OPTIMAL VISIBILITY ALGORITHMS FOR BINARY IMAGES ON THE HYPERCUBE

FRANK DEHNE, QUOC T. PHAM, and IVAN STOJMENOVIC

SCS-TR-136
April 1988

OPTIMAL VISIBILITY ALGORITHMS FOR BINARY IMAGES ON THE HYPERCUBE

Frank Dehne *
Center for Parallel and Distributed Computing,
Carleton University, Ottawa, Canada K1S 5B6

Quoc T. Pham
Dept. 7H65,
Bell-Northern Research,
P.O. Box 3511, Ottawa, Canada K1Y 4H7

Ivan Stojmenovic
Department of Computer Science
University of Miami
P.O. Box 249085, Coral Gables, FL 33124, USA

ABSTRACT

Consider a nxn binary image. Given a direction D, the *parallel visibility* problem consists of determining for each pixel of the image the portion that is visible (i.e., not obstructed by any other black pixel of the image) in direction D from infinity. A related problem, referred to as *point visibility*, is to compute for each pixel the portion that is visible from a given point p.

In this paper, we derive 0(logn) time SIMD algorithms for each of these two problems on the hypercube where a processor is assigned to every pixel of the image. Since the worst case communication distance of two processors in a n^2-processor hypercube is 2 log n, it follows that both of the above algorithms are asymptotically optimal.

Keywords: Hypercube Algorithms, Image Processing, Visibility.

---

* Research supported by NSERC grant A9173

1. INTRODUCTION

A d-dimensional hypercube is a set of 2^d synchronized processing elements P(i), $0 \leq i \leq 2^d-1$, where two processors P(i) and P(j) are connected by a bidirectional communication link if and only if the binary representations of i and j differ in exactly one bit. (See [6], [8] and [10] for more details on hypercubes.)

In [4], [5], and [9], several hypercube algorithms are proposed for geometric and topological problems on digitized images: labeling of the extreme points, finding the diameter, finding the smallest enclosing box of a figure, component labeling, etc. In this paper, we continue this line of research by studying two visibility problems: the parallel visibility problem and the point visibility problem.

We consider a digitized image π consisting of $n \times n = 2^d$ pixels $\pi(i)$ indexed from 0 to 2^d-1 in row-major ordering and stored in the hypercube such that every pixel $\pi(i)$ is assigned to processor P(i). (If the pixels are numbered in Gray code ordering, a transformation can be performed in time O(log n); see [3], [7].)

For a given direction D, the *parallel visibility* problem for the image π is to compute for each pixel $\pi(i)$ the portion of $\pi(i)$ that is visible in direction D; i.e., the portion of $\pi(i)$ that is illuminated in the direction D by a light source at infinity, assuming that the black pixels of π obstruct light. The problem of computing the *visibility from a point* p is to compute for each pixel $\pi(i)$ the portion of $\pi(i)$ that is visible from p (i.e., the portion illuminated by a light source located at p). Figure 1 illustrates these two definitions.

[2] presented $O(\log^2 n)$ time algorithms to solve these two problems on the hypercube. In this paper we improve these results and present, for both problems, O(d) = O(logn) time algorithms. Since the worst case communication distance of two processors in a hypercube of size n^2 is 2 log n, it follows that both solutions are asymptotically optimal. The the remainder of this paper is organised as follows: Section 2 will present the algorithm for determining parallel visibility and in Section 3, we will solve the point visibility problem.

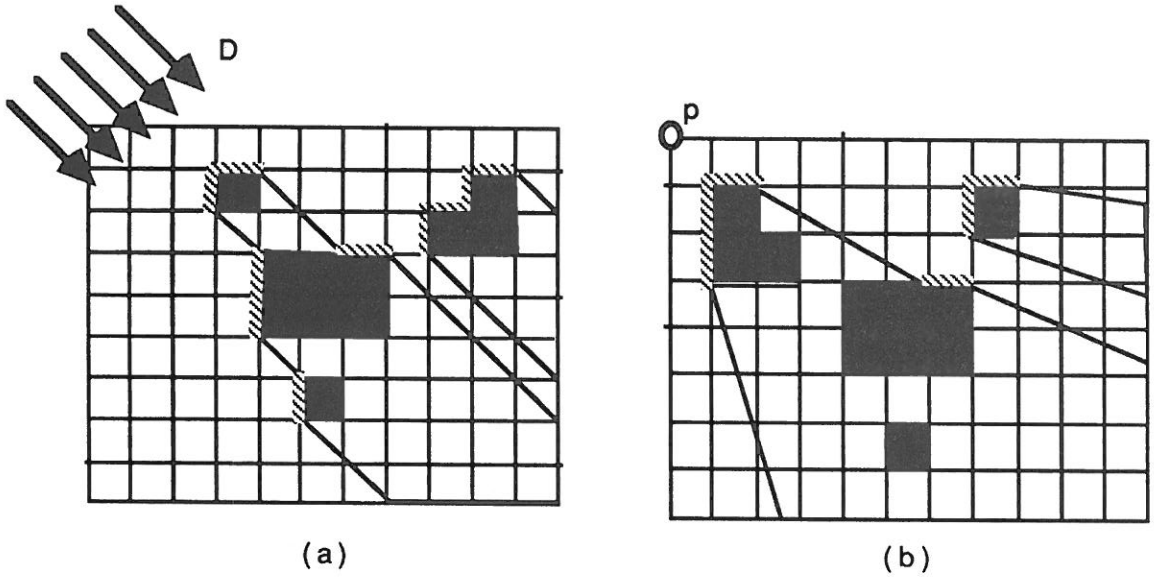

Figure 1: (a) Parallel Visibility in Direction D, (b) Visibility From Point p.

2. PARALLEL VISIBILITY

The basic geometric idea for the algorithm for parallel visibility is to divide the image π into strips parallel to direction D (as illustrated in Figure 2), where each strip is the portion of light going through the top edge of a pixel in the top row of the image (as if it were unobstructed by any black pixel). We shall assume without loss of generality that the angle β between the direction D and the North-South direction (in counter-clockwise direction) is between 0 and 45°; otherwise, the algorithm can be obtained similarly by symmetry.

Let w_0 and $w = \cos(\beta)w_0$ denote the width of a (square shaped) pixel and the width of a strip, respectively. Since $0 \le \beta \le 45^o$, $\frac{\sqrt{2}}{2} w_0 \le w \le w_0$; this yields [1]

Property 1:

1) No pixel $\pi(i)$ is properly contained in a strip (i.e., every $\pi(i)$ intersects either the left or right border of its strip) and

2) every pixel intersects at most two strips.

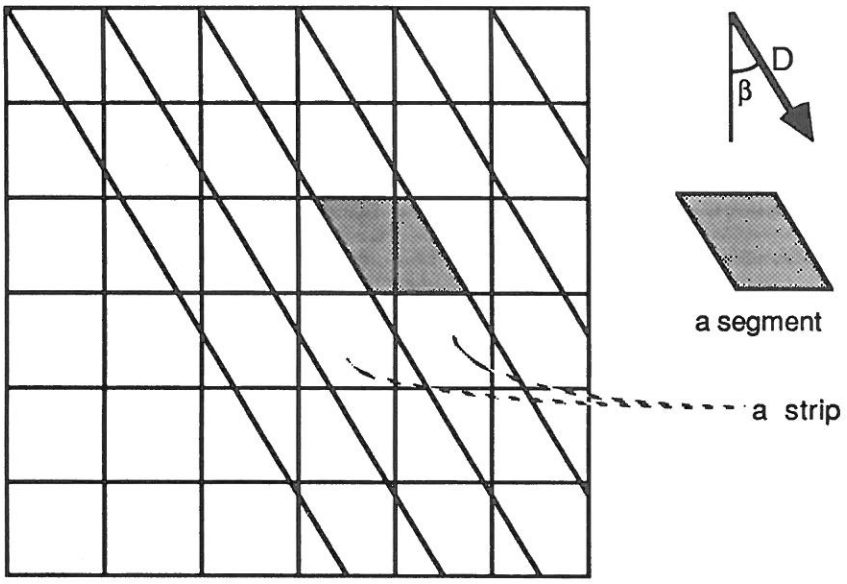

Figure 2: Dividing the Image into Strips and Segments

We further divide each strip into *segments,* where a segment is the portion of a strip contained in one row of pixels (see Figure 2). Since$0 \leq \beta \leq 45^{o}$, each segment intersects at most two neighboring pixels in the respective row; the leftmost of these will be referred to as the *representative* pixel of the segment.

For each segment, we define the *black interval* to be the projection of the black portion of the segment on the cross-section of the strip; the *white interval* is complement of the black interval (with respect to the cross-section of the strip). The projection of the visible portion of the segment on the cross-section of the strip will be referred to as *visible interval* of the segment.

The algorithm will solve the parallel visibility problem by computing, for all strips in parallel, the visible intervals for all segments in the strip. Note that, from Property 1, it follows that each white as well as visible interval consists of at most one connected component.

In the remainder of this section we will first show how to compute for each segment its white intervall and, then, how to compute all visible intervals in time O(log n).

We employ two data movement operations on a hypercube defined in [8]:

The first operation is the *distribute* operation. Assume that several processors P(i) of the hypecube stores a record r(i) and a processor destination address dest(i) such that, if i < j, then dest(i) < dest(j). The distribute operation consists of routing, for each of these P(i), the record r(i) to processor P(dest(i)). In [8], it is shown that this operation can be performed in O(logN) time on a hypercube with a total number of N processors.
The second operation, *concentrate* [8], is the reverse of the above operation. It can also be performed in O(logN) time.

A special case of the distribute operation is the *shift* operation defined as follows: Every processor P(i) representing a pixel $\pi(i)$ sends a record r(i) to the processor P(j) that represents the pixel $\pi(j)$ which is obtained by shifting $\pi(i)$ k pixels to the right (or to the left, upwards, downwards, respectively). The shift operation can be executed in time O(log n) by applying one (or, for cyclic shifts, two) distribution operation(s).

With this, all white intervals can be computed as follows: every processor determines in O(1) time which segment in which strip it represents; then, using the shift operation, every processor examines its local neighborhood and computes the white interval for the segment it represents in O(logN) time.

The geometric idea for computing all visible intervals is the following: Consider, within each strip, the sorted ordering of the segments with respect to direction D such that the topmost segment is the first in this ordering; the visible interval of each segment is the intersection of the white intervals of its predecessors (the visible interval of the topmost segment is the entire cross-section of the strip).

Therefore, for each strip, the problem of computing all visible intervals is a particular instance of the partial sum problem:

Given N values $a_0,\ldots,a_{N-1}$ and an associative binary operator *, compute a_0, $a_0{*}a_1$, $a_0{*}a_1{*}a_2$, ..., $a_0{*}a_1{*}a_2{*}\ldots{*}a_{N-1}$.

If each value a_i is stored in processor P(i) of an N-processor hypercube, then the partial sum problem can be solved in time O(logN) as follows (see also [12]):

Consider the problem of computing at each P(i) the partial sum $ps(i) = a_0{*}a_1{*}a_2{*}\ldots{*}a_i$ and the total sum $ts(i) = a_0{*}a_1{*}a_2{*}\ldots{*}a_{N-1}$

(1) Recursively, solve the problem on the sub-hypercube $P(0),\ldots,P(\frac{N}{2} - 1)$ for $a_0, a_1, a_2, \ldots, a_{N/2-1}$ and, in parallel, on the sub-hypercube $P(\frac{N}{2}),\ldots,P(N-1)$ for $a_{N/2}, a_{N/2+1}, a_{N/2+2}, \ldots, a_{N-1}$. At each P(i), ps(i) and ts(i) store the partial sum and total sum, respectively, for the subproblems.

(2) Each processor P(i) executes the following steps:
- IF $i<\frac{N}{2}$ THEN send ts(i) to $P(i+\frac{N}{2})$ ELSE send ts(i) to $P(i-\frac{N}{2})$.
- Receive value ts(j).
- $ts(i) := ts(i) + ts(j)$
- IF $i>\frac{N}{2}$ THEN $ps(i) := ps(i) + ts(j)$

To solve the parallel visibility problem, for each strip in parallel a partial sum problem has to be solved where the operands for the partial sum operation are the white intervals of the segments and the associative binary operator is set intersection. However, the above algorithm assumes that the operands for each partial sum problem are stored in exactly one sub-hypercube which, in general, is not the case for the white intervals of a strip.

We observe that the processors which store a row or column of pixels form a subcube. Therefore, our strategy is to move each of the strips into a column-subcube of the hypercube so that the partial sums, for all strips, can be computed independently in O(log n) time. Finally the obtained visibility information is returned to the original segment locations and propagated to the neighboring pixel in the segment.

In detail, the computation is as follows: (For simplicity we describe the algorithm only for the portion of the image shown in Figure 2; i.e., the region bounded by the top, right, and bottom edges of the image, and the line through the top left corner of the image parallel to direction D. The parallel visibility problem for the remaining pixels can be solved in a second, analogous step.)

1) All strips are "rotated" to the left to coincide with the columns as follows:
In each row of segments (i.e., the sub-hypercube of processors representing the segment in one row of pixels), all segments are shifted simultaneously to the left such that the leftmost sector is stored in the leftmost processor of the row. For each row-subcube this operation can be performed independently in time O(log n) using the shift operation described above.

2) In each column-subcube, which now stores the white intervals of the segments of one strip, the partial sum operation can be performed independently in O(log n) steps.

3) The inverse of Step 1 is performed to return the "rotated" (actually shifted) segments with the values obtained in the previous step back to their original positions.

4) After Step 3, each processor has received the visible interval for the segment it represents. Finally, for each segment, the visibility information is propagated from the representative pixels to the neighboring pixel in the segment (using a shift operation) and, for each pixel, the visible portion determined.

We obtain

Theorem 1:

The parallel visibility problem for a digitized image of size nxn can be solved on a d-dimensional hypercube, $2^d = nxn$, in time O(d)=O(log n).

3. POINT VISIBILITY

In order to determine the visibility from a point p, we will assume without loss of generality that the point p is located at the upper left corner of the image; otherwise, the digitized image can be split by the horizontal and the vertical lines through p into (at most) four quadrants and the problem can be solved for each quadrant separately.

In the remainder of this section we will show how to compute for all pixels in the area below the 45^o ray emanating from p (again, all angles are defined with respect to the north-south axis and in counter-clockwise direction) the portion that is visible from p. For all pixels above the ray, the visibility problem can be solved in a second analogous step.

Consider the 22.5^o ray emanating from p. It splits the image (below the 45^o ray) into two strips each of whose width (i.e., the horizontal distance between left and right border) increases with the distance from p and will, eventually, exceed width w_0 which ensured that no pixel is properly contained in a strip (cf. Property 1). At the level where the width of the rightmost strip reaches w_0, each strip is split again (by rays emanating from p) into two strips such that the angles between the borders of the four

strips are equal. When the width of the rightmost of these four strips reaches w_0, they are bisected again. This process is repeated until the entire image below the 45° ray is covered (see Figure 4). We define a *sector* to be the section of a strip between two consecutive splittings.

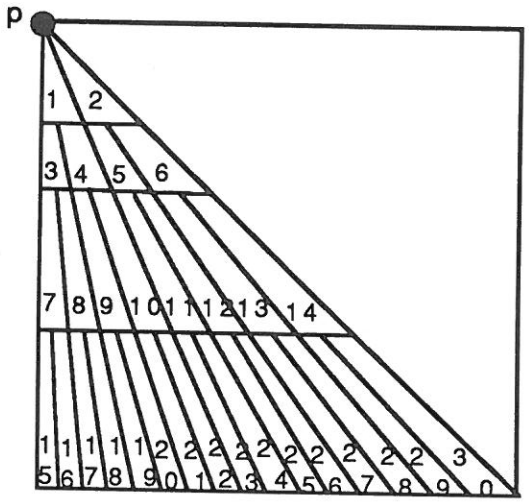

Figure 4: Subdividing the Image into Sectors

For every horizontal cut through the image, the width w_l of the leftmost sector intersected by the cut has the property

$$w_r > w_l = \frac{1 + \tan \alpha}{2} w_r \geq \frac{1}{2} w_r \quad ,$$

where w_r is the width of the rightmost sector (intersected by the cut) and α the angle between the left and right border of a sector($0° < \alpha \leq 22.5°$). Since $\frac{1}{2} w_0 \leq w_r \leq w_0$, and the width of all other sectors on the cut is between w_l and w_r, we obtain for the horizontal width w of any sector (along any horizontal cut)

$$\frac{1}{4} w_0 \leq w \leq w_0 \quad .$$

This yields

Property 2:

1) No pixel is properly contained in a sector (i.e., every pixel intersects either the left or right border of its sector) and

2) every pixel intersects at most four sectors.

Furthermore, we divide each sector into *segments*. A segment of a sector S is the portion of S contained in one row of pixels (see Figure 5).

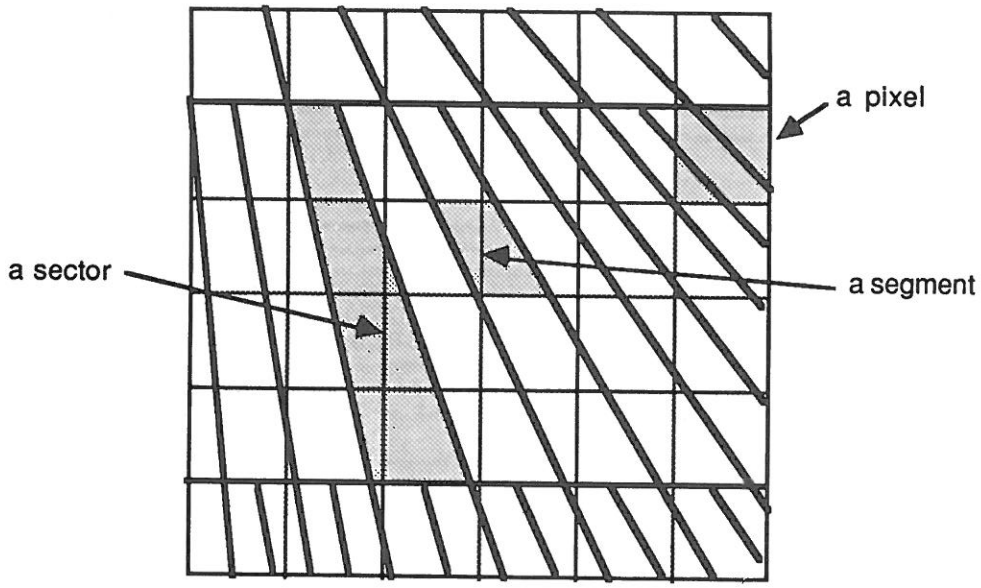

Figure 5: Sectors and Segments

For each segment, the representative pixel and the white interval are defined analogously to the parallel visibility problem.

The sectors, together with the relation "a black pixel in sector S_1 may obstruct part of a pixel of sector S_2", form a binary tree which we will refer to as *sector tree* (see Figure 6). The point visibility problem can be solved by executing for every path from the root to a leaf of the sector tree, for the white intervals of the segments of these sectors, the partial sum operation of Section 2.

Compared to Section 2, the problem arising here is not only that segments involved in a partial sum operation are (in general) not stored in a sub-hypercube but also that segments in non-leaf sectors are involved in several partial sum operations.

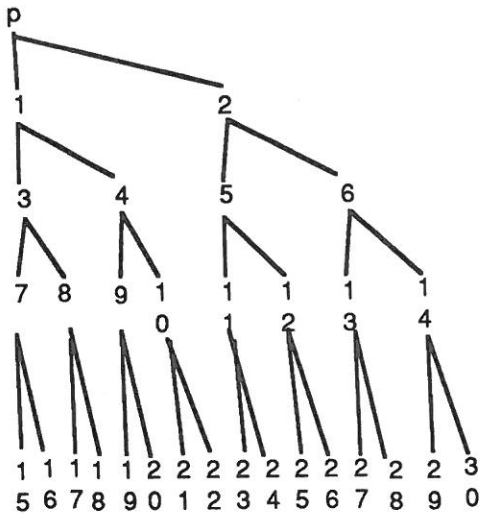

Figure 6: Sector Tree for the Subdivision in Figure 4

1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2
3	3	3	3	4	4	4	4	5	5	5	5	6	6	6	6
7	7	8	8	9	9	10	10	11	11	12	12	13	13	14	14
15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30

Figure 7: Copying Each Path of the Sector Tree Into a Column Subcube

The general idea to solve this problem is to copy, for every path from the root to a leaf of the sector tree, the segments of these sectors into a column sub-hypercube (see Figure 7).

In order to do this efficiently, we need to use the *generalize* operation of [8] which can be executed in time O(log N) on a N-processor hypercube:

> Every processor P(i), or any subset of processors, stores a record r(i) and a destination address dest(i) such that, if i<j, then dest(i)<dest(j); the result of the generalize operation is that every record r(i) is routed to the processors P(dest(i-1)+1), ..., P(dest(i)).

The point visibility problem can now be solve as follows:

1) Every processor P(i) determines, in constant time, which segment, *Seg(i)*, in which sector, *Sect(i)*, it represents (if any).

2) Since the sector tree is a complete binary tree, every P(i) can also determine in constant time the column number, *c(i)*, of the representative pixel in the bottom row of the rightmost leaf-sector of the sub sector tree rooted at Sect(i). If Sect(i) is a leaf-sector then c(i) is the column number of the representative pixel in the bottom row of Sect(i).

3) Every P(i) determines in time O(log n) the white interval, *w(i)*, of Seg(i); see Section 2.

4) A generalize operation is performed where, for every P(i) representing a segment Seg(i), r(i)=w(i) and dest(i) is the row major index, i^*, of the pixel with the same row number as π(i) and column number c(i). (It is easy to see that the requirement, i<j => dest(i)<dest(j), for generalize operations holds.)

5) After Step 4, for each path from the root to a leaf of the sector tree, the white intervalls of the segments of all sectors on the path are stored in a column subcube in sorted order.

Therefore, for all paths in parallel, the partial sum operation with respect to their white intervals can be computed in time O(log n).

6) The visible intervals obtained in Step 5 are returned from each P(i*) to P(i), the processor storing the representative pixel of the segment. (This can be implemented with time complexity O(log n) by using the concentrate operation. Note that, in Step 5, for all copies of the white interval of a segment the result of the partial sum operation is the same.)

Finally (by using the shift operation), for each segment, the visible interval is sent in time O(log n) from the represenттative pixel to the other pixel in its segment (if exists).

Summarizing, we obtain

Theorem 2:

The point visibility problem for a digitized image of size n x n can be solved on a d-dimensional hypercube, 2^d = n x n, in time O(d)=O(logn).

REFERENCES

[1] F.Dehne, A. Hassenclover, J.-R. Sack, and N. Santoro, Parallel Visibility on a Mesh-Connected Computer, in: Proc. Int. Conference on Parallel Processing and Applicatins, L'Aquila, 1987, pp. 173-180.

[2] F.Dehne, Q.T.Pham, Visibility ALgorithms for Binary Images on the Hypercube and the Perfect-Shuffle Computer, to appear in: Proc. of the International Federation for Information Processing WG 10.3 Working Conference on Parallel Processing, Pisa, 1988

[3] S.L. Johnson, Communication efficient basic linear algebra computaions on hypercube architectures, Journal on Parallel and Distributed Computing, 4, 1987, pp. 133-172

[4] R. Miller and S.E. Miller, Using Hypercube Multiprocessors to Determine Geometric Properties of Digitized Pictures, in: Proc. IEEE Conference on Parallel Processing, 1987, pp.638-640.

[5] R. Miller and Q.F. Stout, Some Graph and Image Processing Algorithms on the Hypercube, in: Proc. Second Conference on Hypercube Multiprocessors, 1986, pp.418-425.

[6] R. Miller and Q.F. Stout, Parallel Algorithms for Regular Architectures, MIT Press, to appear.

[7] I. Stojmenovic, Computational Geometry on the Hypercube, Technical Report, Computer Science Department, Washington State University, Pullman, WA, 1987

[8] D. Nassimi, S. Sahni, Data Broadcasting in SIMD Computers, IEEE Transactions on Computers, Vol. C-30, No. 2, 1981, pp.101-106.

[9] Q.T. Pham, Parallel Algorithm and Architecture for Binary Image Component Labeling, Technical Report, Deparment of Computer Science, University of California, Los Angeles, CA, 1987.

[10] C.L. Seitz, The Cosmic Cube, Comm. of the ACM, 28, 1985, pp. 22-33.

[11] R.E. Tarjan and U. Vishkin, Finding Biconnected Components and Computing Tree Functions in Logarithmic Parallel Time, in: Proc. 25th IEEE Symp. on Foundations of Computer Science, 1984, pp. 12-20.