OPTIMAL VISIBILITY ALGORITHMS FOR BINARY IMAGES ON THE HYPERCUBE

-preliminary version-

FRANK DEHNE

Center for Parallel and Distributed Computing, School of Computer Science, Carleton University, Ottawa, Canada K1S 5B6. Research partially supported by the Natural Sciences and Engineering Research Council of Canada under Grant A9173.

QUOC T. PHAM

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

IVAN STOJMENOVIC

Department of Computer Science, University of Ottawa, Ottawa, Canada K1N 9B4.

1. INTRODUCTION

Consider a n x n binary image π with pixels $\pi(i)$, $1 \le i \le n^2$, numbered in row-major ordering. 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 improve the results in [2] and derive 0(log n) time SIMD algorithms for these two problems on the hypercube, where processor P(i) is assigned to pixel $\pi(i)$, $1 \le i \le n^2$. Since the worst case communication distance of two processors in a n^2-processor hypercube is 2 log n, it follows that both algorithms are asymptotically optimal.

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, 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 North-South direction and the direction D (in counter-clockwise direction) is between 0 and 45^o; 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 $0 \le \beta \le 45^o$, $(\sqrt{2}/2)w_0 \le w \le w_0$; this yields **Property 1** ([1]): *No pixel $\pi(i)$ is properly contained in a strip (i.e., every $\pi(i)$ intersects either the left or the right border of its strip) and every pixel intersects at most two strips.*

We further divide each strip into *segments*, where a segment is the portion of a strip contained in one row of pixels. Since $0 \le \beta \le 45^o$, 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 the 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. 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 interval and, then, how to compute all visible intervals in time O(log n). We employ two O(log n) time data movement operations on a hypercube defined in [3]: *distribute* and *concentrate*. A special case of the distribute operation, which can be executed in the same time, is the *shift* operation where every processor P(i) sends a record to the processor P(i+k) in the same row. With this, all white intervals can be computed as follows: every processor determines in O(1) time the segment it represents and the corresponding strip number; then, using the shift operation, every processor examines its local neighborhood and computes the white interval for the segment it represents in O(log n) 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 (or parallel prefix) problem which can be solved on the hypercube in time O(log n).

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 partial sum 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 sub-hypercube. Therefore, our strategy is to move (using the shift operation) 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.

Theorem 1: *The parallel visibility problem for a digitized image of size nxn can be solved on a d-dimensional hypercube, $2^d = n \times n$, 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

F. Dehne, Q. T. Pham, and I. Stojmenovic, "Optimal visibility algorithms for binary images on the hypercube - preliminary version," in Proc. *Allerton Conference on Communication, Control and Computing*, Monticello, Ill., 1988, pp. 1035-1036.

generality that the point p is located at the upper left corner of the image; otherwise, the 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 whose widths (i.e., the horizontal distance between left and right border) increase 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^o ray is covered. We define a *sector* to be the section of a strip between two consecutive splittings. For every horizontal cut through the image, the width w_l of the leftmost sector intersected by the cut has the property $w_r > w_l = (1+\tan\alpha/2)w_r \geq (1/2)w_r$, 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 the sector ($0^o < \alpha \leq 22.5^o$). Since $(1/2)w_0 \leq w_r \leq w_0$, 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) $(1/4)w_0 \leq w \leq w_0$. This yields

Property 2: *No pixel is properly contained in a sector (i.e., every pixel intersects either the left or right border of its sector) and 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. For each segment, the *representative pixel* and the *white interval* are defined analogously to the parallel visibility problem. The sectors, together with the relation "<" defined by "*$S_1 < S_2$ if and only if the top horizontal border of S_2 is contained in the bottom horizontal border of S_1*", form a binary tree which we will refer to as *sector tree*. 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. 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 segmfor the solution of 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. In order to do this efficiently, we utilize the *generalize* operation of [3]. The point visibility problem can now be solved as follows (To simplify exposition, we shall assume that every processor representing a pixel simulates four 'virtual' processors, one for each of the at most four segments intersected by the pixel.):

(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 of the bottommost segment in 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 of the bottommost segment 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), the record to be sent is w(i) and the destination address is the row major index, *i**, of the pixel with the same row number as $\pi(i)$ and column number c(i). (It is easy to see that the requirement i<j => dest(i)<dest(j) for generalize operations holds; see [3].) (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 representative pixel to the other pixel in its segment (if exists).

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

REFERENCES

[1] F.Dehne, A. Hassenclover, J.-R. Sack, N. Santoro, Parallel Visibility on a Mesh-Connected Computer, in: Proc. Int. Conference on Parallel Processing and Applications, 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, in: Proc. of the International Federation for Information Processing WG 10.3 Working Conference on Parallel Processing, Pisa, 1988.

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