. its own rank) in BA(v). This information will allow these points to compute their ranks in RC(v). Now, Lemma 3.5 guarantees that we can obtain a compacted version of RC(v) in O(1) time. The computation of LC(v) in (2) is perfectly similar.

Finally, we need to show how the correct values of t(e_i) and a(e_i) are obtained for every endpoint e_i. Consider, again, the processing that takes place in the second stage of the algorithm, in the transition from u and w to v. Having computed the sets RC(u), RC(w), LC(u), and LC(w), we are ready to determine the values of t(e_i) and a(e_i) for all endpoints in BA(v) and BT(v). For this purpose, we merge RC(u) with BT(v). As noted before, Proposition 3.3 guarantees that this operation takes O(1) time. In the process of merging, every endpoint e_i in BT(v) determines the identity of two endpoints e_j and e_k such that e_j=pred(e_i,RC(u)) and e_k=succ(e_i,RC(u)). The value of t(e_i) is set as follows:

• in case e_j and e_k are endpoints of the same segment s_p, then t(e_i)=s_p;

• if both e_j and e_k are last endpoints, then t(e_i) is set to the segment s_p whose last endpoint is e_k;

• if both e_j and e_k are first endpoints, then t(e_i) is set to the segment s_p whose first endpoint is e_j;

• if e_k is a first endpoint and e_j is a last endpoint then t(e_i)=t(e_j)=t(e_k).

The correctness of this assignment follows by an easy inductive argument. The correct value of a(e_i) for endpoints e_i in BA(u) are computed similarly. Since the computation of Stage 2 takes O($\log n$) time, we have the following result.

Theorem 4.6. An arbitrary n-segment instance of the endpoint visibility problem can be solved in O($\log n$) time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal. □

It is important to note that from the information in LC($root(T)$) at the end of Stage 2, along with t(e_i) and a(e_i), the contour of S from ω can be computed in O(1) time as follows. Let LC($root(T)$) contain the endpoints $e_1, e_2, \ldots, e_m$ sorted in increasing polar angle. For every i ($2 \leq i \leq m$):

• if e_{i-1} and e_i belong to the same segment s_p in S, then s_p belongs to the contour;

• if e_{i-1} is a last endpoint and e_i is a first endpoint, then with s_p standing for the common value of a(e_{i-1}) and a(e_i), the portion of s_p between the rays $e_{i-1}\overline{\omega}$ and $e_i\overline{\omega}$ belongs to the contour;

• if both e_{i-1} and e_i are first endpoints, then with s_p standing for the segment whose first endpoint is e_{i-1}, the portion of s_p between e_{i-1} and the ray $e_i\overline{\omega}$ belongs to the contour;

• if both e_{i-1} and e_i are last endpoints, then with s_p standing for the segment whose last endpoint is e_i, the portion of s_p between the ray $e_{i-1}\overline{\omega}$ and e_i belongs to the contour.

Consequently, the algorithm just described also solves the SV problem. We summarize our

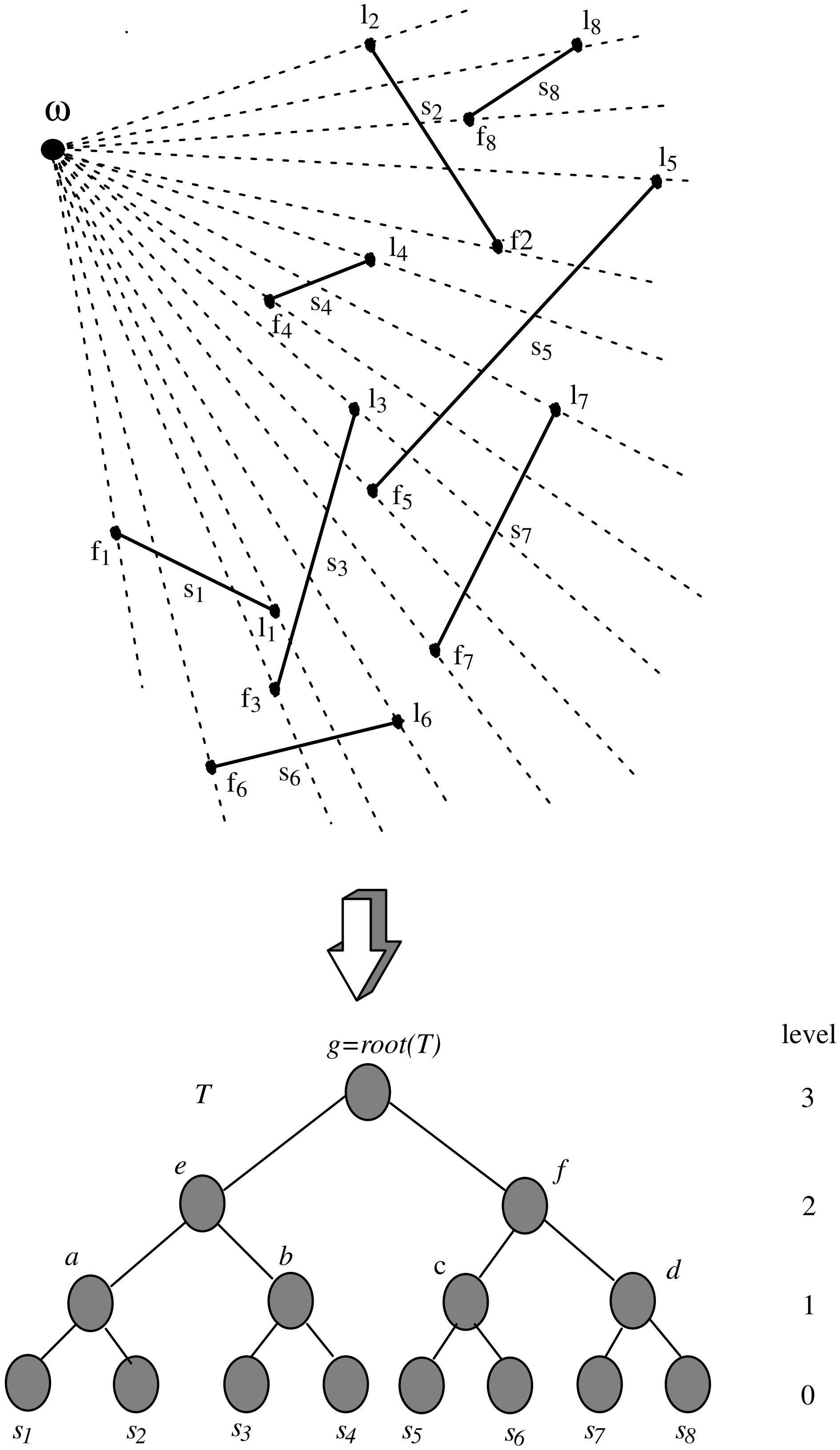

Figure 3: The set of segments in Figure 2 and the associated binary tree

level	0		1		2		3	
values of	t-blocked	a-blocked	t-blocked	a-blocked	t-blocked	a-blocked	t-blocked	a-blocked
f_1	0	0	0	0	0	0	0	0
l_1	0	0	0	0	0	e	0	e
f_2	0	0	0	0	0	0	0	g
l_2	0	0	0	0	0	0	0	0
f_3	0	0	0	0	e	0	e	g
l_3	0	0	0	0	0	0	0	g
f_4	0	0	0	0	0	0	0	g
l_4	0	0	0	0	0	0	0	g
f_5	0	0	0	0	0	f	g	f
l_5	0	0	0	0	0	0	g	0
f_6	0	0	0	0	0	0	g	0
l_6	0	0	0	0	0	0	g	0
f_7	0	0	0	0	0	0	g	0
l_7	0	0	0	0	f	0	f	0
f_8	0	0	0	0	0	0	g	0
l_8	0	0	0	0	0	0	g	0

Table 1: Illustrating Stage 1 of the algorithm

NODE	BT	BA	LC	RC
a	ϕ	ϕ	$f_1l_1f_2l_2$	$f_1l_1f_2l_2$
b	ϕ	ϕ	$f_3l_3f_4l_4$	$f_3l_3f_4l_4$
c	ϕ	ϕ	$f_6l_6f_5l_5$	$f_6l_6f_5l_5$
d	ϕ	ϕ	$f_7l_7f_8l_8$	$f_7l_7f_8l_8$
e	f_3	l_1	$f_1l_1l_3f_4l_4f_2l_2$	$f_1f_3l_3f_4l_4f_2l_2$
f	l_7	f_5	$f_6l_6f_7f_5l_5f_8l_8$	$f_6l_6f_7l_7l_5f_8l_8$
g	f_5 l_5 f_6 l_6 f_7 f_8 l_8	f_2 f_3 l_3 f_4 l_4	$f_1l_1l_3f_4l_4f_2l_2$	$f_6l_6f_7l_7l_5f_8l_8l_2$

Table 2: Illustrating Stage 2 of the algorithm

findings as follows.

Theorem 4.7. An arbitrary n-segment instance of the segment visibility problem can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal. □

A complete worked example based on the set of segments featured in Figure 2 is presented for the reader's benefit. Figure 3 shows the set of input segments along with the binary tree T that guides the algorithm. The various data items computed in Stage 1 are summarized in Table 1. The results of Stage 2 are captured, in succinct form, in Tables 2 and 3. Specifically, the solution to the endpoint visibility problem is contained in Table 3.

5 Applications

The purpose of this section is to show that the EV and SV problems discussed in the previous section yield time-optimal solutions to a number of problems of import to computer graphics, robotics, and VLSI design.

level →	0		1		2		3	
Values of	t	a	t	a	t	a	t	a
f_1	$-\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$
l_1	$-\infty$	0	$-\infty$	0	$-\infty$	s_3	$-\infty$	s_3
f_2	$-\infty$	0	$-\infty$	0	$-\infty$	0	$-\infty$	s_5
l_2	$-\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$	$-\infty$	$+\infty$
f_3	0	0	0	0	s_1	0	s_1	s_6
l_3	$-\infty$	0	$-\infty$	0	$-\infty$	0	$-\infty$	s_5
f_4	$-\infty$	0	$-\infty$	0	$-\infty$	0	$-\infty$	s_5
l_4	$-\infty$	0	$-\infty$	0	$-\infty$	0	$-\infty$	s_5
f_5	0	0	0	0	0	s_7	s_3	s_7
l_5	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_2	$+\infty$
f_6	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_1	$+\infty$
l_6	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_3	$+\infty$
f_7	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_3	$+\infty$
l_7	0	$+\infty$	0	$+\infty$	s_5	$+\infty$	s_5	$+\infty$
f_8	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_2	$+\infty$
l_8	0	$+\infty$	0	$+\infty$	0	$+\infty$	s_2	$+\infty$

Table 3: The solution to the endpoint visibility problem

5.1 The All Nearest Larger Values Problem

The All Nearest Larger Values problem (ANLV, for short) has been introduced in [4] where it is argued that ANLV is a fundamental problem of parallel processing, as a number of other problems reduce to it.[1] The ANLV problem can be formulated as follows: given a sequence of n real numbers a_1, a_2, ..., a_n, for each a_i ($1 \leq i \leq n$), find the nearest element to its left and the nearest element to its right (if any) that is larger than a_i.

The purpose of this subsection is to exhibit a time-optimal solution for the ANLV problem on a mesh with multiple broadcasting of size $n \times n$. We begin by proving an $\Omega(\log n)$ time lower bound for this problem on both the CREW-PRAM and the mesh with multiple broadcasting. To this end, we reduce the OR problem to ANLV. Suppose that $b_1, b_2, \ldots, b_n$ is an arbitrary input to OR. Construct an instance of ANLV by setting for every i ($1 \leq i \leq n$), $a_i = b_i$. To complete the construction, let a_0=1 and a_{n+1}=0. Now, the answer to the OR problem is 0 if, and only if, the nearest larger value for a_{n+1} is a_0. The conclusion follows by Proposition 3.1. To summarize our discussion we state the following result.

Lemma 5.1.1. The task of solving an instance of size n of the ANLV problem has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □

Now Lemma 5.1.1 and Proposition 3.2 combined, imply the following result.

Corollary 5.1.2. The task of solving an instance of size n of the ANLV problem has a time lower bound of $\Omega(\log n)$ on a mesh with multiple broadcasting of size $n \times n$. □

Next, we show that the lower bound of Corollary 5.1.2 is tight by demonstrating that an arbitrary instance of size n of the ANLV can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Given an arbitrary sequence of real numbers $a_1, a_2, \ldots, a_n$ as input, we associate with every a_i a vertical line segment s_i with endpoints $(i, -\infty)$ and (i, a_i). We also

[1] Actually, in [4] a variant termed the All Nearest Smaller Values is solved.

assume that the viewpoint ω lies at $(-\infty, 0)$. It is easy to confirm that the resulting collection S of vertical line segments is well ordered, and so we can apply the EV algorithm discussed in Section 4.

Clearly, for every endpoint (i, a_i) the solution corresponds to the nearest line segment that is blocking a horizontal ray emanating from (i, a_i) to the left and to the right. This, in turn, translates immediately into a solution to the ANLV, as desired. Consequently, we have the following result.

Theorem 5.1.3. An arbitrary instance of size n of the all nearest larger values problem stored in one row of a mesh with multiple broadcasting of size $n \times n$ can be solved in $O(\log n)$ time. Furthermore, this is time-optimal. □

The ANSV problem can be formulated as follows: given a sequence of n real numbers a_1, a_2, ..., a_n, for each a_i $(1 \leq i \leq n)$, find the nearest element to its left and the nearest element to its right (if any) that is smaller than a_i. It is easy to show that the ANSV problem has the same lower bound as the ANLV. Furthermore, with a minor modification our algorithm for the ANLV can be used to solve the ANSV problem time-optimally.

Theorem 5.1.4. An arbitrary instance of size n of the ANSV problem stored in one row of a mesh with multiple broadcasting of size $n \times n$ can be solved in $\mathrm{O}(\log n)$ time. Furthermore, this is time-optimal. □

5.2 The Triangulation Problem

The problem of triangulating a set of point in the plane finds applications to facility-location problems [40], clustering in pattern recognition [15], and interpolation problems in numerical analysis [43]. The problem is formally defined as follows. Given a set $S = \{p_1, p_2, \ldots, p_n\}$ of points in the plane, join all the points by non-intersecting line segments in such a way that every region internal to the convex hull of S is a triangle.

The main goal of this section is to propose a novel, time-optimal, solution to the triangulation problem on meshes with multiple broadcasting. We begin by showing that for both the CREW-PRAM and the mesh with multiple broadcasting, the task of triangulating a set of n points in the plane has a time lower bound of $\Omega(\log n)$. We then show that this lower bound is tight by exhibiting a matching algorithm for meshes with multiple broadcasting of size $n \times n$. Our algorithm for triangulation relies crucially on the visibility algorithm developed in the previous section.

The stated time lower bound can be derived by reducing the OR problem to triangulation. Let $b_1, b_2, \ldots, b_n$ be an arbitrary input to OR. We construct a set $\{p_0, p_1, \ldots, p_{n+1}\}$ of points in the plane by setting for every i $(1 \leq i \leq n)$, p_i=$(i, 0)$ if $b_i = 0$, and by setting p_i=$(i, 1)$ if $b_i = 1$. To complete the construction, we add the points p_0=$(0, 1)$ and $p_{n+1} = (n+1, 1)$. Now, the solution to the OR problem is 0 if, and only if, the segment $p_0 p_{n+1}$ belongs to the triangulation. The conclusion follows by Proposition 3.1.

Lemma 5.2.1. The problem of triangulating a set of n points in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □

Now Lemma 5.2.1 and Proposition 3.2 combined, imply the following result.

Corollary 5.2.2. The problem of triangulating a set of n points in the plane has a time lower bound of $\Omega(\log n)$ on a mesh with multiple broadcasting of size $n \times n$. □

A key ingredient in our triangulation algorithm for points in the plane is an algorithm for triangulating a restricted class of monotone polygons that we are about to define. Recall that a simple polygon is said to be monotone in the x-direction if its boundary can be decomposed

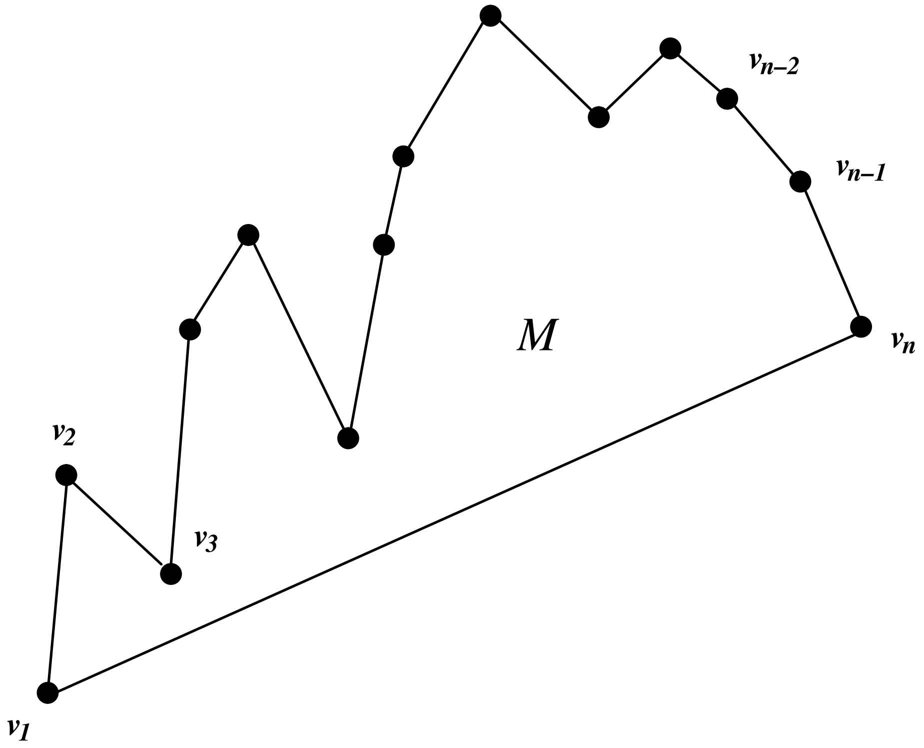

Figure 4: A special monotone polygon

into two chains, both monotone with respect to the x-axis. A monotone polygon is termed *special* if one of the monotone chains is a straight line joining the first and the last vertex on the other monotone chain. We shall refer to these chains as the *base edge* and the *monotone chain*, respectively. Refer to Figure 4 for an illustration. Let $M = v_1, v_2, \ldots, v_n$ be an n-vertex special monotone polygon with its vertices specified in clockwise order and with v_1v_n denoting the base edge. As usual, the vertices of the polygon are assumed to be stored in the first row of a mesh with multiple broadcasting of size $n \times n$, one vertex per processor. To simplify the exposition we assume that the monotone chain lies in the left halfplane determined by the directed line collinear with v_1v_n. We further subdivide the monotone chain into (sub)chains monotone in the y-direction. Such chains are termed *ascending* and *descending*. The details of the various steps involved in triangulating the special monotone polygon M are spelled out as follows:

Step 1. By checking its neighbors, every vertex v_i of M determines whether it belongs to an ascending or descending chain. Vertices achieving local minima in the y-direction are treated as part of both ascending and descending chains.

Step 2. With each vertex $v_i = (x_i, y_i)$ of M associate a vertical segment s_i with endpoints (x_i, y_i) and (x_i, ∞) and solve the resulting instance of the EV problem with the viewpoint ω at (-∞,0). In the notation of Section 4, we write t(v_i)=s_j and a(v_i)=s_k.

Comment: For a vertex v_i on an ascending (resp. descending) chain of M the vertex v_j is said to be a *match* if s_j is a solution obtained in Step 2 and v_j belongs to a descending (resp. ascending) chain.

Step 3. Every vertex v_i that has identified (at least) a match v_j adds the diagonal v_iv_j to the triangulation;

Comment: Note that no processor stores more than two segments added to the triangulation in Step 3.

Step 4. The following vertices mark themselves:

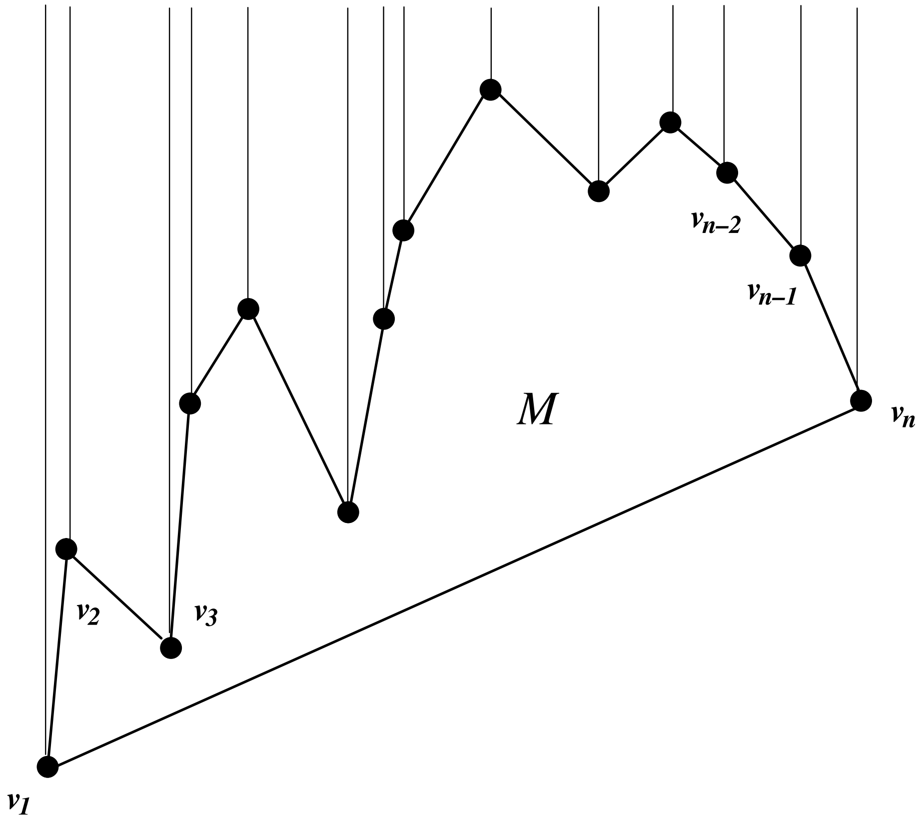

Figure 5: Vertical line segments associated with vertices in Step 2

- v_1 and v_n;
- vertices that have identified no match;
- vertices achieving local minima in the y-direction that have found only one match.

Comment: It is important to note that in case the base edge v_1v_n is horizontal, only v_1 and v_n are marked.

Step 5. Let $v_1 = v_{i_1}, v_{i_2}, \ldots, v_{i_r} = v_n$ be the list of marked vertices enumerated by increasing x-coordinate and let M' be the monotone polygon determined by these marked vertices. Rotate M' so that v_1v_n becomes parallel to the x-axis and repeat Steps 2–4.

Several of the steps of this algorithm are illustrated in Figures 5–8. For example, Figure 5 shows the vertical segments associated with the vertices in Step 2. Figure 6 shows the solution of the corresponding instance of the EV problem. Figure 7 shows all the diagonals added in Step 3 are drawn in full edges in Figure 5. Marked vertices are hashed. Notice that at the end of Step 4, the only part of the original polygon that is not triangulated is "bounded" by the marked vertices. Figure 8 shows the entire polygon triangulated. It is easy to see that after having rotated the edge v_1v_n, the solution $\mathrm{t}(v_{i_2})=s_n$, confirming that the diagonal $v_{i_2}v_n$ (i.e. v_3v_n) will be added to the triangulation.

The correctness and the time complexity of this algorithm are established by the following result.

Theorem 5.2.3. The problem of triangulating an n-vertex *special* monotone polygon can be solved in $O(\log n)$ time on an mesh with multiple broadcasting of size $n \times n$.

Proof. In order to show that the triangulation is done correctly, we need to prove that the

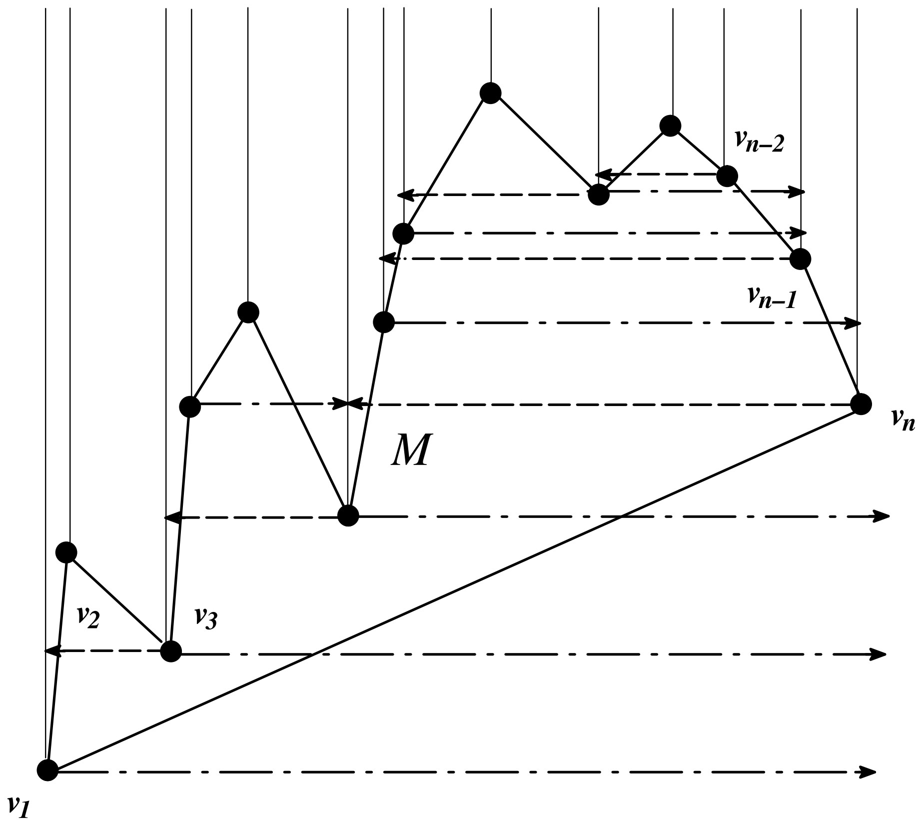

Figure 6: The solution to the corresponding endpoint visibility problem

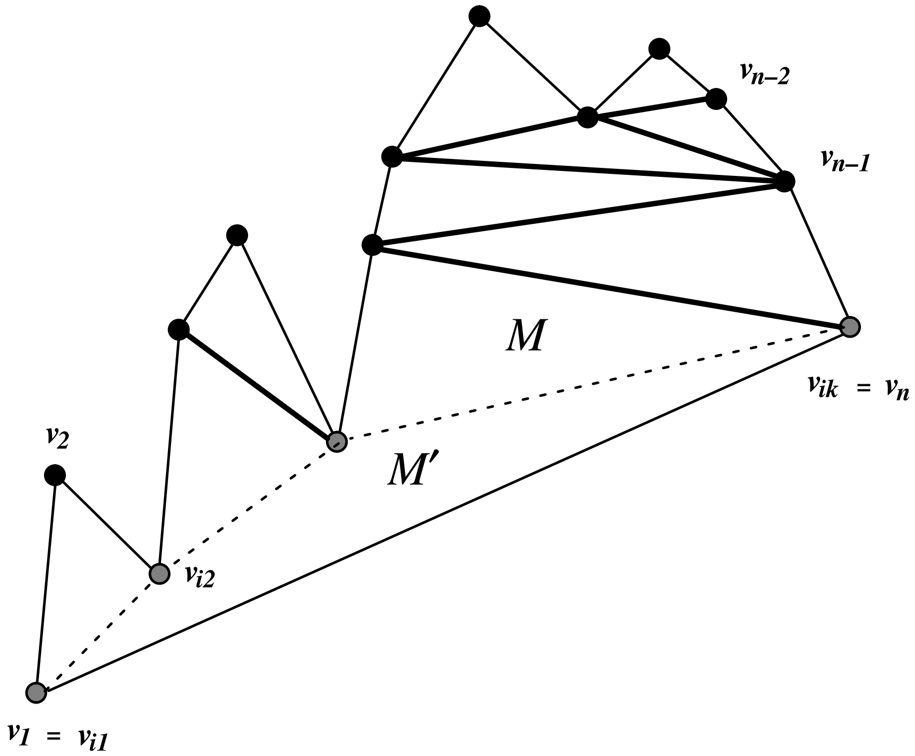

Figure 7: The state of the computation after Step 4

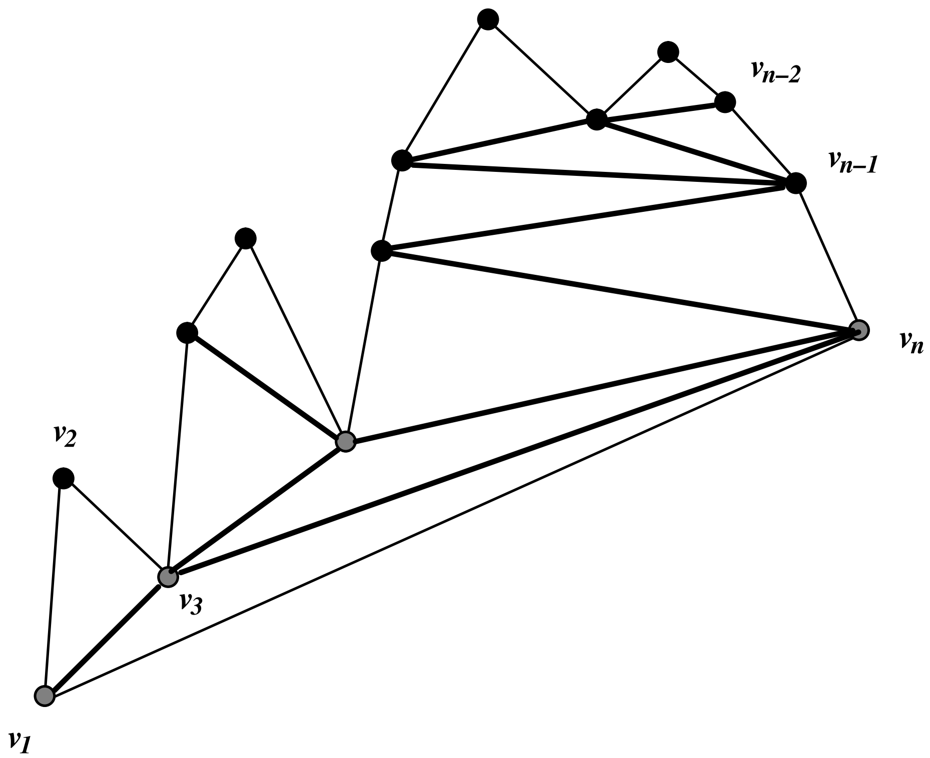

Figure 8: The completed triangulation

diagonals added in Step 3 do not intersect and that when the algorithm terminates there are no polygons with more than three sides left.

Let v_i belong to an ascending chain and let v_k be a match found in Step 2. By definition, v_k belongs to a descending chain and v_k has a lower y-coordinate than v_i. The diagonal $v_i v_k$ is added in Step 3. If some other diagonal $v_p v_q$, added in Step 3, intersects $v_i v_k$ then, exactly one of v_p and v_q lies on the monotone chain from v_i to v_k. Assume, without loss of generality, that v_p does. Now, two cases can occur, either the y-coordinate of v_p is lower than that of v_i implying that a(v_i)=s_p or the y-coordinate of v_p is higher than that of v_i, implying that t(v_p)=$s_{i'}$ and a(v_p)=$s_{k'}$, with $v_{i'}$ and $v_{k'}$ lying between v_i and v_k. Both these scenarios lead to a contradiction.

Let $v_1 = v_{i_1}, v_{i_2}, \ldots, v_{i_r} = v_n$ be the list of marked vertices obtained in Step 4, enumerated by increasing x-coordinate. Let A be the portion of the monotone chain between two adjacent marked vertices v_{i_j} and $v_{i_{j+1}}$. We claim that the interior of A is triangulated. The proof involves a simple counting argument. Let m be the total number of vertices between v_{i_j} and $v_{i_{j+1}}$. Let p be the number of local maxima in the y-direction in A; it follows that the number of local minima is $p-1$. Every vertex *internal* to A that is not a local maximum or a local minimum adds exactly one diagonal in Step 3. Further, vertices that are local maxima add no edges, while vertices that are local minima add two edges. Thus, the total number of edges added to A in Step 3 is $m - 2 - 2p + 1 \ + \ 2(p-1) \ = \ m - 3$. As shown before, these internal diagonals are non-intersecting, and so A is triangulated, as claimed.

Finally, let M' be the polygon determined by the marked vertices. To complete the proof we only need show that when the algorithm terminates M' is triangulated. It is clear that M' is monotone in the x-direction and that M' is special. As we are about to show, M' enjoys much stronger properties.

Observation 5.2.4. The polygon M' is monotone in both x and y directions.
(First, assume that v_1 has a lower y-coordinate than v_n. Now, if M' fails to be monotone in the y-direction, then there must exist two vertices v_{i_p}= (x_{i_p}, y_{i_p}) and v_{i_q}= (x_{i_q}, y_{i_q}) in M' such that $x_{i_p} < x_{i_q}$ and $y_{i_p} > y_{i_q}$. However, now we have reached a contradiction: both horizontal rays to the right and to the left originating at v_{i_p} must find a solution in Step 2 and so v_{i_p} cannot possibly be marked. The case where v_n has a lower y-coordinate than v_1 is similar.)

Observation 5.2.5. M' is monotone with respect to the direction of the edge $v_1 v_n$.
(Follows immediately from the definition of M' and Observation 5.2.4.)

Now, consider what happens when M' is rotated as to make the edge $v_1 v_n$ parallel to the x-axis. By Observations 5.2.4 and 5.2.5, M' is a special polygon monotone in the new x-direction. Therefore, after applying Steps 2–4 above, the only marked vertices of M' are v_1 and v_n and so, by the above argument, the triangulation of the original polygon M is complete. The correctness of the algorithm is thus proved.

Since each of the steps runs in O$(\log n)$ time, the problem of triangulating an n-vertex special monotone polygon is solved in O$(\log n)$ time. □

We are now in a position to discuss the problem of triangulating a given set S of n points in the plane. Begin by computing the convex hull of S and refer to Figure 9(a). By Proposition 3.6 this task can be performed in O$(\log n)$ time. Next, sort all the points in S by their x coordinates. By virtue of Proposition 3.4, this task can be performed in O$(\log n)$ time. Further, join every point with its immediate neighbor in the list sorted by x. All the convex hull edges and the edges drawn between two adjacent points are included in the triangulation. The chain determined by joining adjacent points in the sorted list divides the entire region within the hull into special monotone polygons in the sense previously described. (In Figure 9(b) these special monotone polygons are enumerated in left to right order as M_1, M_2, and M_3.) Each of these

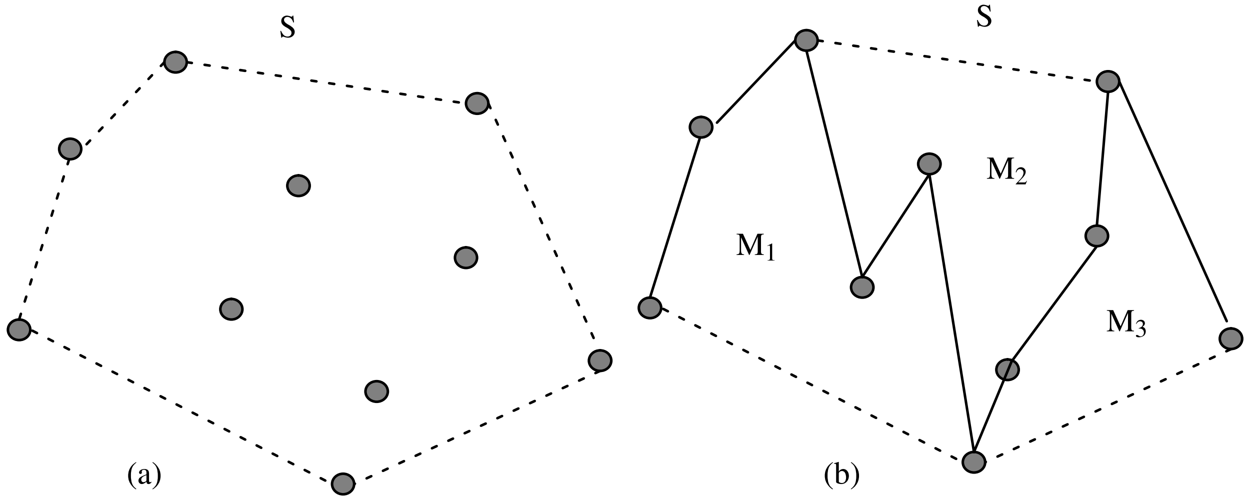

Figure 9: Triangulating a set of points in the plane

polygons with a base edge on the lower hull can be triangulated independently in parallel using the algorithm described above. The same can be repeated for the polygons with a base edge belonging to the upper hull. Now, Theorem 5.2.3 guarantees that each of the above steps can be performed in $O(\log n)$ time and thus the triangulation can be computed in $O(\log n)$ time. The time-optimality of the algorithm is guaranteed by Corollary 5.2.2. Therefore, we have the following result.

Theorem 5.2.6. The problem of triangulating a set S of n points in the plane can be done in $O(\log n)$ time on a mesh with multiple broadcasting. Furthermore, this is time-optimal. □

5.3 The Disk Visibility Problem

Given a collection $D = \{d_1, d_2 \ldots, d_n\}$ of n non-overlapping opaque disks and a viewpoint ω in the plane, the disk visibility problem (DV, for short) involves determining the portions of each disk that are visible to an observer positioned at ω. The DV problem finds applications to path planning in robotics where a mobile robot must navigate amidst a set of planar obstacles. It is customary to consider, in a first approximation, that all these obstacles are circular (i.e. disks). In this setup, the robot is shrunk to a point while the disks are augmented using Minkowski sums [21, 25], reducing the navigational problem to an instance of the DV problem.

The purpose of this subsection is to show that the SV problem discussed in Section 4 affords us a time-optimal algorithm for the DV problem. We begin by establishing an $\Omega(\log n)$ lower bound for the DV problem on the CREW-PRAM model by reducing OR to DV. Let b_1, b_2, ..., b_n be an arbitrary input to the OR problem. Now, consider any algorithm that correctly solves the DV problem with ω at $(-\infty, 0)$ and with input $d_1, d_2, \ldots, d_{n+1}$, where d_i $(1 \leq i \leq n)$ is the disk of unit radius, centered at $(i, -1)$ if $b_i = 0$, and centered at $(i, 1)$ if $b_i = 1$. To complete the construction, we add the disk d_{n+1} of unit radius centered at $(n + 1, 1)$. Our construction guarantees that the solution to OR is 0 if and only if d_{n+1} is visible from ω. The conclusion follows by Proposition 3.1. To summarize our discussion we state the following result.

Lemma 5.3.1. The task of solving the disk visibility problem for a collection of n disks in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. $\square$
Lemma 5.3.1 and Proposition 3.2 combined, imply the following result.
Corollary 5.3.2. The task of solving the disk visibility problem for a collection of n disks in the plane has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$. $\square$

To show that the time lower bound derived in Corollary 5.3.2 is tight, we now present an algorithm to solve an arbitrary instance of size n of the DV problem in O($\log n$) time on a mesh with multiple broadcasting of size $n \times n$. Assume that an arbitrary collection $D = \{d_1, d_2 \ldots, d_n\}$ of disks is stored, one disk per processor, in the first row of the mesh. As in the case of line segments we assume, for simplicity, that ω lies to the left of $\mathcal{D}$, that is, all the disks lie in the left halfplane determined by the vertical ray from ω to $-\infty$. Should this not be the case, the problem decomposes into two subproblems that are solved separately and whose solutions are combined directly.

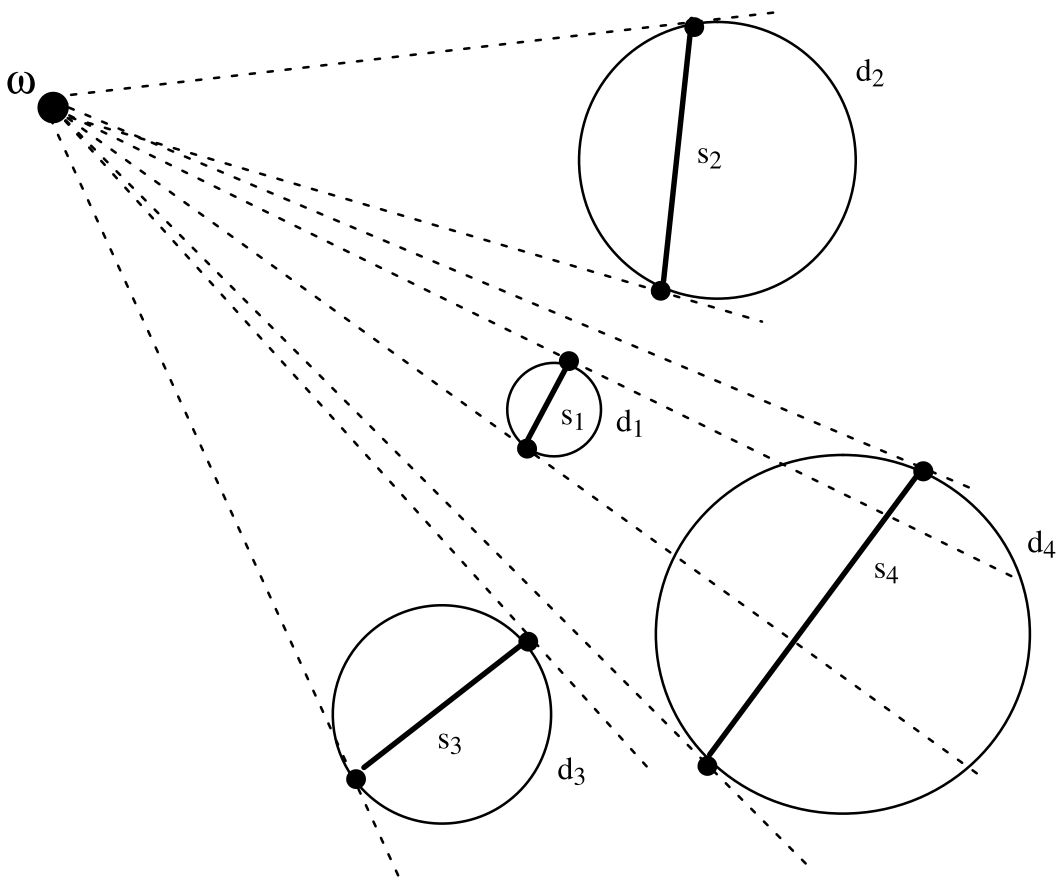

Figure 10: Illustrating the disk visibility problem

The algorithm begins by broadcasting ω to all the processors in the first row. Now, each processor holding a disk can determine the tangents to the disk from ω, as well as the length of these tangents, i.e. the distance between ω and the tangency points, in O(1) time.

With every disk d_i we associate the line segment s_i obtained by joining the corresponding tangency points. For an illustration, refer to Figure 10. Next, we sort the s_i's by increasing distance of their endpoints to ω. By Proposition 3.4, this can be done in $O(\log n)$ time. Without loss of generality, let $S{=}s_1, s_2, \ldots, s_n$ be the set of these segments in sorted order. For further reference, we need the following technical result.

Lemma 5.3.3. S is well ordered.

Proof. Suppose not. This implies that there exist subscripts i, j with $i < j$ and some ray δ originating at ω that intersects s_j before intersecting s_i. Let d_i and d_j be the two corresponding disks and let δ_1 and δ_2 be the supporting rays to d_j from ω. We let a and b be the points where δ_1 and δ_2 meet d_j.

Consider the circle C centered at ω and of radius the common length $\mid \overline{\omega a} \mid = \mid \overline{\omega b} \mid$ of the segments ωa and ωb. Let A stand for the planar region defined as the intersection of C with the halfplane determined by the line collinear with a and b that does not contain ω. Let O_j be the center of d_j. Simple geometric considerations guarantee that A lies entirely within the triangle determined by a, b, and O_j, which in turn, lies completely within d_j.

Observe that the ray δ that intersects both s_i and s_j must lie in the wedge determined by δ_1 and δ_2. Since δ intersects s_j before s_i, it follows that at least one of the endpoints of s_i lies in A. This, however, contradicts the assumption that the disks do not intersect. □

Lemma 5.3.3 guarantees that we can apply the SV algorithm developed in Section 4 to the collection $S{=}s_1, s_2, \ldots, s_n$. Once the visible portions of the segments are determined, the portions of the disks visible from ω can be trivially computed in $O(1)$ time. Thus, we have the following result.

Theorem 5.3.4. The disk visibility problem for a set of n disks can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal. □

5.4 The Rectangle Visibility Problem

Given a collection $\mathcal{R} = \{R_1, R_2 \ldots, R_n\}$ of n iso-oriented, non-overlapping, opaque rectangles in the plane along with a viewpoint ω, the rectangle visibility problem (RV, for short) involves determining the portions of each rectangle that are visible to an observer positioned at ω. The rectangle visibility problem finds applications to computer graphics, digital geometry, collision avoidance, VLSI design, and image processing [34, 35, 40, 41].

The purpose of this subsection is to show that the SV problem discussed in Section 4 affords us a time-optimal algorithm for the RV problem. We begin by establishing an $\Omega(\log n)$ lower bound for the RV problem on the CREW-PRAM model by reducing the OR problem to RV. Let b_1, b_2, ..., b_n be an arbitrary input to the OR problem. Now consider any algorithm that correctly solves the instance of the RV problem with ω at $(-\infty, 0)$ and with input $R_1, R_2, \ldots, R_{n+1}$, where R_i $(1 \leq i \leq n)$ is the rectangle with top-left corner at $(i, 2)$ and bottom-right corner at $(i{+}0.5, 0)$ in case $b_i = 1$, and with top-left corner at $(i, 1)$ and bottom right corner at $(i{+}0.5, 0)$ otherwise. To complete the construction, we add the rectangle R_{n+1} with with top-left and bottom-right corners at $(n{+}1, 2)$ and $(n{+}1.5, 0)$. Our construction guarantees that the solution to OR is 0 if and only if R_{n+1} is visible from ω. The conclusion follows by Proposition 3.1. To summarize our discussion we state the following result.

Lemma 5.4.1. The task of solving the rectangle visibility problem for a collection of n iso-oriented rectangles in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □

Lemma 5.4.1 and Proposition 3.2 combined, imply the following result.

Corollary 5.4.2. The task of solving the rectangle visibility problem for a collection of n iso-oriented rectangles in the plane has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$. □

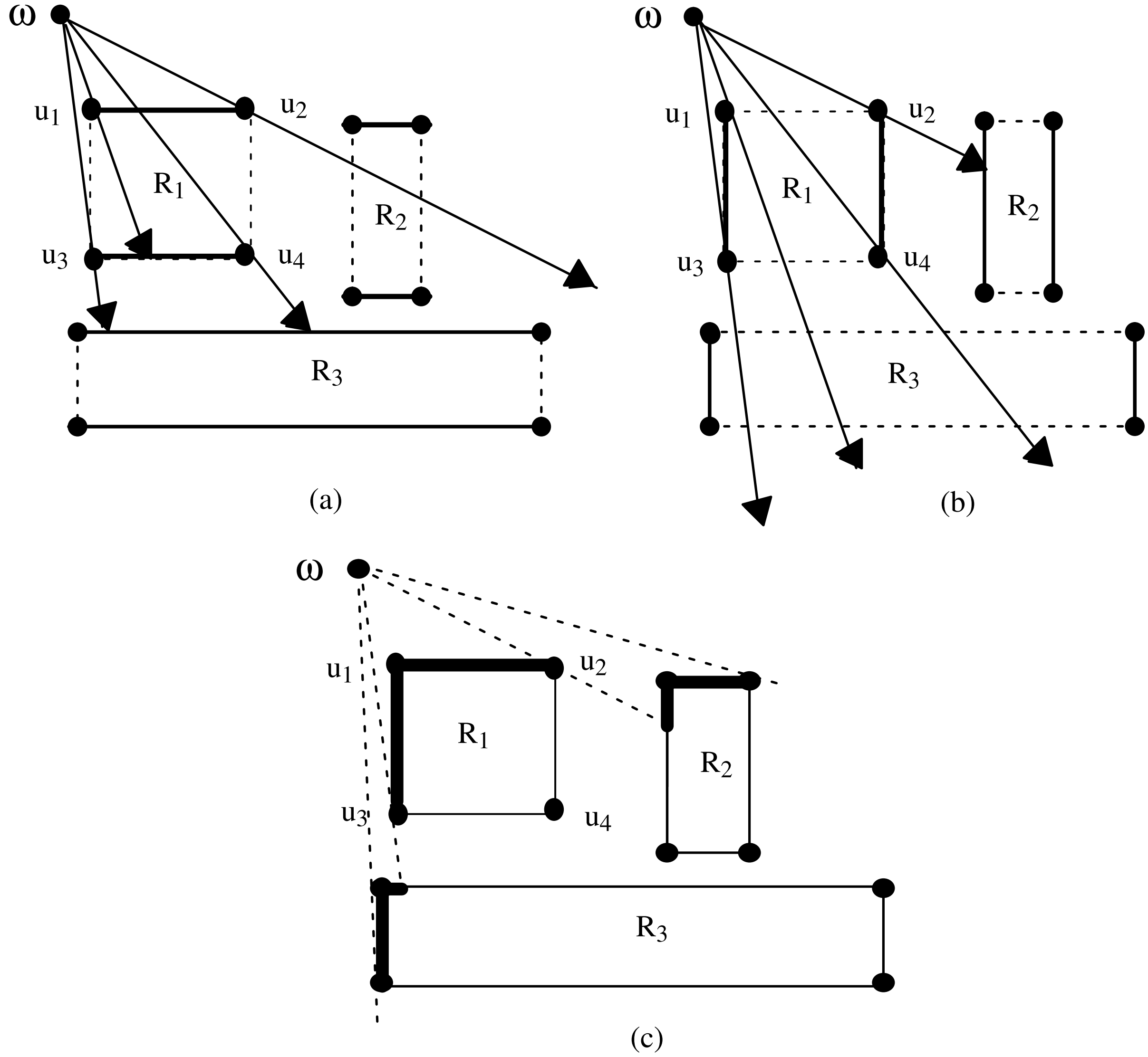

Figure 11: Illustrating the rectangle visibility problem

Our next goal is to show that the lower bound established above is tight. For this purpose, consider a collection $\mathcal{R} = \{R_1, R_2 \ldots, R_n\}$ of iso-oriented, non-overlapping, rectangles stored one per processor in the first row of a mesh with multiple broadcasting of size $n \times n$. We assume for simplicity that the viewpoint ω lies to the left of $\mathcal{R}$, i.e. that all the rectangles lie in the left halfplane determined by the vertical ray from ω to $-\infty$. Should this not be the case, the problem decomposes into two subproblems that are solved separately and whose solutions are combined directly. Our algorithm to solve the RV problem involves two stages that we describe next.

In the first stage, we solve the instance of the EV problem obtained by considering the top and bottom edges of every rectangle in $\mathcal{R}$. We begin by sorting these top and bottom edges by increasing y-coordinate. It is an easy observation that the sorted collection of these segments is well ordered and so the algorithm developed in Section 4 applies. In the second stage, the above process is repeated for the vertical segments of every rectangle. By virtue of Theorem 4.6, both these stages can be performed in $O(\log n)$ time.

At the end of the second stage, every generic corner e_i of rectangle r_i has four solutions: $a1(e_i)$, $t1(e_i)$, $a2(e_i)$, and $t2(e_i)$ obtained in the first and second stage, respectively. A corner e_i is *marked* if $t1(e_i)=t2(e_i)=0$. Now, every marked corner e_i combines the information stored in $a1(e_i)$ and $a2(e_i)$ by selecting, among them, the segment closer to e_i along the ray $e_i\overline{\varpi}$. If in the process e_i discovers that the closer of $a1(e_i)$ and $a2(e_i)$ is an edge that belongs to its own rectangle, then e_i becomes unmarked.

Finally, sorting the remaining marked corners by increasing polar angle, the task of computing the contour of the collection of rectangles proceeds exactly as its counterpart for line segments detailed in Section 4.

For an illustration, the reader is referred to Figure 11. For every rectangle R_i $(1 \leq i \leq 3)$, we let t_i, b_i, l_i, and r_i stand for the top, bottom, left, and right edges of R_i, respectively. Stage 1 is depicted in Figure 11(a). At the end of this stage, the solutions corresponding to the corners of R_1 are as follows: $a1(u_1)=b_1$, $a1(u_2)=+\infty$, $a1(u_3)=t_3$, $a1(u_4)=t_3$, $t1(u_1)=0$, $t1(u_2)=0$, $t1(u_3)=0$, and $t1(u_4)=t_1$.

Stage 2 is depicted in Figure 11(b). At the end of this stage, the solutions corresponding to the corners of R_1 are as follows: $a2(u_1)=+\infty$, $a2(u_2)=l_2$, $a2(u_3)=+\infty$, $a2(u_4)=+\infty$, $t2(u_1)=0$, $t2(u_2)=0$, $t2(u_3)=0$, and $t2(u_4)=0$.

After Stage 2, only the corners u_1, u_2, and u_3 are active. Of these, u_1 detects that the closer segment along the ray $u_1\overline{\varpi}$ is b_1, and so u_1 becomes inactive. The resulting contour is featured in Figure 11(c). To summarize our findings we state the following result.

Theorem 5.4.3. An arbitrary instance of size n of the rectangle visibility problem can be solved in $O(\log n)$ time on a mesh with multiple broadcasting. Furthermore, this is time-optimal.□

5.5 The Visibility Pair and Visibility Graph Problems

The compaction step of integrated circuit design motivates associating several kinds of graphs with a collection of non-overlapping rectangles in the plane. These graphs are intended to capture various visibility relations amongst the rectangles in the collection. Typically, the compaction process is based on the notion of *constraint graph*. Each vertex of the constraint graph corresponds to a rectangle or to a group of electrically equivalent elements; the edges of the constraint graph correspond to the design constraints that must be satisfied by the circuit [27, 38, 44].

However, in general, the constraint graph is rather complicated, with many extraneous edges [44]. From a computational standpoint [25], one is motivated to investigate properties of simpler graphs that provide useful heuristic solutions to a large number of special cases of compaction. An important such graph is the *visibility graph* (VG, for short) studied by several workers [25, 26, 38]. Consider a collection of iso-oriented, non-overlapping, rectangles in the plane. Two rectangles R and R' are said to be *visible* if there exists a horizontal line intersecting R and R' and not intersecting any other rectangle that lies between R and R'. The visibility graph has the set of rectangles as its vertices, with two vertices joined by an edge if and only if the corresponding rectangles are visible. To compute the visibility graph of a collection of rectangles, it is customary to shrink every rectangle to a vertical line segment. Since we are interested in the visibility in the x-direction only, no information is lost, as the visibility among rectangles is identical to the visibility among the resulting collection of vertical segments [24].

Our solution to the VG problem relies on the solution of the visibility pair problem (VP, for short) that we define next. Let $\mathcal{S}=\{s_1, s_2, \ldots, s_n\}$ be a collection of vertical segments in the

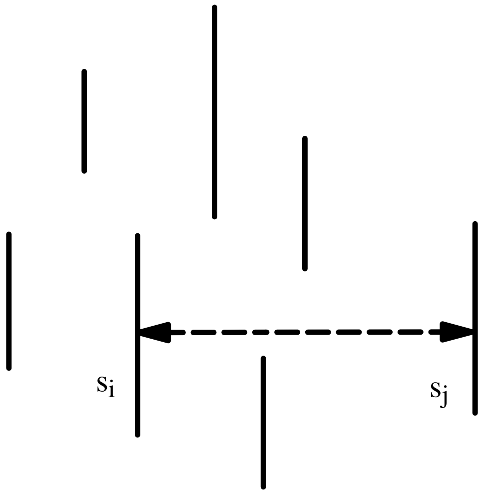

Figure 12: Illustrating visibility pairs

plane, sorted by their x-coordinate. Two segments $s_i, s_j \in S$, with $i < j$, are said to constitute a *visibility pair* (s_i, s_j) if there exists a horizontal line intersecting s_i and s_j but not intersecting any other segment s_k, with $i < k < j$ (see Figure 12). The visibility pair problem involves computing all visibility pairs in S. Recently, Lodi and Pagli [24] proposed an algorithm for the VP problem running in $O(\log n)$ time on a mesh of trees with $2n \times n$ leaves.

The purpose of this subsection is to offer a time-optimal solution to the VP problem on meshes with multiple broadcasting. We begin by showing that the task of solving an n-segment instance of the VP problem has a time lower bound of $\Omega(\log n)$ on both the CREW-PRAM and the mesh with multiple broadcasting. This is done by reducing the OR problem to VP. Let $b_1, b_2, \ldots, b_n$ be an arbitrary input to the OR problem. Based on this sequence we construct the input $\mathcal{S}= \{s_0, s_1, \ldots, s_{n+1}\}$ for the VP problem as follows:
• the endpoints of s_0 are $(0, 1.1)$ and $(0, 1.5)$;
• the endpoints for s_{n+1} are $(n+1, 1.1)$ and $(n+1, 1.5)$;
• if $b_i = 1$ then the endpoints of s_i are $(i, 0)$ and $(i, 2)$;
• if $b_i = 0$ then the endpoints of s_i are $(i, 0)$ and $(i, 1)$.
Clearly the construction can be performed in $O(1)$ time. It is easy to verify that the solution to the OR problem is 0 if, and only if, (s_0, s_{n+1}) is a visibility pair. The conclusion follows by Proposition 3.1. To summarize our discussion we state the following result.
Lemma 5.5.1. The task of solving the visibility pair problem for a collection of n vertical lines segments in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □
Lemma 5.5.1 and Proposition 3.2 combined, imply the following result.
Corollary 5.5.2. The task of solving the visibility pair problem for a collection of n vertical line segments in the plane has a time lower bound of $\Omega(\log n)$ on a mesh with multiple broadcasting of size $n \times n$. □
It is easy to see that the VG problem reduces easily to the VP problem. Consequently, we have the following result.
Corollary 5.5.3. The task of constructing the visibility graph of a collection of n rectangles in the plane has a time lower bound of $\Omega(\log n)$ on a mesh with multiple broadcasting of size $n \times n$. □

Our next goal is to show that the lower bound derived in Corollaries 5.5.2 and 5.5.3 is tight.

For this purpose, assume that a collection $\mathcal{S}$ of n vertical line segments is stored in the first row of a mesh with multiple broadcasting of size $n \times n$, in the usual way. For simplicity we assume that the segments are in general position, with no two endpoints sharing the same y-coordinate. Every segment s_i is specified by its top and bottom endpoints t_i and b_i, respectively.

Suppose that the solution to the endpoint visibility problem with ω at $(-\infty, 0)$ provides the following left and right solutions for each segment s_i in $\mathcal{S}$: t(t_i)=s_p, a(t_i)=s_q, t(b_i)=s_r and a(b_i)=s_u. Next, the processor storing the segment s_i generates the following visibility pairs: (s_p, s_i), (s_i, s_q), (s_r, s_i), (s_i, s_u), (s_p, s_q), and (s_r, s_u). Clearly, in this process a visibility pair can be reported several times. Note, however, that the total number of pairs generated is at most $6n$.

To eliminate duplicates, we sort all the pairs (s_i, s_j) by s_i and then by s_j. As a result, all the duplicates corresponding to a visibility pair occur consecutively. In every such group, mark the first pair. Now collecting the marked pairs can be easily reduced to sorting. It is easy to see that the above steps can be performed in $O(\log n)$ time. The correctness of the algorithm and its time-optimality are established by the following result.

Theorem 5.5.4. Given a collection $\mathcal{S}$ of n vertical segments, the corresponding instance of the visibility pair problem can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal.

Proof. The time-optimality follows directly from Corollary 5.5.2. We claim that every visibility pair in the set $\mathcal{S}$ is reported by our algorithm. Suppose that there exists a visibility pair (s_i, s_j) which is not reported. By definition, there exists at least one horizontal line which intersects both s_i and s_j and does not intersect any segment s_k, with $i < k < j$. Of all such horizontal lines, consider the line l with the maximum y-coordinate. Note that l must pass through some endpoint of a segment s_t in $\mathcal{S}$. Since the algorithm computes the visibility for all endpoints, it must have reported the visibility for the endpoints of s_t. Thus the pair (s_i, s_j) must have been reported. Similarly, it is easy to verify that it is not possible for the algorithm to report a pair (s_i, s_j) which is not a visibility pair. □

Once the set of visibility pairs in S are computed, the visibility graph can be constructed directly. Consequently, we state the following result.

Theorem 5.5.5. Given a collection $\mathcal{R}$ of n iso-oriented non-overlapping rectangles in the plane, the visibility graph of $\mathcal{R}$ can be constructed in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal. □

5.6 The Dominance Graph Problem

Consider a collection $\mathcal{R}= \{R_1, R_2, \ldots, R_n\}$ of n non-overlapping iso-oriented rectangles in the plane. We say that a rectangle R_i is *above* rectangle R_j if there are points in R_i and R_j sharing the same x-coordinate, with the points in R_i having larger y-coordinates. A rectangle R_i is *directly above* R_j if R_i is above R_j and no rectangle R_k is such that R_i is above R_k and R_k is above R_j. The *dominance graph* of the collection $\mathcal{R}$ is a directed graph $\vec{D}$ whose vertices correspond to the rectangles in $\mathcal{R}$ with two vertices u and v in $\vec{D}$ linked by a directed edge (u, v) whenever the rectangle corresponding to v is directly above the rectangle corresponding to u (see Figure 13). The dominance graph problem (DG, for short), involves computing the dominance graph of a given set of non-overlapping rectangles in the plane. The dominance graph is a subgraph of the visibility graph, discussed in the previous subsection, that happens to be of practical relevance, being of import to the compaction process in the y-direction.

The purpose of this subsection is to offer a time-optimal solution to the DG problem on meshes with multiple broadcasting. We begin by establishing a time lower bound of DG holding

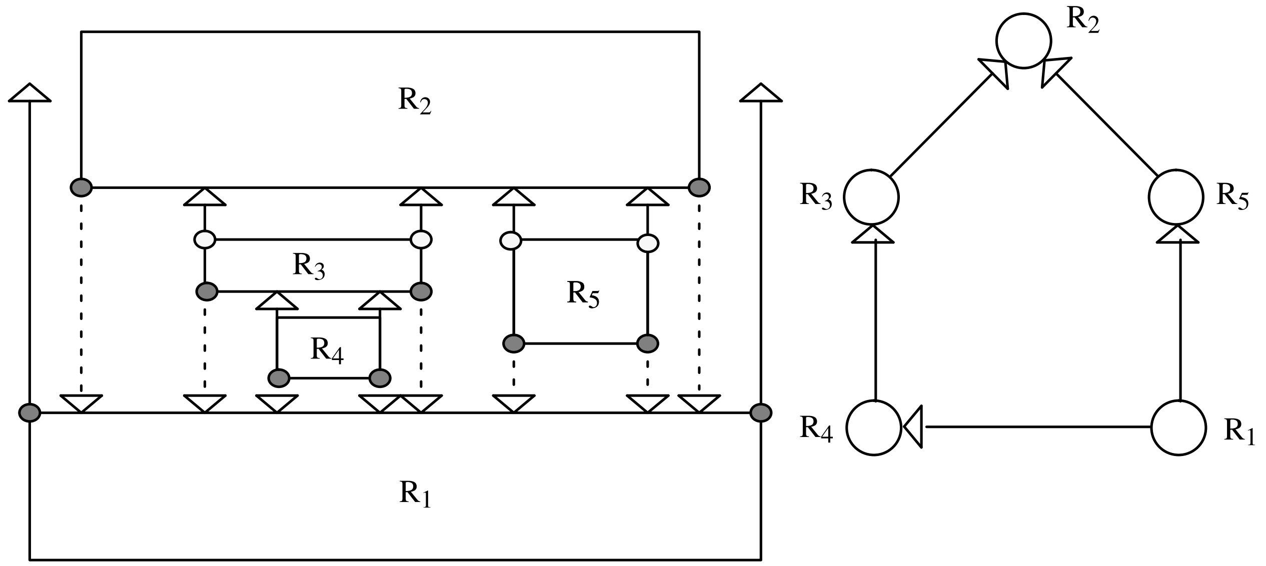

Figure 13: A collection of rectangles and its dominance graph

on both the CREW-PRAM and the mesh with multiple broadcasting. As usual, this is done by reducing the OR problem to DG. Let $b_1, b_2, \ldots, b_n$ be an arbitrary input to the OR problem. Based on this sequence, we construct an instance $\mathcal{R} = \{R_0, R_1, \ldots, R_n\}$ of the DG problem as follows:

- the bottom-left and the top-right corners of R_0 are $(0, -1)$ and $(n, -0.75)$;
- if $b_i = 0$, then the bottom-left and the top-right corners of R_i are $(n + i - 0.75, 0)$ and $(n + i - 0.25, 1)$;
- if $b_i = 1$, then the bottom-left and the top-right corners of R_i are $(i - 0.75, 0)$ and $(i - 0.25, 1)$.

Clearly this construction takes O(1) time. It is easy to verify that the solution to the OR problem is 0 if, and only if, the out-degree of the vertex corresponding to R_0 is 0.
The conclusion follows by Proposition 3.2. Thus, we state the following result.
Lemma 5.6.1. The task of computing the dominance graph of a collection of n non-overlapping iso-oriented rectangles in the plane has a time lower bound of $\Omega(\log n)$ on the CREW-PRAM, no matter how many processors and memory cells are used. □
Lemma 5.6.1 and Proposition 3.2 combined, imply the following result.
Corollary 5.6.2. The task of computing the dominance graph of a collection of n non-overlapping iso-oriented rectangles in the plane has a time lower bound of $\Omega(\log n)$ on the mesh with multiple broadcasting of size $n \times n$. □

Our next goal is to demonstrate that the time lower bound of Corollary 5.6.2 is tight by exhibiting a matching upper bound. To this end, consider an arbitrary instance of size n of the DG problem stored in the first row of a mesh with multiple broadcasting of size $n \times n$. We assume that the rectangles are specified by their bottom-left and top-right corners. It is easy to see that, in this setup, for every i $(1 \le i \le n)$, processor $P(1, i)$ can compute the top edge, h_i and the bottom edge, l_i of rectangle R_i in O(1) time.

As a preprocessing step the rectangles are sorted by the x-coordinate of their bottom left corners. For convenience, we shall continue to refer to the resulting sequence as $\mathcal{R} = \{R_1, R_2, \ldots, R_n\}$. For each rectangle R_i, i is said to be the identity of R_i.

Next, we solve the instance of the endpoint visibility problem consisting of the set of top and bottom edges of rectangles, with the viewpoint ω at $(0, -\infty)$. For each l_i, we are interested in computing segments visible in the negative y-direction. Similarly, for each h_i we are interested in computing the segments visible in the positive y-direction. By virtue of Theorem 4.6, this step can be performed in $O(\log n)$ time. With each endpoint we associate a 4-tuple (L, U, x, TB), whose semantics are as follows.

For each endpoint of a top segment, L is assigned the identity of its own rectangle and U is assigned the identity of the rectangle visible in the positive y-direction (-1 if undefined). Similarly, for each endpoint of a bottom segment, U is assigned the identity of its own rectangle and L is assigned the identity of the rectangle visible in the negative y-direction (-1 if undefined). In both cases, TB is a bit indicating whether the endpoint belongs to a top or bottom segment, and x is the x-coordinate of the endpoint.

Further, sort the set of tuples first by L and then by x. Clearly, this step requires $O(\log n)$ time. Now, consider the tuples (L_1, U_1, x_1, TB_1) and (L_2, U_2, x_2, TB_2) in adjacent processors. If $L_1 = L_2$ and $U_1 = U_2$ then we record an edge in $\vec{D}$, from the rectangle corresponding to L_1 to the rectangle corresponding to U_1. Each edge is stored as (L_1, U_1). After sorting the resulting ordered pairs, the dominance graph can be constructed trivially. This leads to the following result.

Theorem 5.6.3. Given a collection $\mathcal{R}$ of n rectangles as described above, the DG problem can be solved in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$. Furthermore, this is time-optimal.

Proof. In order to prove the correctness of this algorithm, it must be shown that the algorithm reports all directly above relations and no others.

Consider first the situation where R_i is directly above R_j. A number of cases occur. For illustration, let us consider the case where both bottom endpoints of R_i report R_j as visible. The proofs of all the other cases are similar.

Since both bottom endpoints report R_j as visible, both will set $U = i$ and $L = j$. Due to the assumption that R_i is directly above R_j, no other tuples can appear between these in the sorted list. Thus, the algorithm will report an edge in the dominance graph corresponding to these rectangles.

Next, consider the case where R_i is not directly above R_j. We distinguish between the following two cases.

(a) R_i is not above R_j. In this case R_i does not have any tuple containing the identity of R_j, so the edge between R_i and R_j cannot be reported.

(b) R_i is above R_j and there exists a rectangle R_k such that R_i is above R_k and R_k is above R_j. In this case the tuples containing information about R_i and R_j cannot occur consecutively. Again, the edge between R_i and R_j cannot be reported. Since each of the steps runs in $O(\log n)$ time, the time optimality follows. This completes the proof of the theorem. $\square$

6 Conclusions and Open Problems

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 recently been enhanced by the addition of various types of buses. Such a system, referred to as mesh with multiple broadcasting, has been adopted by the DAP family of computers [33] and involves enhancing the mesh architecture by the addition of row and column buses.

The contribution of this work is to offer the first known time-optimal solutions for a number of visibility-related problems on meshes with multiple broadcasting. All these problems are motivated by, and find applications to, fundamental tasks in computer graphics, robotics, and VLSI design. In generic form, the visibility problems that we have addressed can be stated as follows: given a collection of objects in the plane along with a viewpoint ω determine the portion of each object that is visible to an observer positioned at ω. Specifically, we have provided time-optimal solutions to this problem for several classes of objects, namely ordered line segments, disks, and iso-oriented rectangles in the plane. In addition, we have shown that our visibility algorithm for line segments leads directly to time-optimal solutions for the problems of determining the all nearest larger values for a given sequence, triangulating a set of points in the plane, determining the visibility pairs among a set of vertical line segments, and constructing the dominance and the visibility graphs of a set of iso-oriented rectangles in the plane. All the algorithms in this paper involve an input of size n and run in $O(\log n)$ time on a mesh with multiple broadcasting of size $n \times n$.

Other problems seem to be harder. First, we are working on extending these results to the three dimensional case with special emphasis on algorithms for hidden surface removal, ray tracing, and computing visibility between a collection of polyhedra.

Yet another way in which this work can be extended is to consider the smallest possible meshes on which the algorithms run time-optimally. Finding the smallest such mesh is a problem that has been open for some time.

Acknowledgement: The authors would like to thank Professor Jamieson for her timely and professional way of handling our submission.

References

[1] M. J. Atallah, R. Cole and M. T. Goodrich, Cascading divide-and-conquer: A technique for designing parallel algorithms, *SIAM Journal on Computing*, 18 (1989) 499–532.

[2] A. Bar-Noy and D. Peleg, Square meshes are not always optimal, *IEEE Transactions on Computers*, 40 (1991) 196–204.

[3] K. E. Batcher, Design of Massively Parallel Processor, *IEEE Transactions on Computers*, C-29, (1980), 836–840.

[4] O. Berkman, D. Breslauer, Z. Galil, B. Schieber, and U. Vishkin, Highly parallelizable problems, *Proceedings of Annual Symposium of Theory of Computing*, 1989, 770–780.

[5] D. Bhagavathi, H. Gurla, R. Lin, S. Olariu, J. L. Schwing and J. Zhang, Time- and VLSI-optimal sorting on meshes with multiple broadcasting, *Proc. International Conference on Parallel Processing*, St-Charles, Illinois, August 1993, III, 196–201.

[6] D. Bhagavathi, P. J. Looges, S. Olariu, J. L. Schwing, and J. Zhang, A fast selection algorithm on meshes with multiple broadcasting, *IEEE Transactions on Parallel and Distributed Systems*, 5, (1994) 772–778.

[7] D. Bhagavathi, S. Olariu, W. Shen, and L. Wilson, A time-optimal multiple search algorithm on enhanced meshes, with applications, *Journal of Parallel and Distributed Computing*, to appear.

[8] D. Bhagavathi, S. Olariu, J. L. Schwing, and J. Zhang, Convex polygon problems on meshes with multiple broadcasting, *Parallel Processing Letters*, 2 (1992) 249–256.

[9] D. Bhagavathi, S. Olariu, W. Shen, and L. Wilson, A unifying look at semigroup computations on meshes with multiple broadcasting, *Proc. Parallel Architectures and Languages Europe*, München, Germany, June 1993, LNCS 694, 561–569.

[10] D. Bhagavathi, V. Bokka, H. Gurla, S. Olariu, J. L. Schwing, and Zhang, Square meshes are not optimal for convex hull computation, *Proc. of International Conference on Parallel Processing* St-Charles, Illinois, August 1993, III, 307–311.

[11] S. H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Transactions on Computers* 33, (1984) 133–139.

[12] I.-W. Chan, and D. K. Friesen, An optimal parallel algorithm for the vertical segment visibility reporting problem, *Proc. ICCI'91*, Lecture Notes in Computer Science Vol. 497, Springer-Verlag, 1991, 323–334.

[13] Y. C. Chen, W. T. Chen, G.-H. Chen and J. P. Sheu, Designing efficient parallel algorithms on mesh connected computers with multiple broadcasting, *IEEE Transactions on Parallel and Distributed Systems*, 1, (1990).

[14] S. A. Cook, C. Dwork, and R. Reischuk, Upper and lower time bounds for parallel random access machines without simultaneous writes, *SIAM Journal on Computing*, 15 (1986) 87–97.

[15] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley and Sons, New York, 1973.

[16] L. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics, principles and practice*, Second Edition, Addison-Wesley, Reading, MA, 1990.

[17] H. Freeman and G. Pieroni, Eds., *Computer architecture for spatially distributed data*, Springer-Verlag, Heidelberg, Berlin, 1985.

[18] J. JáJá, *An introduction to parallel algorithms,* Addison-Wesley, Reading, MA, 1992.

[19] V. Prasanna Kumar and C. S. Raghavendra, Array processor with multiple broadcasting, *Journal of Parallel and Distributed Computing,* 2 (1987) 173–190.

[20] V. Prasanna Kumar and D. I. Reisis, Image computations on meshes with multiple broadcast, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 11 (1989) 1194–1201.

[21] J.-P. Laumond, Obstacle growing in a non-polygonal world, *Information Processing Letters*, 25 (1987) 41–50.

[22] H. Li and M. Maresca, Polymorphic-torus network, *IEEE Transactions on Computers,* 38 (1989) 1345–1351.

[23] R. Lin, S. Olariu, J. L. Schwing, and J. Zhang, Simulating enhanced meshes, with applications, *Parallel Processing Letters*, 3 (1993) 59–70.

[24] E. Lodi and L. Pagli, A VLSI solution to the vertical segment visibility problem, *IEEE Transactions on Computers*, 35 (1986) 923–928.

[25] T. Lozano-Perez, Spatial planning: a configurational space approach, *IEEE Transactions on Computers*, 32, (1983) 108–119.

[26] F. Luccio, S. Mazzone, and C. K. Wong, A note on visibility graphs, *Discrete Mathematics*, 64, (1987) 209–219.

[27] A. A. Malik, An efficient algorithm for generation of constraint graph for compaction, *Proc. International Conference on CAD*, 1987, 130–133.

[28] M. Maresca and H. Li, Connection autonomy and SIMD computers: a VLSI implementation, *Journal of Parallel and Distributed Computing*, 7 (1989) 302–320.

[29] C. A. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1979.

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

[31] S. Olariu and I. Stojmenović, Time-optimal proximity algorithms on meshes with multiple broadcasting, *Proc. 8th International Parallel Processing Symposium*, Cancun, Mexico, April 1994, 94–101.

[32] S. Olariu and I. Stojmenović, Time-optimal nearest-neighbor computations on enhanced meshes, *Proc. PARLE*, Patras, Greece, July 1994, to appear.

[33] D. Parkinson, D. J. Hunt, and K. S. MacQueen, The AMT DAP 500, 33^{rd} *IEEE Comp. Soc. International Conf.*, 1988, 196–199.

[34] T. Pavlidis, *Computer Graphics*, Computer Science Press, Potomac, MD, 1978.

[35] B. T. Preas and M. J. Lorenzetti (Eds.) *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, 1988.

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

[37] J. Rothstein, Bus automata, brains, and mental models, *IEEE Transactions on Systems Man Cybernetics* 18 (1988) 522–531.

[38] M. Schlag, F. Luccio, P. Maestrini, D. T. Lee, and C. K. Wong, A visibility problem in VLSI layout compaction, in F. P. Preparata (Ed.), *Advances in Computing Research*, Vol. 2 (1985) 259–282.

[39] H. S. Stone, *High-Performance Computer Architecture*, Second Edition, Addison-Wesley, Reading, MA, 1990.

[40] G. T. Toussaint, *Computational Geometry*, Elsevier Science Publishers, North-Holland, Amsterdam, 1985.

[41] G. T. Toussaint, Movable Separability of Sets, in [40].

[42] D. Vernon, *Machine vision, automated visual inspection and robot vision,* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[43] C. A. Wang and Y. H. Tsin, An O($\log n$) time parallel algorithm for triangulating a set of points in the plane, *Information Processing Letters*, 25 (1988) 55–60.

[44] W. H. Wolf and A. E. Dunlop, Symbolic layout and compaction, in [35], 211–281.