Image Processing On The OTIS-Mesh Optoelectronic Computer*

Chih-fang Wang and Sartaj Sahni
Department of Computer and Information Science and Engineering
University of Florida
Gainesville, FL 32611
{wang,sahni}@cise.ufl.edu

Abstract

We develop algorithms for histogramming, histogram modification, Hough transform, and image shrinking and expanding on an OTIS-Mesh optoelectronic computer. Our algorithm for the Hough transform is based upon a mesh algorithm for the Hough transform which is also developed in this paper. This new mesh algorithm improves upon the mesh Hough transform algorithms of [4] and [14].

Keywords and **phrases:** optoelectronic computer, OTIS-Mesh, image processing, histogramming, histogram modification, Hough transform, image shrinking and expanding.

1 Introduction

It is well known that optical interconnect has superior power, speed, and crosstalk properties compared to electronic interconnect when the interconnect distance is more then a few millimeters [5, 18]. With this knowledge in mind, Marsden *et al.* [26], Hendrick *et al.* [9], and Zane *et al.* [44] have proposed an architecture in which the processors are partitioned into groups. Within each group electronic interconnect is used to connect the processors. Optical interconnect is used to connect processors in different groups.

The optical transpose interconnection system (OTIS) was proposed by Marsden *et al.* [26], where processor i of group j is connected to processor j of group i via an optical connection. Krishnamoorthy *et al.* [19] have shown that when the number of groups equals the number of processors within a group, the bandwidth and power efficiency are maximized, and system area and volume minimized. As a result, an N^2 processor OTIS computer is partitioned into N groups of N processors each for our study. Figure 1 shows a 16 processor OTIS architecture. The processor indices are of the form (G, P) where G is the group index and P the processor index within the group.

*This work was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111.

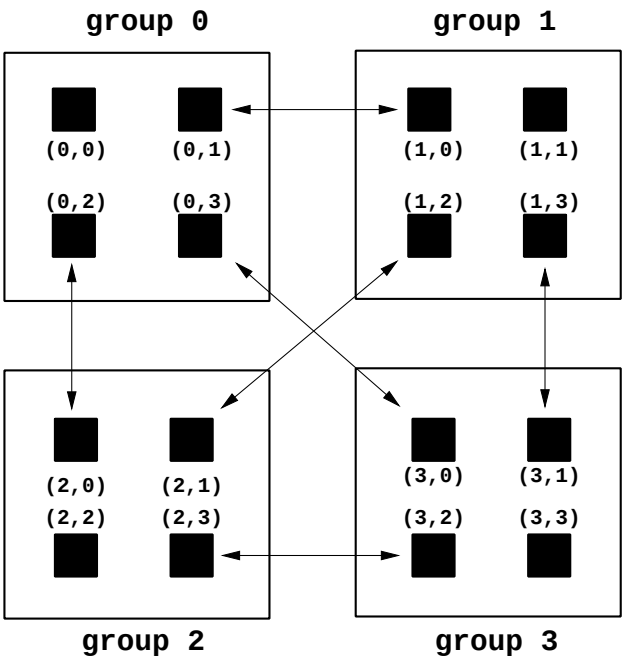

Figure 1: Example of OTIS connections with 16 processors

The OTIS-Mesh optoelectronic computer is a class of OTIS computers in which the electronic interconnect within each group follows the mesh paradigm [44, 35]. A 16 processor OTIS-Mesh computer is shown in Figure 2. The large boxes enclose groups of processors which are denoted by small boxes. The pair of numbers g, p inside a small box represents the group (g) and processor (p) indexes. Groups are numbered in row-major fashion, as they are laid out as a two-dimensional array. The pair (i, j) above a group denotes the row (i) and column (j) in which a group lies. Electronic links are indicated by arrows inside a group, while optical links are shown by arrows among groups. We shall refer to moves utilizing optical links as OTIS moves.

Zane *et al.* [44] have shown that the OTIS-Mesh computer can simulate a $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ four-dimensional mesh computer. Sahni and Wang [35] have developed algorithms for data routing algorithms. Wang and Sahni have studied algorithms for basic operations [41] such as prefix sum, consecutive sum, concentrate, sort, and they have also developed algorithms for various matrix multiplication operations [42].

In this paper, we focus on four problems from the image processing area. These problems are histogramming (Section 2), histogram modification (Section 3), image shrinking and expanding (Section 4), and Hough transform (Section 5).

As noted in [35], there are two plausible ways to map an $N \times N$ image onto an N^2 processor OTIS-Mesh – group row mapping (GRM), and group submesh mapping (GSM). In GRM, the

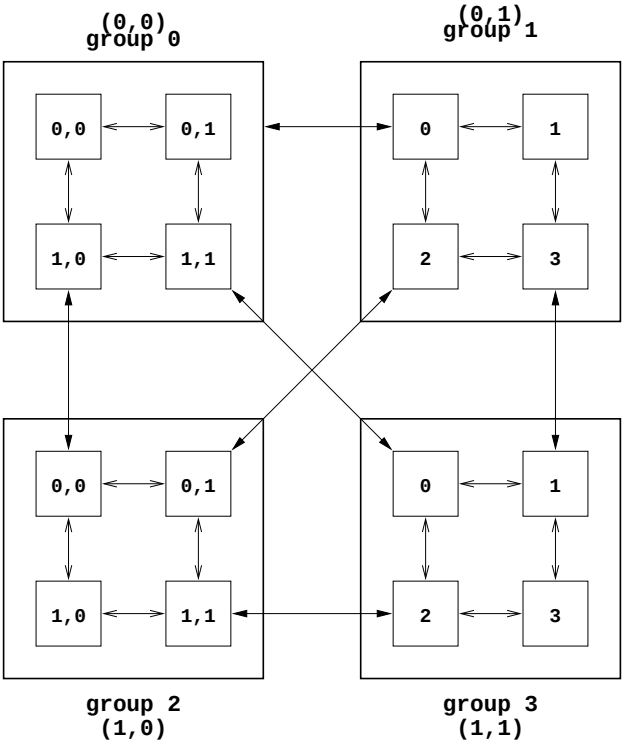

Figure 2: Example of OTIS-Mesh with 16 processors

N processors in each group are ordered in snake-like row-major fashion. Likewise, the group layout is also in snake-like row-major fashion. Row i of a two-dimensional image is mapped onto the processors in group i of the OTIS-Mesh. Figure 3(a) shows the GRM mapping of a 4×4 image onto a 16 processor OTIS-Mesh. The pair i, j in each processor square is the row and column index of the array entry.

In GSM, each group of the OTIS-Mesh denotes a subarray of size $\sqrt{N} \times \sqrt{N}$ of the $N \times N$ image. The four-dimensional index (g_x, g_y, p_x, p_y) denotes processor (p_x, p_y) of group (g_x, g_y). Figure 3(b) shows the GSM mapping of a 4×4 image onto a 16 processor OTIS-Mesh.

Our histogramming and histogram modification algorithms are insensitive to how the image is mapped onto the OTIS-Mesh. Therefore, these algorithms are developed without regard to the mapping used. The algorithms for Hough transform and image shrinking and expanding depend on the particular mapping used. In Sections 4 and 5, we develop algorithms for both the GRM and GSM mappings.

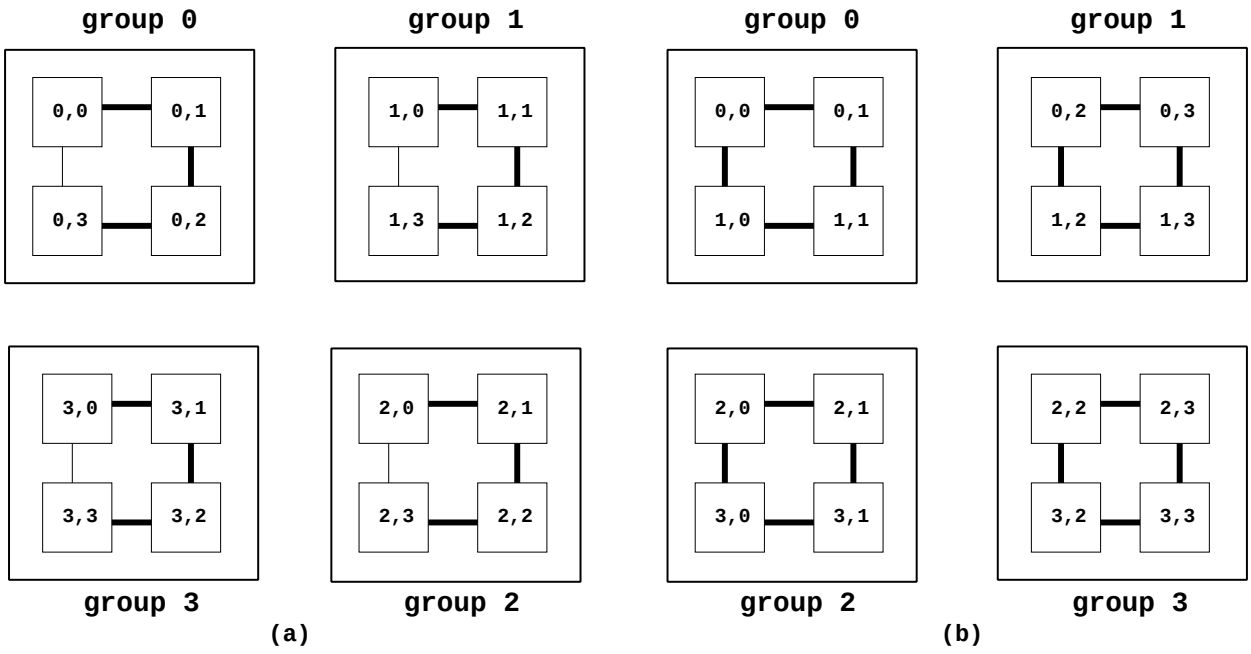

Figure 3: Mapping a $\mathbf{4 \times 4}$ image onto a 16 processor OTIS-Mesh: (a) GRM; (b) GSM

2 Histogramming

2.1 Background

The input to the histogramming problem is an $N \times N$ digitized image I with $I(i,j)$ being the gray level of pixel (i,j), $0 \leq i,j < N$. The gray levels are integers in the range $[0,B)$; that is, $0 \leq I(i,j) < B$, $0 \leq i,j < N$. The histogram of the image is a vector H such that $H[b]$ is the number of pixels with gray value b, $0 \leq b < B$.

Parallel algorithms to compute the histogram of an image have been developed for many parallel architectures. For example, Siegel *et al.*[36] have developed a histogramming algorithm for a p processor PASM multicomputer, $p \leq N^2$, and Yasrebi *et al.*[43] have done this for the TRAC multicomputer. Grinberg *et al.*[7] have developed an algorithm to compute the histogram on an N^2 processor cellular machine called the 3-D machine; Tanimoto [38] has developed an $O(B+\log N)$ algorithm for a pyramid computer with an $N \times N$ base; Bestul and Davis [1] have developed an $O(\sqrt{B}+\log(N/B))$ algorithm for an N^2 processor SIMD hypercube; and Jenq and Sahni [15] and Jang *et al.*[12] have developed algorithms for various reconfigurable mesh models.

The histogramming algorithms of Jang *et al.*[12] and Jenq and Sahni [15] partition B into the ranges $0 < B \leq \sqrt{N}$, $\sqrt{N} < B \leq N$ and $B > N$ and solve for each range separately. Further, where appropriate, they consider the cases $O(1)$ memory per processor and $O(B)$ memory per processor.

We shall follow this strategy here also.

2.2 Algorithm for $0 < B \leq \sqrt{N}$

In this case, the histogram is left in row 0 of the group (0,0) mesh. Using our four-dimensional indexing scheme, processor $(0, 0, 0, i)$ will have $H[i]$, $0 \leq i < B$, following the histogram computation. Our strategy is *(a)* compute the histogram for each row of each $\sqrt{N} \times \sqrt{N}$ mesh, *(b)* use the row histograms to obtain the histogram for each group, and *(c)* combine the group histograms into a single histogram. More formally, our algorithm for the case $0 < B \leq \sqrt{N}$ is:

Step 1: Processor (g_x, g_y, p_x, p_y) determines the number of pixels on its row of the group (g_x, g_y) mesh, $0 \leq p_y < B$.

Step 2: Processor $(g_x, g_y, 0, p_y)$ adds up the values, along its column, that were computed in Step 1, $0 \leq p_y < B$.

Step 3: Perform an OTIS move on the values computed in Step 2.

Step 4: Processor $(0, g_y, 0, 0)$ sums up all the values received in Step 3 by processors in group $(0, g_y)$, $0 \leq g_y < B$.

Step 5: Perform an OTIS move on the results computed in Step 4.

Step 1 is accomplished by shifting the image values first leftward and then rightward within rows of the meshes. When an image value passes through processor (g_x, g_y, p_x, p_y), this processor increments its counter if the image value equals p_y. $\sqrt{N}-1$ leftward and $B-1$ rightward shifts, for a total of $\sqrt{N}+B-2$ electronic moves are needed. Step 2 is done by shifting the counts upwards along columns to row 0 of the group. This step takes $\sqrt{N}-1$ electronic moves. In Step 4 we add all values in a $\sqrt{N} \times \sqrt{N}$ mesh. This takes $2(\sqrt{N}-1)$ electronic moves. Therefore, histogramming can be done with $4(\sqrt{N}-1)+B-1$ electronic and 2 OTIS moves.

Theorem 1 *Our histogramming algorithm for* $0 < B \leq \sqrt{N}$ *is optimal*

Proof We need to show that every OTIS-Mesh histogramming algorithm must make $4(\sqrt{N}-1)+B-1$ electronic and 2 OTIS moves when $0 < B \leq \sqrt{N}$. To see this, consider an image in which $I(0,0) = B-1$ and $I(N-1, N-1) = 0$. Since $I(0,0)$ is mapped to processor (0,0,0,0) of the OTIS-Mesh and since $H[B-1]$ is left in processor $(0, 0, 0, B-1)$, it is necessary for the histogramming

algorithm to move in formation from (0,0,0,0) to $(0,0,0,B-1)$, requiring at least $B-1$ electronic moves that increase the row index (note that OTIS moves can only transpose indices, not change their value). Further, since $I(N-1,N-1)$ is mapped to processor $(\sqrt{N}-1,\sqrt{N}-1,\sqrt{N}-1,\sqrt{N}-1)$ and $H[0]$ is left in (0,0,0,0), it is necessary to make at least $\sqrt{N}-1$ electronic moves to decrease each of the four indices g_x, g_y, p_x, and p_y, a total of $4(\sqrt{N}-1)$ electronic moves. Since moves that increase an index cannot be overlapped with those that decrease an index in the SIMD model, at least $4(\sqrt{N}-1)+B-1$ electronic moves are necessary.

For the 2 OTIS moves, we see that information from all processors in group $(\sqrt{N}-1,\sqrt{N}-1)$ (say) must get to group 0. Assume that group $(\sqrt{N}-1,\sqrt{N}-1)$ has at least 2 gray values. To get information out of a group, an OTIS move must be made. A single OTIS move, however, moves data from different processors of a group into processors of different groups. Therefore, at least 2 OTIS moves are necessary to move different data from 2 or more different processors into a single other group. □

2.3 Algorithms for $\sqrt{N} < B \leq N$

We first present an algorithm that uses $O(1)$ memory per processor. Next, we present an optimal algorithm that uses $O(\sqrt{B})$ memory per processor. Our $O(1)$ memory per processor algorithm leaves the histogram in the processors of group 0, one histogram value per processor. More specifically, processor $(0,0,p_x,p_y)$ contains $H[p_x\sqrt{N}+p_y]$. The algorithm is given below.

Step 1: Processor (p_x,p_y) of each group computes $H[p_x\sqrt{N}+p_y]$ for the subimage in its group. This is done by sorting the gray values in a group using the integer sort algorithm of [20]. During the sort, equal gray values are combined into a single gray value.

Step 2: Perform an OTIS move on the H values computed in Step 1.

Step 3: Processor (0,0) of each group sums the H values in its group that were received in Step 2. That is, processor $(g_x,g_y,0,0)$ computes $H[g_x\sqrt{N}+g_y]$ for the entire image.

Step 4: Processor (0,0) of each group performs an OTIS move on the sum computed in Step 3. Following this move, processor $(0,0,p_x,p_y)$ has $H[p_x\sqrt{N}+p_y]$.

Step 1 takes $4\sqrt{N}+o(\sqrt{N})$ electronic moves; Steps 2 and 4 take 1 OTIS move each; and Step 3 takes $2(\sqrt{N}-1)$ electronic moves. The total number of moves is $6\sqrt{N}+o(\sqrt{N})$ electronic moves and 2 OTIS moves.

Theorem 2 *Every histogramming algorithm for the case* $\sqrt{N} < B \leq N$ *must make at least* $5(\sqrt{N}-1)+\lfloor(B-1)/\sqrt{N}\rfloor-1$ *electronic and 2 OTIS moves to compute the histogram configuration obtained by the* $O(1)$ *memory algorithm. When the output configuration has the histogram in a* $\sqrt{B} \times \sqrt{B}$ *submesh of group (0,0), at least* $4\sqrt{N}+2\sqrt{B}-6$ *electronic and 2 OTIS moves are needed.*

Proof Consider the image in which the gray value of the pixel in processor $(\sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1)$ is 0 and the pixels in group $(\sqrt{N}-1, \sqrt{N}-1)$ have at least 2 different gray values. Using the reasoning in Theorem 1, we see that at least $4(\sqrt{N}-1)$ electronic and 2 OTIS moves are needed to get the histogram information from the group $(\sqrt{N}-1, \sqrt{N}-1)$ processors to the target processors in group (0,0).

Next, suppose that the pixel in processor $(0,0,0,0)$ has gray value v such that $v \bmod \sqrt{N} = \sqrt{N}-1$ and $\lfloor v/\sqrt{N}\rfloor = \lfloor(B-1)/\sqrt{N}\rfloor - 1$. It takes $\lfloor(B-1)/\sqrt{N}\rfloor - 1 + \sqrt{N} - 1$ electronic moves to get information from (0,0,0,0) to $(0, 0, \lfloor(B-1)/\sqrt{N}\rfloor - 1, \sqrt{N}-1)$ and these electronic moves cannot be overlapped with those used to move information from $(\sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1)$ to (0,0,0,0). Therefore, at least $5(\sqrt{N}-1) + \lfloor(B-1)/\sqrt{N}\rfloor - 1$ electronic and 2 OTIS moves are needed to obtain the output configuration obtained by the $O(1)$ memory algorithm.

For the $\sqrt{B} \times \sqrt{B}$ submesh output configuration, suppose that the gray value in processor (0,0,0,0) is $B-1$. Therefore, information needs to flow from (0,0,0,0) to $(0, 0, \sqrt{B}-1, \sqrt{B}-1)$, requiring $2(\sqrt{B}-1)$ electronic moves that cannot be overlapped with the electronic moves made when moving information from $(\sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1, \sqrt{N}-1)$ to (0,0,0,0). Thus a total of $4\sqrt{N}+2\sqrt{B}-6$ electronic and 2 OTIS moves are necessary. □

An optimal histogramming algorithm for the $\sqrt{B} \times \sqrt{B}$ submesh output configuration is possible when $O(\sqrt{B})$ memory per processor is available. The algorithm given below adapts the method used in [15], and assumes that B is a perfect square and that $\sqrt{B}$ divides $\sqrt{N}$.

Step 1: Tile each $\sqrt{N} \times \sqrt{N}$ mesh by $\sqrt{B} \times \sqrt{B}$ tiles.

Step 2: Processor i on each row of each $\sqrt{B} \times \sqrt{B}$ tile computes an array $A[0 : \sqrt{B}-1]$ of values such that $A[j]$ equals the number of pixels in that row of the tile whose gray value is $j\sqrt{B}+i$, $0 \leq i < \sqrt{B}$.

Step 3: Processors in the same column of each tile perform a consecutive sum operation; processor i of a tile column sums the $A[i]$ values of the processors on its column. Following this step, processor (i, j) of a tile has the number of pixels in its tile whose gray value is $i\sqrt{B}+j$.

Step 4: Perform a window sum operation on the results of Step 3 using a window size $\sqrt{B} \times \sqrt{B}$. This operation does not span group boundaries. The result of the window sum operation is in the top left $\sqrt{B} \times \sqrt{B}$ window/tile of each group. Following this operation, processor (g_x, g_y, i, j) has the number of pixels in group (g_x, g_y) whose gray value is $i\sqrt{B} + j$.

Step 5: Do an OTIS move on the values computed in Step 4.

Step 6: Processor $(g_x, g_y, 0, 0)$ sums all the values received by its group.

Step 7: Do an OTIS move on the values computed by $(g_x, g_y, 0, 0)$ in Step 6.

For the time complexity, we see that Steps 2 and 3 take $2(\sqrt{B} - 1)$ electronic moves each; Step 4 takes $2(\sqrt{N} - \sqrt{B})$ electronic moves; Steps 5 and 7 take 1 OTIS move each; and Step 6 takes $2(\sqrt{N} - 1)$ electronic moves. The total number of moves is $4\sqrt{N} + 2\sqrt{B} - 6$ electronic and 2 OTIS moves.

2.4 Algorithm for $B > N$

This case can be done with $22\sqrt{N} + O(N^{3/8})$ electronic and $O(N^{3/8})$ OTIS moves by modifying the sort algorithm of [41] so that during the sort, pixels with the same gray value are combined into a single pixel.

3 Histogram Modification

Histogram modification is the process of changing the gray values of an image based on a mapping function f; $f(i)$ gives the new gray value for pixels whose original gray value is i, $0 \leq i < B$. In histogram flattening or equalization [31], the function f is computed by first computing the prefix sums $S[i] = \sum_{j=0}^{i} H[j]$, $0 \leq i < B$ of the histogram. Next, $f(i)$ is obtained using one of the following equations

$$f(i) = \lfloor S[i]/B \rfloor,\ 0 \leq i < B,$$

or

$$f(i) = \lfloor \tfrac{S[i]+S[i-1]}{2B} \rfloor,\ 0 \leq i < B$$

where $S[-1] = 0$.

In the OTIS implementation of histogram flattening, we explicitly consider only the case $B = N$. Other values of B may be handled similarly. The prefix sums may be computed from the histogram

(which is in group 0) using $3(\sqrt{N}-1)$ electronic moves [41]. To compute f using the first equation, no additional moves are needed. When the second definition is used, additional electronic moves are needed to shift the prefix sums by 1 processor.

Following the computation of f, the gray values of all pixels must be updated according to f. When we are limited to $O(1)$ memory per processor, this updating of pixel values may be done by first performing a window broadcast of the f values to all groups. This broadcast can then be followed by a random access read (RAR) in which each processor obtains the needed f value from within its group. The window broadcast takes $2(\sqrt{N}-1)$ electronic moves and the RAR takes $23\sqrt{N}+o(\sqrt{N})$ electronic moves [41]. Thus, the updating phase takes $25\sqrt{N}+o(\sqrt{N})$ electronic and 2 OTIS moves.

When $O(\sqrt{N})$ ($= O(\sqrt{B})$) memory per processor is available, the updating of group values may be done by first doing a window broadcast of the f values as was done in the $O(1)$ memory case. Next, each processor accumulates the f values in the $\sqrt{N}$ processors that are in its column. This accumulation is done in an array C. For a processor in column j of its group, $C[i] = f(i\sqrt{N}+j)$, $0 \leq i,j < \sqrt{N}$. This accumulation step takes $2(\sqrt{N}-1)$ electronic moves. Following the construction of the C arrays, each processor sends a token to the processor on its row that has the f value it needs. When the token reaches the target processor, the f value is written into the token, and the token returned to the originating processor. This token send/receive step can be broken into two phases – one in which tokens are sent to and received from processors to the left of the source processors and another in which the target processors are to the right. Each of these phases takes $2(\sqrt{N}-1)$ electronic moves. Thus, the $O(\sqrt{N})$ memory algorithm takes $8(\sqrt{N}-1)$ electronic and 2 OTIS moves to update the gray values following the computation of f.

The complexity of the $O(\sqrt{N})$ memory updating algorithm can be reduced to $6(\sqrt{N}-1)$ electronic and 2 OTIS moves if the histogram computation phase saves, in processors 0 and $\sqrt{N}-1$ of each row of a group, the gray values of all $\sqrt{N}$ pixels in that row. This can be done without increasing the number of moves taken by the histogramming algorithm. When processors 0 and $\sqrt{N}-1$ of each row know the gray values of all pixels in their row, the pixel values can be updated using the C arrays in $2(\sqrt{N}-1)$ electronic moves rather than in $4(\sqrt{N}-1)$ electronic moves. In the new method, processor 0 of each row initiates token $t_{\sqrt{N}-1}$, $t_{\sqrt{N}-2}$, ..., t_1 in that order. Token t_i contains the gray value of the pixel in processor i of the row. The tokens move rightward, one processor at a time. When a processor receives a token, it checks the token's gray value and appends the updated value for that gray value in case this updated value is stored in the processor's

C array. The rightward token moves are made for exactly $\sqrt{N}-1$ steps. Following this, processor $\sqrt{N}-1$ of each row initiates tokens t_0, t_1, ..., $t_{\sqrt{N}-2}$ that move leftward for exactly $\sqrt{N}-1$ steps. At the end of these moves, each processor should have received a pair $(i, f(i))$ where i is the original gray value in the processor.

4 Shrinking and Expanding

4.1 Background

Let I be an $N \times N$ image and let $B_{2q+1}[i,j]$ represent the pixel block:

$$\{[u,v] | 0 \leq u, v < N, \max\{|u-i|, |v-j|\} \leq q\}.$$

The q-step expansion, E^q, and the q-step shrinking, S^q, of I are given by the equations [33]:

$$E^q[i,j] = \max\{I[u,v] | [u,v] \in B_{2q+1}[i,j]\},\ 0 \leq i,j < N$$
$$S^q[i,j] = \min\{I[u,v] | [u,v] \in B_{2q+1}[i,j]\},\ 0 \leq i,j < N$$

Rosenfeld [33] has developed an $O(k)$ algorithm to compute E^{2^k-1} and S^{2^k-1} at coarsely resampled points using a pyramid computer with an $N \times N$ base. Jenq and Sahni [16] develop an $O(\sqrt{q})$ time algorithm to compute E^q and S^q exactly on an $N \times N$ base pyramid. Ranka and Sahni [32] have developed an $O(\log N)$ time algorithm to compute E^q and S^q exactly using an N^2 processor hypercube computer. Jenq and Sahni have developed an $O(1)$ time RMESH algorithm to compute E^q and S^q for binary images [13].

Since image expansion and shrinking are computationally equivalent, we discuss only image expansion explicitly. Following Ranka and Sahni [32], we compute E^q using the decomposition given below.

$$\begin{aligned}
E^q[i,j] &= \max\{top^q[i,j], bottom^q[i,j]\}, \text{ where} \\
top^q[i,j] &= \max\{R^q[u,j] | 0 \leq i-u \leq q\}, \text{ and} \\
bottom^q[i,j] &= \max\{R^q[u,j] | 0 \leq u-i \leq q\}. \\
R^q[i,j] &= \max\{left^q[i,j], right^q[i,j]\}, \text{ where} \\
left^q[i,j] &= \max\{I[i,v] | 0 \leq j-v \leq q\}, \text{ and} \\
right^q[i,j] &= \max\{I[i,v] | 0 \leq v-j \leq q\};
\end{aligned}$$

The algorithm to use to compute $right^q$ depends on whether the image is mapped onto the OTIS computer using the GRM or the GSM mapping.

4.2 GRM Mapping

In the GRM mapping, each row of the image is mapped onto an OTIS group in snake-like order. A simple way to compute $right^q$ is to shift the gray values leftward by q units along the snake. A left shift by 1 takes 1 electronic left move, 1 electronic right move, and 1 electronic up move. Therefore, $right^q$ can be computed using a total of $3q$ electronic moves. We can overlap the left and right electronic moves required to compute $left^q$ and $right^q$ so that the total number of moves in each of the four mesh directions is q. That is, we can compute $left^q$ and $right^q$ using $4q$ electronic moves. To compute top^q and $bottom^q$, we first do an OTIS move so that each column of R^q is in a single group in snake-like order; then we run the $left^q$ and $right^q$ algorithm; and finally do another OTIS move to get the E^q values back in the proper locations. Thus E^q can be computed using $8q$ electronic and 2 OTIS moves. When $q > 1.25\sqrt{N}$, this simple strategy can be improved upon as below.

When $q > 1.25\sqrt{N}$, $right^q$ of a column 0 processor on an even indexed row (within a group) depends on the $\sqrt{N}-1$ values to its right and on the same row, all values on the following $q_f = \lfloor (q-\sqrt{N}+1)/\sqrt{N} \rfloor = \lfloor (q+1)/\sqrt{N} \rfloor - 1$ rows, and $q_m = q-\sqrt{N}+1-q_f\sqrt{N} = (q+1) \bmod \sqrt{N}$ values in the row q_f+1 rows away (Figure 4(a) and (b)). Likewise, $right^q$ of a column $\sqrt{N}-1$ processor on an odd indexed row depends on the $\sqrt{N}-1$ values to its left and on the same row, all values on the following q_f rows, and q_m values in the row q_f+1 rows away (Figure 4(c) and (d)).

Also, $right^q$ of a column i processor on an even indexed row, $i \neq 0$, depends on the $\sqrt{N}-i-1$ values to its right and on the same row, all values in the following q_f rows, and an additional q_m+i values (Figure 5(a) and (b)). Similarly, for a column i processor on an odd indexed row, $i \neq \sqrt{N}-1$, $right^q$ depends on the i values to its left and on the same row, all values in the following q_f rows, and an additional q_m+i values (Figure 5(c) and (d)).

When $q_m = 0$ or 1, the additional q_m+i values lie on a single row for all $i \in [0, \sqrt{N}-1]$. When $q_m = 0$ or 1, we use the following steps to compute $right^q$.

Step 1: Column 0 processors compute the maximum gray value in their row.

Step 2: Column 0 processors shift the maximum computed in Step 1 up column 0 for q_f steps. Each column 0 processor computes the max of the q_f values it receives.

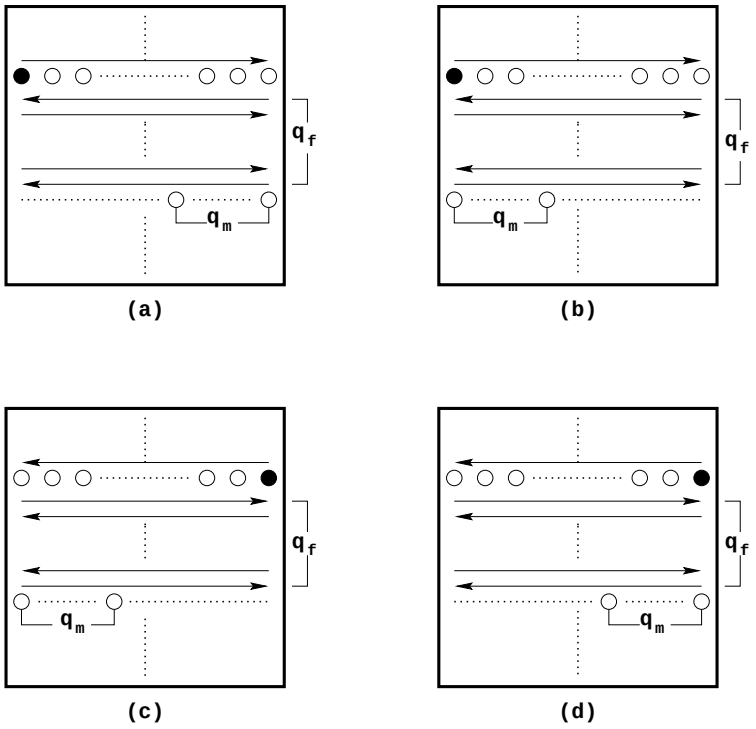

Figure 4: Data required in GRM for end processor: (a) q_f even; (b) q_f odd; (c) q_f even; (d) q_f odd

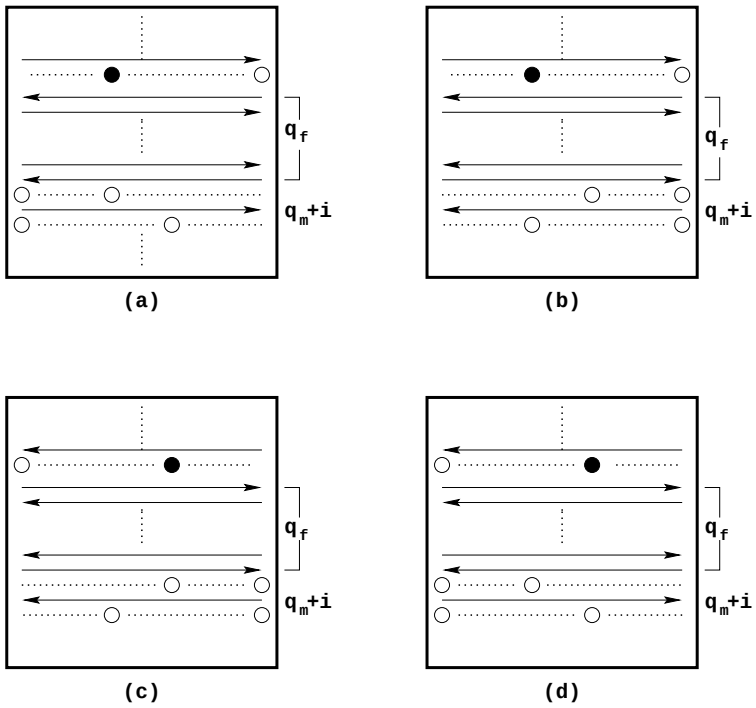

Figure 5: Data required in GRM mapping for middle processor: (a) q_f even; (b) q_f odd; (c) q_f even; (d) q_f odd

Step 3: Each processor shifts its image value up by $q_f + 1$ rows. Following this step, each row has the additional $\sqrt{N}$ values it needs from a row $q_f + 1$ rows away.

Step 4: $right^q$ may now be computed by circulating the original image values in a row, the additional values from a row $q_f + 1$ away, and the max value of the intermediate q_f rows.

Step 1 takes $\sqrt{N} - 1$ leftward moves, Step 2 takes q_f upward moves, and Step 3 takes $q_f + 1$ upward moves. For Step 4, we note that in some rows, the original row values are to be shifted left while in others, these are to be shifted right. Rows which require a left shift of same row values, require a right shift of the row $q_f + 1$ away values, and rows that require a right shift of same row values, require a left shift of row $q_f + 1$ away values. So, the same row and row $q_f + 1$ away values can be circulated through the processors that need these values using $2(\sqrt{N} - 1)$ electronic moves. The rightward circulation of the max value of the intermediate rows can be done with one additional move by pipelining it with the rightward moves for that row. The total number of moves for the computation of $right^q$ is $3(\sqrt{N} - 1) + 2q_f + 2 = 3\sqrt{N} + 2q_f - 1$ electronic and 0 OTIS. The computation of $left^q$ requires only $2(\sqrt{N} - 1) + 2q_f + 2$ electronic moves because the row max values need not be recomputed. Thus the $left^q$ and $right^q$ values may be computed using a total of $5(\sqrt{N} - 1) + 4q_f + 4$ electronic moves. top^q and $bottom^q$ can be computed similarly using $5(\sqrt{N} - 1) + 4q_f + 4$ electronic and 2 OTIS moves. Thus the total number of moves to compute E^q is $10(\sqrt{N} - 1) + 8q_f + 8$ electronic and 2 OTIS moves.

When $q_m > 1$, the additional values needed to compute $right^q$ lie on two rows – one is $q_f + 1$ away and the other is $q_f + 2$ away. The number of gray values needed from a row $q_f + 2$ away is $q_m - 1$. These $q_m - 1$ values can reach the row that needs them if Step 3 of the $q_m \leq 1$ algorithm shifts upwards for $q_f + 2$ steps, rather than $q_f + 1$. So, only one additional upward move is needed. The row $q_f + 2$ values can be circulated in Step 4 by pipelining their movement along with that of the row $q_f + 1$ values. This increases the leftward and rightward moves by $q_m - 1$ each. The total move count for $right^q$ increases by $2q_m - 1$ over that for the case $q_m \leq 1$. The new count for E^q becomes $10(\sqrt{N} - 1) + 8q_f + 8q_m + 4$ electronic and 2 OTIS moves.

4.3 GSM Mapping

The strategy to compute E^q when a GSM mapping is used is similar to that used for a GRM mapping. Notice that in a GSM mapping, a row of the image is distributed over $\sqrt{N}$ groups with $\sqrt{N}$ row elements per group (Figure 6).

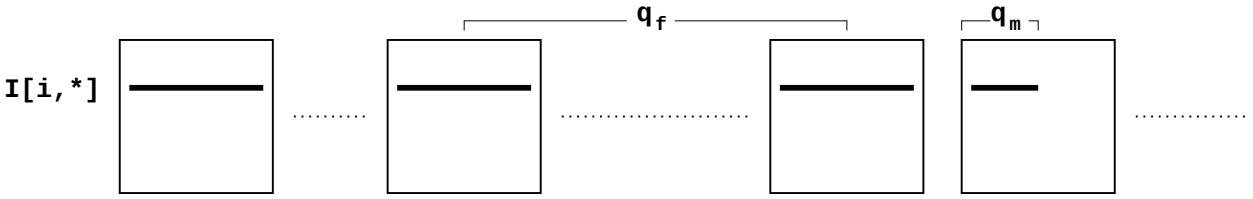

Figure 6: Data required in GSM mapping

We consider three cases: *(a)* $q \leq \sqrt{N}$, *(b)* $q_f = \lfloor (q - \sqrt{N} + 1)/\sqrt{N} \rfloor = \lfloor (q+1)/\sqrt{N} \rfloor - 1 \neq 0$ and $q_m = (q+1) \bmod \sqrt{N} = 0$, and *(c)* $q_f \geq 0$ and $q_m \neq 0$. The case $q_f = 0$ and $q_m \leq 1$ is included in *(a)*.

When $q \leq \sqrt{N}$, the gray values needed to compute $right^q$ for processor (g_x, g_y, p_x, p_y) are in $(g_x, g_y, p_x, *)$ and $(g_x, g_y + 1, p_x, *)$. The Steps to compute $right^q$ are:

Step 1: Perform the following sequence of moves on the gray values:

$$\begin{aligned} (g_x, g_y + 1, p_x, p_y) &\xrightarrow{\text{O}} (p_x, p_y, g_x, g_y + 1) \\ &\xrightarrow{\text{E}} (p_x, p_y, g_x, g_y) \\ &\xrightarrow{\text{O}} (g_x, g_y, p_x, p_y) \end{aligned}$$

This moves gray values from $(g_x, g_y + 1, p_x, p_y)$ to (g_x, g_y, p_x, p_y). Now. each row in a group has all the gray values needed to compute $right^q$ for all processors in the group row.

Step 2: Shift original gray values leftwards within a group row by q units.

Step 3: Shift gray values received in Step 1 rightwards by $\sqrt{N} - 1$ units.

Steps 2 and 3 cause all data needed by a processor to go through the processor, enabling the processor to compute its $right^q$ value. The total number of moves needed for the case $q \leq \sqrt{N}$ is $\sqrt{N} + q$ electronic and 2 OTIS moves. The moves needed to compute E^q become $4\sqrt{N} + 4q$.

When $q_f \neq 0$ and $q_m = 0$, the $right^q$ value of (g_x, g_y, p_x, p_y) is the max of $I(g_x, g_y, p_x, p_y)$, the value to the right of (g_x, g_y, p_x, p_y) and in the same row and group, the max of the values in $(g_x, g_y + i, p_x, *)$, $1 \leq i \leq q_f$, and some of the values $(g_x, g_y + q_f + 1, p_x, *)$ (but not $(g_x, g_y + q_f + 1, p_x, \sqrt{N} - 1)$). The steps to use are:

Step 1: Processor $(g_x, g_y, p_x, 0)$ determines the max of the gray values in $(g_x, g_y, p_x, *)$.

Step 2: Processor (g_x, g_y, p_x, p_y) sends its gray value to $(g_x, g_y, p_x, p_y + 1)$, $0 \leq p_y < \sqrt{N} - 1$.

Step 3: Perform the following move sequence on the max values computed in Step 1:

$$\begin{aligned}
(g_x, g_y, p_x, 0) &\xrightarrow{\text{O}} (p_x, 0, g_x, g_y) \\
&\xrightarrow{\text{E}} (p_x, 0, g_x, g_y - 1) \\
&\xrightarrow{\text{E}} (p_x, 0, g_x, g_y - 2) \\
&\vdots \\
&\xrightarrow{\text{E}} (p_x, 0, g_x, g_y - q_f) \\
&\xrightarrow{\text{O}} (g_x, g_y - q_f, p_x, 0)
\end{aligned}$$

During the electronic moves, each processor computes the max of the q_f max values that pass through it.

Step 4: Perform the following move sequence on the shifted values of Step 2. Note that $p_y \neq 0$.

$$\begin{aligned}
(g_x, g_y, p_x, p_y) &\xrightarrow{\text{O}} (p_x, p_y, g_x, g_y) \\
&\xrightarrow{\text{E}} (p_x, p_y, g_x, g_y - 1) \\
&\vdots \\
&\xrightarrow{\text{E}} (p_x, p_y, g_x, g_y - q_f - 1) \\
&\xrightarrow{\text{O}} (g_x, g_y - q_f, p_x, p_y)
\end{aligned}$$

This moves the additional values that processors in group (g_x, g_y) need from group $(g_x, g_y + q_f + 1)$.

Step 5: Shift the max of the max' in $(g_x, g_y, p_x, 0)$ (received in Step 3) and the additional values received in Step 4 by (g_x, g_y, p_x, p_y), $p_y \neq 0$, rightwards $\sqrt{N} - 1$ times within a group row.

Step 6: Shift original values leftward $\sqrt{N} - 1$ times within a group row.

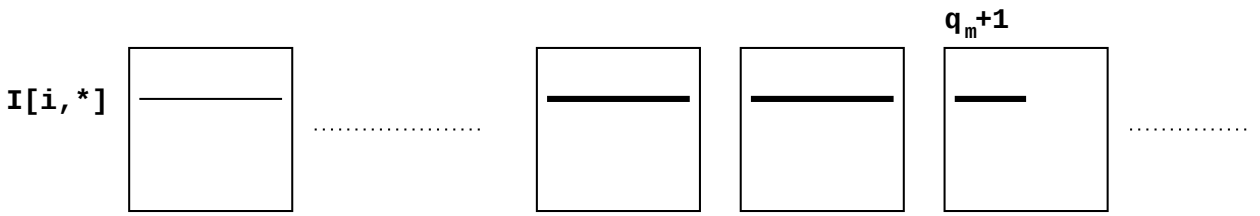

Figure 7: Data required in GSM mapping when $q_f = 0$ and $q_m \neq 0$

Steps 5 and 6 cause all data needed by a processor to go through it and enable the processor to compute $right^q$. Step 1 requires $\sqrt{N}-1$ electronic moves, Step 3 requires 2 OTIS and q_f electronic moves, Step 4 requires 2 OTIS and q_f+1 electronic moves, and Steps 5 and 6 each require $\sqrt{N}-1$ electronic moves. Note, however, that all moves of Step 3 can be overlapped with moves of Step 4. Moreover, Step 6 can be combined with Step 1. Therefore $right^q$ can be computed using $2\sqrt{N}+q_f$ electronic and 2 OTIS moves.

When computing $left^q$, Step 1 can be omitted as the max values have already been computed and Step 2 is not required either. However, Step 6 contributes $\sqrt{N}-1$ electronic moves now. So $left^q$ can be computed with an additional $2\sqrt{N}+q_f-1$ electronic and 2 OTIS moves. top^q and $bottom^q$ can be computed similarly. The total number of moves needed to compute E^q is $8\sqrt{N}+4q_f-2$ electronic and 8 OTIS moves.

The final case to consider is when $q_f \geq 0$ and $q_m \neq 0$. If $q_f = 0$, we may assume $q_m > 1$ because the case $q_f = 0$ and $q_f \leq 1$ is covered by the case $q \leq \sqrt{N}$. Now we need to shift values from two adjacent groups (Figure 7); $\sqrt{N}$ from the group on the right and $q_m - 1$ from the group two to the right. This shifting can be done using the move sequence

$$\begin{aligned}
(g_x, g_y, p_x, p_y) &\xrightarrow{\text{O}} (p_x, p_y, g_x, g_y) \\
&\xrightarrow{\text{E}} (p_x, p_y, g_x, g_y - 1) \\
&\xrightarrow{\text{E}} (p_x, p_y, g_x, g_y - 2) \\
&\xrightarrow{\text{O}} (g_x, [g_y - 1, g_y - 2], p_x, p_y)
\end{aligned}$$

In the last OTIS move, two values from each processor are moved. These are the two values that pass through the processor during the two electronic moves stated above. To compute $right^q$, we must now shift the original gray values leftward within each group $\sqrt{N}-1$ times, shift the values from the right adjacent group leftward $q_m - 1$ times, and shift the values from the group

two away rightward $\sqrt{N}-1$ times. The computation of $right^q$ takes 2 OTIS and $2\sqrt{N}+q_m-1$ electronic moves. $left^q$, top^q, and $bottom^q$ may be similarly computed. The total number of moves to compute E^q is therefore $8\sqrt{N}+4q_m-4$ electronic and 8 OTIS.

If $q_f > 0$, we must also compute the max of the values in the intermediate q_f groups as was done for case *(b)*. This adds $4q_f$ electronic and 0 OTIS move to the computation of E^q.

5 Hough Transform

5.1 Background

The Hough transform is used to detect straight lines or edges in an image. The p angle Hough transform [32] of an $N \times N$ image I is a two-dimensional array H such that

$$H[r,j] = |\{(x,y) | r = \lfloor x\cos\theta_j + y\sin\theta_j \rfloor,\ \theta_j = \tfrac{\pi}{p}(j+1) \text{ and } I[x,y]=1\}|$$

Here j has the values 0, 1, ..., $p-1$. These values of j correspond to the p angles $\theta_j = \frac{\pi}{p}(j+1)$. Since θ_j is in the range $(0,\pi]$ and since $0 \le x,y < N$, i is in the range $[-\sqrt{2}N, \sqrt{2}N]$.

Parallel algorithms to compute the Hough transform have been developed for several architectures. Chaung and Li [2] and Li *et al.*[23] do this for systolic arrays; Rosenfeld *et al.*[34], Kannan and Chaung [17], Cypher *et al.*[4], Guerra and Hambrusch [8], and Silberberg [37] consider mesh computers; Fisher and Highnam [6] use a scan line array; Ibrahim *et al.*[10] uses a SIMD tree; Li *et al.*[21, 22], Maresca *et al.*[25], use a polymorphic torus; Ranka and Sahni [32] use hypercube computers; Choudhary and Ponnusamy [3] and Thazhuthaveetil and Shah [39] use shared-memory multiprocessors; Jenq and Sahni [14] and Olariu *et al.*[29] use reconfigurable meshes; and Pavel and Akl [30] use optical arrays. Illingworth and Kittler [11] provide a survey of work related to the Hough transform.

5.2 An Improved Algorithm For $N \times N$ Meshes

Our development here is based on the work of Jenq and Sahni [14], which itself is closely related to the work reported in several of the other references cited above. The computation of H is generally broken into four phases; each phase computes H for a certain range of θ_j. The four ranges are *(a)* $0 < \theta_j \le \pi/4$, *(b)* $\pi/4 < \theta_j \le \pi/2$, *(c)* $\pi/2 < \theta_j \le 3\pi/4$, and *(d)* $3\pi/4 < \theta_j \le \pi$. Since the computation of H in each of these θ_j ranges is similar, the computation is explicitly described for only one of the four ranges. As in [14], we explicitly consider only the case $\pi/2 < \theta_j \le 3\pi/4$.

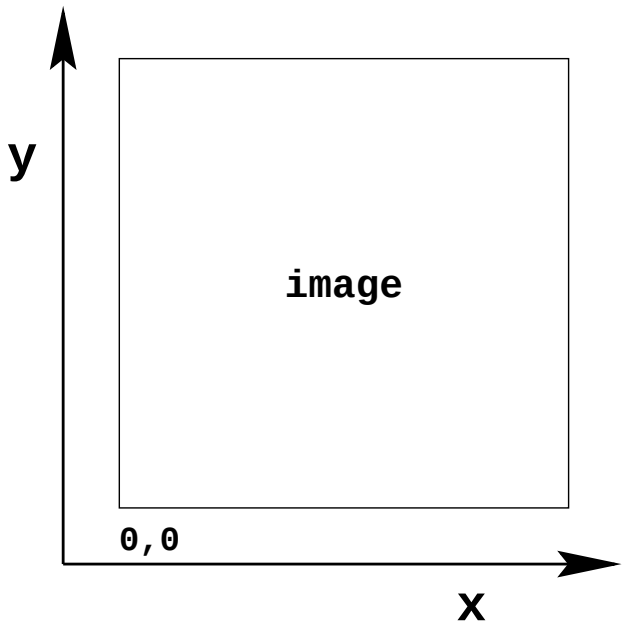

Figure 8: Coordinate system used in Hough Transform

Jenq and Sahni [14] have shown how to compute $H[r, j]$ for $r \geq 0$ and j such that $\pi/2 < \theta_j \leq 3\pi/4$ on a two-dimensional mesh computer by starting tokens on two boundaries of the mesh and successfully moving these tokens towards the remaining two boundaries. These tokens accumulate the $H[r, j]$ values. When we use the normal convention of locating the original of the coordinate system at the bottom-left corner of the and so at the bottom-left corner of the $N \times N$ mesh (see Figure 8), tokens originate at the left and bottom boundaries of the image/mesh and move up and right till they reach either the top or right boundary. For $\pi/2 < \theta_j \leq 3\pi/4$, the rules governing token movement are derived from the following facts. Here θ is the angle and $r(x, y) = \lfloor x \cos\theta + y \sin\theta \rfloor$.

(a) If $r(x, y) = r(x, y + k)$ for some $k > 0$, then $k = 1$.

(b) If $r(x, y) = r(x + 1, c)$, then $c = y$ or $c = y + 1$.

(c) If $r(x, y) = r(x, y + 1) = r(x + 1, c)$, then $c = y + 1$.

(d) $If r(x, y) \neq r(z, y + 1)$ for $z > x$, then $r(x, y) \neq r(z, w)$ for $w > y$.

Using these facts, we arrive at the following algorithm to compute $H[r, j]$ for a single angle $\theta = \theta_j$.

Step 1: [Create Tokens on Left and Bottom Boundaries] Processor $(0, y)$ creates the token

$(sinv, cosv, r, n) = (\sin\theta, \cos\theta, r(0,y), I[0,y])$ provided $r(0,y) \neq r(0,y-1)$. Processor $(x,0)$ creates the token $(\sin\theta, \cos\theta, r(x,0), I[x,0])$ provided $r(x,0) \neq r(x-1,0)$.

Step 2: [Move Tokens Up] Let $(sinv, cosv, r, n)$ be the token (if any) in processor (x,y). If $r = r(x,y+1)$ or $r \neq r(x+1,y)$, move the token to $(x,y+1)$. If (x,y) receives a token $(sinv, cosv, r', n')$ in this step, it increments n' by $I[x,y]$ provided $r' = r(x,y)$.

Step 3: [Move Tokens Right] All tokens move right from (x,y) to $(x+1,y)$. If (x,y) receives a token $(sinv, cosv, r', n')$ in this step, it increments n' by $I[x,y]$ provided $r' = r(x,y)$.

Step 4: Repeat Steps 2 and 3 until all tokens reach the top or right boundary.

Theorem 3 *The four step procedure given above is correct.*

Proof To establish the correctness of this procedure, we must show that every token $(sinv, cosv, r, n)$ visits all processors (x,y) for which $r(x,y) = r$. Note that when $\pi/2 < \theta \leq 3\pi/4$, $-1/\sqrt{2} \leq \cos\theta < 0$, $1/\sqrt{2} \leq \sin\theta < 1$, and $\sin\theta + \cos\theta \geq 0$. Consider the configuration following Step 1. If token $(sinv, cosv, r, n)$ is in $(0,y)$ then all (x,z) for which $r(x,z) = r$ satisfy $x \geq 0$ and $z \geq y$. Further, if the token is in $(x,0)$, then all (z,y) for which $r(z,y) = r$ satisfy $z \geq x$ and $y \geq 0$. Therefore, all unreached processors with the same r value can be reached by making upward and rightward moves along. For any token $(sinv, cosv, r, n)$ in processor (x,y), let property P be: all unreached processors (u,v) with $r(u,v) = r$ have $u \geq x$ and $v \geq y$ (*i.e.*, these processors can be reached by making upward and rightward moves alone). We have already shown that P holds following Step 1. We shall show that this property holds following each execution of Steps 2 and 3. Therefore the algorithm is correct. To establish the result, we will also need to show that at the start of Step 2 $r(x,y) = r$ (so, we need not check $r' = r(x,y)$ in Step 3). The first time Step 2 is initiated, $r(x,y) = r$ and so this condition holds. Call this condition Q. Assume, for the induction hypothesis, that P and Q hold at the start of each execution of Step 2.

Consider a token that moves up in Step 2. Suppose that $r = r(x,y)$. If $r = r(x,y+1)$ also, then $r = r(x,y) = r(x,y+1)$. From fact *(c)* it follows that $r \neq r(x+1,y)$. From this and $\cos\theta < 0$, it follows that $r > r(x+1,y) \geq r(x+j,y)$, $j \geq 2$. Therefore following the upward move of the token, property P still holds for the token. From fact *(a)*, $r = r(x,y) = r(x,y+1)$, and $\sin\theta > 0$, it follows that $r < r(x,y+j)$, $j \geq 2$. Therefore, following the rightward move of this token in Step 3, property P again holds. Following the Step 3 move of the token, the token is in processor $(x+1,y+1)$ and $r(x+1,y+1) = \lfloor (x+1)\cos\theta + (y+1)\sin\theta \rfloor = \lfloor x\cos\theta + y\sin\theta + \cos\theta + \sin\theta \rfloor \geq \lfloor x\cos\theta + y\sin\theta \rfloor = r$

(because $\cos\theta + \sin\theta \geq 0$). This, together with the knowledge that $r = r(x, y+1) \geq r(x+1, y+1)$, implies that $r(x+1, y+1) = r$. So condition Q holds after Step 3.

Next, suppose that $r = r(x, y)$, $r \neq r(x, y+1)$, and $r \neq r(x+1, y)$. Again, the token moves to $(x, y+1)$ in Step 2. Since $-1/\sqrt{2} \leq \cos\theta < 0$, $r(x+1, y) = r - 1 \geq r(x+j, y)$, $j \geq 2$. Therefore moving the token up to $(x, y+1)$ preserves property P. P is also preserved following the rightward move made in Step 3 because $r(x, y+j) \geq r(x, y+1) = r+1$, $j \geq 2$. Further, since $1/\sqrt{2} \leq \sin\theta < 1$, $r(x, y+1) = r+1$. Since $r(x+1, y+1) \in \{r(x, y+1), r(x, y+1) - 1\} = \{r+1, r\}$ as well as in $\{r(x+1, y), r(x+1, y) + 1\} = \{r-1, r\}$, $r(x+1, y+1) = r$. Therefore Q also holds following Step 3.

If a token does not move up in Step 2, then $r = r(x, y) = r(x+1, y)$ at the start and end of Step 2. Following Step 3, the token has moved to $(x+1, y)$ and so condition Q holds. Also since $r \neq r(x, y+1)$ and $\sin\theta \geq 1/\sqrt{2}$, $r < r(x, y+1) \leq r(x, y+j)$, $j \geq 2$. Therefore the right move of Step 3 preserves property P also. □

Theorem 3 establishes the correctness of our $N \times N$ mesh single angle Hough transform algorithm by showing that each token goes through every processor with the same r value as that of the token. For the complexity analysis, we first observe that there can never be more than one token in any processor. To see this, observe that when the algorithm completes in Step 1, no processor has more than 1 token. If two tokens t_1 and t_2 end up in the same processor following Step 2, then t_1 must be in (x, y) and t_2 in $(x, y+1)$ prior to the upward move of Step 2. Further, in Step 2, t_1 must move to $(x, y+1)$ and t_2 must remain in $(x, y+1)$. From condition Q of Theorem 3, we know that the r value r_1 of token t_1 must be $r(x, y)$ and $r_2 = r(x, y+1)$. Since all tokens in the same column must originate in the same column (because tokens move right at the same rate), $r_1 \neq r_2$ (alternatively, since all tokens have different r values, $r_1 \neq r_2$). Therefore, $r_2 = r(x, y+1) = r_1 + 1$. For t_1 to move up and t_2 to remain in $(x, y+1)$, we must have $r(x+1, y) = r_1 - 1$ (this causes t_1 to move up), $r(x+1, y+1) = r_1 + 1$ (this causes t_2 to not move up). However, $r(x+1, y)$ and $r(x+1, y+1)$ can differ by at most 1. Therefore, this condition is not possible and so two tokens cannot be in the same processor following Step 2. Since all tokens move right in Step 3, two tokens cannot be in the same processor after Step 3 unless they are in the same processor before Step 3. Therefore, it is not possible for two tokens to ever be in the same processor.

Since tokens are always in different processors, the upward move of Step 2 can be done in one time unit and the rightward move of Step 3 can be done in another time unit. Since tokens can make at most $N-1$ right moves before reaching the right boundary, at most $N-1$ iterations

of Steps 2 and 3 are needed. Therefore, the single angle Hough transform can be computed with $2(N-1)$ moves. This represents an improvement over the algorithm of Jenq and Sahni [14] which takes $3(N-1)$ moves to compute the single angle Hough transform.

To compute the p angle transform, we modify the basic one angle algorithm so that the token originating in $(x,0)$ is created only when the tokens that originated in $(0,y)$ reach column x; that is, the $(x,0)$ token is created after x rightward moves have been made in Step 3. With this change, all tokens are always in the same column and correctness is not affected. Further, when tokens reach the top or right boundary, they start moving down from the top boundary or left from the right boundary. This avoids an accumulation of multiple tokens in the same boundary processor.

Since the modified algorithm uses only one column of the $N \times N$ mesh at any time (excluding the backward movement of tokens from the top and right boundaries), we can pipeline the computation for all $p/4$ angles in the range $\pi/2 < \theta \leq 3\pi/4$. The total computation takes $3(N-1)+4(p/4-1) = 3N+p-7$ moves. The number of moves needed for all p angles is $12N+4p-28$. A final sort step is needed to create the array H in the desired format. This takes an additional $(4+\epsilon)N$ moves [28].

5.3 GRM Mapping

We can simulate the single angle Hough transform algorithm of Section 5.2 as well as the modified version of this algorithm on an N^2 processor OTIS-Mesh in which the image has been mapped using the GRM mapping. Each execution of Step 2 can be done with 2 OTIS moves and 3 electronic moves, and each execution of Step 3 takes 3 electronic moves. The total number of moves for the single angle transform becomes $6(N-1)$ electronic and $2(N-1)$ OTIS moves. To compute the transform for all $p/4$ angles in the range $\pi/2 < \theta \leq 3\pi/4$ takes $7(N-1)+8(p/4-1) = 7N+2p-15$ electronic and $2(N-1)+2(p/4-1) = 2N+p/2-4$ OTIS moves, and to compute the transform for all p angles takes $28N+8p-60$ electronic and $8N+2p-16$ OTIS moves, exclusive of the final sort step.

5.4 GSM Mapping

When the GSM mapping is used, $H[r,j]$ is to be left in processor $(\lfloor r/\sqrt{N} \rfloor, \lfloor j/\sqrt{N} \rfloor, r \bmod \sqrt{N}, j \bmod \sqrt{N})$. Since $-\sqrt{2}N \leq r \leq \sqrt{2}N$, each processor gets at most 2 values. For each set of $p/4$ angles, we compute H using the pipelined $\sqrt{N} \times \sqrt{N}$ mesh algorithm of Section 5.2 in each group of the OTIS-Mesh. However, this pipelined algorithm is interrupted after the tokens for the $k\sqrt{N}$th an-

gles, $1 \leq k \leq \lfloor \frac{p}{4\sqrt{N}} \rfloor$, reach the top and right boundaries of a group. When the pipelined algorithm is interrupted, we combine partial H values from the entire OTIS-Mesh using these steps:

Step 1: Rearrange the tokens in each group for the $\sqrt{N}$ angles just computed. The token for $H[r, j]$ is to be in processor $(*, *, r \bmod \sqrt{N}, j \bmod \sqrt{N})$; here j is an angle index.

Step 2: Perform an OTIS move on the tokens that were rearranged in Step 1. Now, tokens that contribute to $H[r, j]$ are in $(r \bmod \sqrt{N}, j \bmod \sqrt{N}, *, *)$.

Step 3: Combine and rearrange the tokens in each group so that the token for $H[r, j]$ is in processor $(r \bmod \sqrt{N}, j \bmod \sqrt{N}, \lfloor r/\sqrt{N} \rfloor, \lfloor j/\sqrt{N} \rfloor)$.

Step 4: Perform an OTIS move on the tokens. Now, the token for $H[r, j]$ is in $(\lfloor r/\sqrt{N} \rfloor, \lfloor j/\sqrt{N} \rfloor, r \bmod \sqrt{N}, j \bmod \sqrt{N})$.

Steps 1 and 3 can each be done in $4\sqrt{N}+o(\sqrt{N})$ electronic moves using the mesh integer sorting algorithm of Krizanc [20]. So the total number of moves needed to combine the results for $\sqrt{N}$ angles is $8\sqrt{N} + o(\sqrt{N})$ electronic plus 2 OTIS. For all p angles, the number of moves to do the combining becomes $8p + o(\sqrt{N})$ electronic and $2p/\sqrt{N}$ OTIS. Adding in the time spent running the pipelined algorithm of Section 5.2, we get a total count of $12\sqrt{N} + 12p - 28$ electronic moves and $2p/\sqrt{N}$ OTIS moves.

6 Conclusion

We have improved upon the $N \times N$ mesh Hough transform algorithms of [14] and [4]. We are able to compute the p angle Hough transform using $8(N-1)+8(p/4-1)$ moves whereas the algorithm of [14] takes $12(N-1)+8(p/4-1)$ moves and that of [4] takes $48N+20p+4$ moves (exclusive of the final sort step). The p angle Hough transform algorithms takes $28N + 8p - 60$ electronic and $8N + 2p - 16$ OTIS moves when the GRM mapping is used, and $12\sqrt{N} + 4p - 28$ electronic moves when GSM mapping is used (both exclusive of the sort/combine step).

The histogramming algorithm we developed takes $4(\sqrt{N} - 1) + B - 1$ electronic and 2 OTIS moves when $0 < B \leq \sqrt{N}$, $22\sqrt{N} + O(N^{3/8})$ electronic and $O(N^{3/8})$ OTIS moves when $B > N$, and when $\sqrt{N} < B \leq N$, $6\sqrt{N}+o(\sqrt{N})$ electronic moves and 2 OTIS moves for $O(1)$ memory per processor, and $4\sqrt{N} + 2\sqrt{B} - 6$ electronic and 2 OTIS moves for $O(\sqrt{B})$ memory per processor.

For histogram modification, our algorithm for $O(1)$ memory per processor case takes $28\sqrt{N} + o(\sqrt{N})$ electronic and 2 OTIS moves, and that for the $O(\sqrt{N})$ memory per processor case takes $9(\sqrt{N}-1)$ electronic and 2 OTIS moves.

Our algorithm for image shrinking and expanding takes $10(\sqrt{N}-1) + 8q_f + 8q_m + 4$ electronic and 2 OTIS moves when the GRM mapping is used and $8\sqrt{N} + 4q_f + 4q_m - 4$ electronic and 8 OTIS moves when the GSM mapping is used.

References

[1] T. Bestul and L. S. Davis. On computing complete histograms of images in $\log(n)$ steps using hypercubes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(2):212–213, 1989.

[2] H. Y. H. Chaung and C. C. Li. A systolic processor for straight line detection by modified Hough transform. In *IEEE Workshop on Computer Architecture, Pattern Analysis and Data Base Management*, pages 300–303, Las Alamitos, California, 1985. IEEE Computer Society Press.

[3] A. N. Choudhary and R. Ponnusamy. Implementation and evaluation of Hough transform algorithms on a shared-memory multiprocessor. *Journal of Parallel and Distributed Computing*, 12(2):178–188, 1991.

[4] R. E. Cypher, J. L. C. Sanz, and L. Snyder. The Hough transform has $o(n)$ complexity on SIMD $n \times n$ mesh array architecture. In *IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 115–121, Seattle, Washington, 1987.

[5] M. Feldman, S. Esener, C. Guest, and S. Lee. Comparison between electrical and free-space optical interconnects based on power and speed considerations. *Applied Optics*, 27(9):1742–1751, May 1988.

[6] A. Fisher and P. Highnam. Computing the Hough transform on a scan line array processor. In *IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 83–87, Seattle, Washington, 1987.

[7] J. Grinberg, G. R. Nudd, and R. D. Etchells. A cellular VLSI architecture. *IEEE Computer*, 17(1):69–81, Jan. 1984.

[8] C. Guerra and S. Hambrusch. Parallel algorithms for line detection on a mesh. In *IEEE 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 99–106, Seattle, Washington, 1987.

[9] W. Hendrick, O. Kibar, P. Marchand, C. Fan, D. V. Blerkom, F. McCormick, I. Cokgor, M. Hansen, and S. Esener. Modeling and optimization of the optical transpose interconnection system. In *Optoelectronic Technology Center, Program Review*, Cornell University, Sept. 1995.

[10] H. A. Ibrahim, J. B. Kender, and D. E. Shaw. On the application of massively parallel SIMD tree machine to certain intermediate-level vision tasks. *Computer Vision, Graphics, and Image Processing*, 36:53–75, 1986.

[11] J. Illingworth and J. Kittler. A survey of Hough transform. *Computer Vision, Graphics, and Image Processing*, 44, 1988.

[12] J.-W. Jang, H. Park, and V. K. Prasanna. A fast algorithm for computing histogram on reconfigurable mesh. In *Proceedings of the Symposium on the Frontiers of Massively Parallel Computation*, McLean, Virginia, 1992.

[13] J.-F. Jenq and S. Sahni. Reconfigurable mesh algorithms for image shrinking, expanding, clustering, and template matching. In *Proceedings of the 5th International Parallel Processing Symposium*, pages 208–215, Las Alamitos, California, 1991. IEEE Computer Society Press.

[14] J.-F. Jenq and S. Sahni. Reconfigurable mesh algorithms for the Hough transform. In *Proceedings of 1991 International Conference on Parallel Processing*, pages 34–41, New York, 1991. Academic Press.

[15] J.-F. Jenq and S. Sahni. Histogramming on a reconfigurable mesh computer. In *Proceedings of the 6th International Parallel Processing Symposium*, pages 425–432, Beverly Hills, California, 1992.

[16] J.-F. Jenq and S. Sahni. Image shrinking and expanding on a pyramid. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1291–1296, Nov. 1993.

[17] C. S. Kannan and H. Y. H. Chuang. Fast Hough transform on a mesh connected processor array. *Information Processing Letters*, 33:243–248, Jan. 1990.

[18] F. Kiamilev, P. Marchand, A. Krishnamoorthy, S. Esener, and S. Lee. Performance comparison between optoelectronic and vlsi multistage interconnection networks. *Journal of Lightwave Technology*, 9(12):1674–1692, Dec. 1991.

[19] A. Krishnamoorthy, P. Marchand, F. Kiamilev, and S. Esener. Grain-size considerations for optoelectronic multistage interconnection networks. *Applied Optics*, 31(26):5480–5507, Sept. 1992.

[20] D. Krizanc. Integer sorting on a mesh-connected array of processors. Manuscript, 1989.

[21] H. Li, M. A. Lavin, and L. R. Le Master. Fast Hough transform: A hierarchical approach. *Computer Vision, Graphics, and Image Processing*, 36:139–161, Dec. 1986.

[22] H. Li and Maresca. Polymorphic-torus architecture for computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(3), Mar. 1989.

[23] H. F. Li, D. Pao, and R. Jayakumar. Improvements and systolic implementation of the Hough transform for straight line detection. *Pattern Recognition*, 22(6):697–706, 1989.

[24] Maresca, M. A. Lavin, and H. Li. Parallel Hough transform algorithms on polymorphic torus. In Levialdi, editor, *High Level Vision in Multicomputers*. Academic Press, New York, 1988.

[25] Maresca, H. Li, and Sheng. Parallel computer vision on polymorphic torus architecture. *International journal of Computer Vision and Applications*, 2(4), 1989.

[26] G. C. Marsden, P. J. Marchand, P. Harvey, and S. C. Esener. Optical transpose interconnection system architectures. *Optics Letters*, 18(13):1083–1085, July 1 1993.

[27] D. Nassimi and S. Sahni. Data broadcasting in SIMD computers. *IEEE Transactions on Computers*, C-30(2):101–107, Feb. 1981.

[28] M. Nigam and S. Sahni. Sorting n^2 numbers on $n \times n$ meshes. In *Proceedings of the seventh International Parallel Processing Symposium (IPPS'93)*, pages 73–78, Newport Beach, California, 1993.

[29] S. Olariu, J. L. Schwing, and J. Zhang. Computing the Hough transform on reconfigurable meshes. In *Proceedings of Conference on Vision Interface'92*, pages 169–174, 1992.

[30] S. Pavel and S. G. Akl. Efficient algorithms for the Hough transform on arrays with reconfigurable optical buses. In *Proceedings of the 10th International parallel Processing Symposium (IPPS'96)*, pages 697–701, Honolulu, Hawaii, 1996.

[31] T. Pavlidis. *Algorithms for Graphics and Image Processing*. Computer Science Press, Maryland, 1982.

[32] S. Ranka and S. Sahni. *Hypercube Algorithms with Applications to Image Processing and Pattern Recognition*. Springer-Verlag, 1990.

[33] A. Rosenfeld. A note on shrinking and expanding operations in pyramids. *Patter Recognition Letters*, 6(4):241–244, 1987.

[34] A. Rosenfeld, J. Ornelas, Jr., and Y. Hung. Hough transform algorithms for mesh-connected SIMD parallel processors. *Computer Vision, Graphics, and Image Processing*, 41:293–305, 1988.

[35] S. Sahni and C.-F. Wang. BPC permutations on the OTIS-Mesh optoelectronic computer. In *Proceedings of the fourth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'97)*, pages 130–135, Montreal, Canada, 1997.

[36] H. J. Siegel, J. Siegel, F. C. Kemmerer, P. T. Muller, H. E. Smalley, and D. D. Smith. PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition. *IEEE Transactions on Computers*, C-30(12):934–947, Dec. 1981.

[37] T. M. Silberberg. The Hough transform in the geometric arithmetic parallel processor. In *IEEE Workshop on Computer Architecture and Image Database Management*, pages 387–391, Las Alamitos, California, 1985. IEEE Computer Society Press.

[38] S. L. Tanimoto. Sorting, histogramming and other statistical operations on a pyramid machine. In A. Rosenfeld, editor, *Multiresolution Image Processing and Analysis*, pages 136–145. Springer-Verlag, New York, 1984.

[39] M. J. Thazhuthaveetil and A. V. Shah. Parallel Hough transform algorithm performance. *Image and Vision Computing*, 9(2):88–92, 1991.

[40] C.-F. Wang and S. Sahni. Basic operations on the OTIS-Mesh optoelectronic computer. Technical Report 97-029, CISE Department, University of Florida, Gainesville, Florida, 1997. Available by anonymous ftp login from ftp.cise.ufl.edu under directory tech-report/tr97/tr97-029.ps.gz.

[41] C.-F. Wang and S. Sahni. Basic operations on the OTIS-Mesh optoelectronic computer. In *Proceedings of the fifth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'98)*, pages 150–157, Las Vegas, Nevada, 1998.

[42] C.-F. Wang and S. Sahni. Matrix multiplication on the otis-mesh optoelectronic computer. Technical report, CISE Department, University of Florida, Gainesville, Florida, 1998.

[43] M. Yasrebi, S. Deshpande, and J. C. Browne. A comparison of circuit switching and packet switching data transfer using two simple image processing algorithms. In *Proceedings of 1983 International Conference on Parallel Processing*, pages 25–28, Alamitos, California, 1983. IEEE Computer Society Press.

[44] F. Zane, P. Marchand, R. Paturi, and S. Esener. Scalable network architectures using the optical transpose interconnection system (OTIS). In *Proceedings of the second International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96)*, pages 114–121, San Antonio, Texas, 1996.