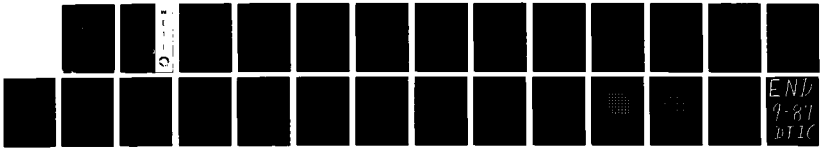
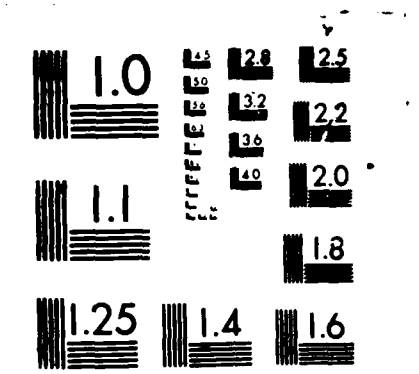
AD-A183 756 ON COMPUTING HISTOGRAMS OF IMAGES IN LOG N TIME USING FAT PYRAMIDS..(U) MARYLAND UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH T BESTUL ET AL. FEB 87 CAR-TR-271 1/1

UNCLASSIFIED ETL-0454 DACA76-84-C-0004 F/G 12/1 NL



END
9-87
DTIC



ETL-0454

On computing histograms of images in log η time using fat pyramids

Thor Bestul
Larry S. Davis

Computer Vision Laboratory
Center for Automation Research
University of Maryland
College Park, MD 20742-3411

February 1987

Prepared for
U.S. ARMY CORPS OF ENGINEERS
ENGINEER TOPOGRAPHIC LABORATORIES
FORT BELVOIR, VIRGINIA 22060-5546

E T L

US ARMY ENGINEER TOPOGRAPHIC LABORATORIES ETL

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-271 CS-TR-1791	5. MONITORING ORGANIZATION REPORT NUMBER(S) ETL-0454 N/A
6a. NAME OF PERFORMING ORGANIZATION University of Maryland	6b. OFFICE SYMBOL (If applicable) N/A
7a. NAME OF MONITORING ORGANIZATION Army Engineer Topographic Laboratories	
6c. ADDRESS (City, State and ZIP Code) Center for Automation Research College Park, MD 20742-3411	7b. ADDRESS (City, State and ZIP Code) Fort Belvoir, VA 22060
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Defense Advanced Research Projects Agency	8b. OFFICE SYMBOL (If applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DACA76-84-C-0004	
8c. ADDRESS (City, State and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22209	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT NO.
11. TITLE (Include Security Classification) On Computing Histograms of Images in log n Time using Fat Pyramids	
12. PERSONAL AUTHOR(S) Thor Bestul and Larry S. Davis	
13a. TYPE OF REPORT Technical	13b. TIME COVERED FROM _____ TO N/A
14. DATE OF REPORT (Yr., Mo., Day) February 1987	15. PAGE COUNT 25
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper presents an algorithm for the log n computation of the complete histogram of an nxn gray-level image. It uses a "fat" pyramid implemented on an SIMD hypercube multiprocessor with very high processor utilization. A "fat" pyramid is a pyramid in which the size of a processor associated with a node in the pyramid depends on the level of the pyramid in which the node appears. We describe how to embed fat pyramids in hypercubes using Gray codes, and then describe the histogramming algorithm.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

1. Introduction

This paper is concerned with an algorithm for the $\log n$ time computation of the complete histogram of an $n \times n$ gray-level array. It makes use of a "fat" pyramid implemented on an SIMD hypercube multiprocessor with very high processor utilization. These terms are explained below.

In [2] Tanimoto describes an algorithm for histogram computation on a standard pyramid which involves performing one $\log n$ time "pass" from the bottom of the pyramid to the top for each gray-level value to be computed. The algorithm presented below for fat pyramids computes the complete histogram in a single pass.

The algorithm is perhaps more properly called a hypercube multiprocessor algorithm. However, the concept of the fat pyramid provides a convenient way of describing the computations involved.

2. SIMD Hypercube Multiprocessors

A hypercube multiprocessor consists of 2^M (for some M) processing cells interconnected as if each were located at one vertex of an M-dimensional hypercube, so that any two cells share a direct connection if and only if their corresponding hypercube vertices are connected by a hypercube edge. Furthermore, if the hypercube has unit side, and we assign to each cell an address given by the M-dimensional coordinates of the corresponding vertex, then we can see that two cells will share a direct connection if and only if their addresses differ in exactly one bit position.

In a SIMD (Single-Instruction stream, Multiple-Data stream) multiprocessor, all processing cells execute in unison a stream of commands broadcast from a single controller. The exception to this is that some subset of the cells may ignore a given instruction, the decision to do so being based on the contents of their local cell memory.

3. Fat Pyramids

A grid multiprocessor consists of a rectangular array of nodes, with a processing cell at each node, and with each node directly connected to its four nearest neighbors (except on the edges of the grid). In a pyramid multiprocessor, there are several levels of grids, each containing (say) one-fourth as many nodes as the one below, and with each node connected directly to four nodes in the level below. This arrangement allows for some combinations of the information in all of the nodes at the bottom level to be computed in $\log n$ time, where the bottom level is (say) an $n \times n$ square, by combining at each node on level $l+1$ the information from that node's four children on level l (level 0 is the bottom level).

In a "fat" pyramid, the size of a processor associated with a node in the pyramid depends on the level of the pyramid in which it appears. Specifically, a processor at level $l+1$ will have four times as much storage, and possibly four times as much processing power, as a processor at level l. A fat pyramid allows for combining operations in which the amount of information per node increases with the level of the node in the pyramid.

4. Fat Pyramids and Hypercubes

The pyramid algorithm presented below has the property that only one level is "active" at any given time; that is, while a computation is being carried out on a given level, no other level needs to perform a computation. Many other pyramid algorithms also have this property. In such cases, it does not seem desirable to have processors lying idle when the level with which they are associated is not active. Suppose instead that some multiprocessor system were multiplexed among all the levels, with all of its processing cells being divided up among the nodes of a given level when that level is active. Thus if one cell were associated with each node at level 0 of the pyramid, there would be four cells cooperating to carry out the computations for each node at level 1, sixteen cells grouped to perform the operations required by each node at level 2, and in general 4^l cells associated with each pyramid node at level l. I will call such an arrangement a "collapsed" fat pyramid.

In order for several cells to cooperate efficiently in performing some computation, it would be useful if there were a high degree of interconnectivity among them. Just such an arrangement can be nicely provided on a hypercube. At any level l of the pyramid, all of the cells of the hypercube are divided up among the nodes at that level, in blocks of 4^l cells per node. It is possible to do this so that the block of cells implementing each node are themselves connected in a hypercube of dimension $2l$; thus none of the 4^l cells in a block are more than $2l$ links apart. Also, if we define two processor blocks to be adjacent if any cell in one block is directly connected to some cell in the other block, then we can also

arrange for all of the blocks at any given level to form a rectangular grid under this kind of adjacency, which is of course what we desire. Further discussion of such an arrangement follows.

5. Gray Codes and Grids

In [1] Chan and Saad describe how to embed a grid in a hypercube by the use of *Gray Codes*, which are number sequences in which the binary representations of two adjacent elements differ in exactly one bit position. The Gray Code used in what follows will be the *binary reflected Gray Code*, for which the i-bit sequence is obtained by appending a reversed copy of the $i-1$-bit sequence to itself with each element prefixed by a 1. (So, for example, the 2-bit binary reflected Gray Code is {00, 01, 11, 10}, and the 3-bit sequence is {000, 001, 011, 010, 110, 111, 101, 100}.)

Consider a cell with grid coordinates (x,y), each of which have p bits, and let

$$g_x = g_{x,p-1}g_{x,p-2}\cdots g_{x,0}$$

and

$$g_y = g_{y,p-1}g_{y,p-2}\cdots g_{y,0}$$

be the x^{th} and y^{th} elements of the p-bit binary reflected Gray Code, respectively. Then the hypercube address of this cell is taken as the concatenation

$$g_{x,p-1}g_{x,p-2}\cdots g_{x,0}g_{y,p-1}g_{y,p-2}\cdots g_{y,0}.$$

It is not hard to see that the hypercube addresses of two cells adjacent in the grid will have addresses differing in exactly one bit position. Thus any two such

cells will also be adjacent in the hypercube. This mapping is illustrated in Figure 1.

6. A New Mapping

If we instead *interleave* the bits of g_x and g_y, giving

$$g_{x,p-1}g_{y,p-1}g_{x,p-2}g_{y,p-2}\cdots g_{x,0}g_{y,0},$$

then we get not only a mapping from a hypercube to a grid which preserves adjacency, but we also get a natural hypercube-to-collapsed-fat-pyramid mapping in the following sense. This mapping allows the hypercube address of any cell in any block at any level l to be regarded as the concatenation of two binary numbers, the first one being a $2(L-l)$ bit number (where L is the number of levels in the pyramid) determining which processor block the cell is in, and the other being a $2l$ bit number giving a local address of the cell within its block. In other words, all of the cells comprising any block on any level l will have hypercube addresses which are all in the range $k4^l$ to $(k+1)4^l-1$ for some k. This is illustrated in Figures 2 through 4. (Notice that this property does *not* hold for the mapping depicted in Figure 1.)

Consider, for example, cell 57 in Figure 3, which depicts a four-level collapsed fat pyramid at level 1. Now, 57 in binary is 111001, so this is cell 01 within processor block 1110. Notice also that if we apply in reverse the mapping described above to just the number 1110, we get (2,3) (de-interleaving the bits of 1110 gives 11 and 10, and these are the two-bit binary reflected Gray Code values g_2 and g_3, respectively), which is precisely the position within the level 1 grid of

the block containing cell 57.

The property that a cell's hypercube address is also simply the concatenation of its block number and local address is not essential to the algorithm presented below, but greatly simplifies its expression.

7. A Sketch of the Histogram Algorithm

Given an $n \times n$ array of gray-level values, the problem is to compute a histogram of these values over the array. Suppose that each gray-level value is stored in one cell at the bottom level of a fat pyramid implemented on a hypercube as described above.

Essentially, each block of processors at a given level will contain the histogram for the gray-level values of the cells in that block. Processing proceeds from one level to the next by combining the histograms of the four child blocks to make up the histogram of the parent block. (Although this seems similar to a divide-and-conquer algorithm on a grid, the two schemes are different in that the cells constituting each block on any level of the collapsed fat pyramid are interconnected to form a hypercube. Furthermore, there is greater interconnectivity between adjacent blocks in the collapsed fat pyramid.)

Suppose that the gray-level values are s bits long (assume s even). The algorithm consists of two different phases, one for levels below $s/2$, and one for the higher levels. This is because at levels below $s/2$, the width of a cell's local address is less than the width of a gray-level value. The significance of this fact will become clear below.

Let the current level be $l < s/2$. Then the algorithm works as follows. (Assume in what follows that k always represents $L-l$.) Consider four blocks B_0, B_1, B_2, and B_3 at level l, which are to be combined into a larger block B at level $l+1$. Consider a cell in any B_i with local address c. Let its complete address be $b_{2k-1}b_{2k-2}\cdots b_0 c_{2l-1}c_{2l-2}\cdots c_0$. We assume at this point in the algorithm that this cell contains the histogram information over B_i for all gray-level values of the form

$$g_{s-1}g_{s-2}\cdots g_{2l}c_{2l-1}c_{2l-2}\cdots c_0,$$

that is, ending in c. (Efficient schemes for doing this are disussed below. It turns out that for a pyramid with 2^{20} nodes on the bottom level, each containing an 8-bit gray-level value, no level of the pyramid requires more than 36 bits per cell for the storage of all necessary histogram information.) Once the blocks are merged to form B, we will want any cell with address

$$b_{2k-1}b_{2k-2}\cdots b_2 c_{2l+1}c_{2l}c_{2l-1}c_{2l-2}\cdots c_0$$

to contain the histogram information over B for all gray-level values of the form

$$g_{s-1}g_{s-2}\cdots g_{2l+2}c_{2l+1}c_{2l}c_{2l-1}c_{2l-2}\cdots c_0.$$

Consider, for example, a cell in B_0, with local address c. Let its complete address be

$$b_{2k-1}b_{2k-2}\cdots b_2 00 c_{2l-1}c_{2l-2}\cdots c_0.$$

This cell must collect the histogram information over each B_i for all the gray-level values of the form

$$g_{s-1}g_{s-2}\cdots g_{2l+2}00c_{2l-1}c_{2l-2}\cdots c_0.$$

Fortunately, all of this information is distributed across the cells at

$$b_{2k-1}b_{2k-2}\cdots b_200c_{2l-1}c_{2l-2}\cdots c_0$$

which is the cell itself,

$$b_{2k-1}b_{2k-2}\cdots b_201c_{2l-1}c_{2l-2}\cdots c_0,$$

which is a cell in block B_1,

$$b_{2k-1}b_{2k-2}\cdots b_210c_{2l-1}c_{2l-2}\cdots c_0,$$

which is a cell in block B_2, and

$$b_{2k-1}b_{2k-2}\cdots b_211c_{2l-1}c_{2l-2}\cdots c_0,$$

which is a cell in block B_3. This is because no other cells in B have addresses ending in c, and so no other cells in B can contain histogram information over B for gray-level values ending in c. But we see by inspecting the addresses of the four cells given above that they are connected in a square by hypercube edges. So it will take only two parallel transmission operations to perform the necessary redistribution of the histogram information.

At level $l = s/2$, consider a cell in block B with local address c. This cell will contain the histogram information over B for all gray-level values ending in c. But c is $2l = s$ bits long. Thus the cell contains only the histogram information over B for the single gray-level value c.

For levels $l \geq s/2$, the algorithm works as follows. Within a block B at some level l, we want any cell whose address ends in

$$g = g_{s-1}g_{s-2}\cdots g_0$$

to contain the histogram information over B for the single gray-level value g. We can see that this is true when we reach level $s/2$. Now consider the four level l blocks B_0, B_1, B_2, and B_3. Let the addresses of these blocks be

$$b_{2k-1}b_{2k-2}\cdots b_200,$$
$$b_{2k-1}b_{2k-2}\cdots b_201,$$
$$b_{2k-1}b_{2k-2}\cdots b_210,$$

and

$$b_{2k-1}b_{2k-2}\cdots b_211,$$

respectively.

Suppose that these four blocks are to be combined to form the level $l+1$ block B. We must combine the histogram information from each of those blocks to form the histogram for B.

For example, the histogram information over B_0 for the gray-level value

$$g = g_{s-1}g_{s-2}\cdots g_0$$

is contained at any cell in B_0 with local address ending in g, that is, with complete address

$$b_{2k-1}b_{2k-2}\cdots b_200c_{2l-1}c_{2l-2}\cdots c_s g_{s-1}g_{s-2}\cdots g_0$$

for some c_i's. But we know also that the histogram information over B_1, B_2, and B_3 for the gray-level value g can be found in

$$b_{2k-1}b_{2k-2}\cdots b_201c_{2l-1}c_{2l-2}\cdots c_s g_{s-1}g_{s-2}\cdots g_0,$$
$$b_{2k-1}b_{2k-2}\cdots b_210c_{2l-1}c_{2l-2}\cdots c_s g_{s-1}g_{s-2}\cdots g_0,$$

and

$$b_{2k-1}b_{2k-2}\cdots b_{2l}1c_{2l-1}c_{2l-2}\cdots c_s g_{s-1}g_{s-2}\cdots g_0,$$

respectively. These four cells are connected in a square by hypercube edges. So in two parallel transmission operations the cell in each block can collect the histogram information from the cells in the other three blocks, and can compute the histogram information over the entire block B for the gray-level value g.

8. The Algorithm in More Detail

Suppose that the current level is $l < s/2$. Consider a block B at this level, and a cell whose local address within B is

$$c = c_{2l-1}c_{2l-2}\cdots c_0.$$

The histogram to be stored in cell c will be stored as a mapping from

$$g_{s-1}g_{s-2}\cdots g_{2l}$$

to the number of cells in B containing the gray-level value

$$g_{s-1}g_{s-2}\cdots g_{2l}c_{2l-1}c_{2l-2}\cdots c_0.$$

That is, the histogram will actually be stored as a mapping from the first $s-2l$ bits of a gray-level value to the histogram value over B for that gray-level value, with the lower $2l$ bits of the gray-level value being given implicitly by the local address of cell c within block B.

When combining blocks from level l to form larger blocks at level $l+1$, the histograms are combined in the following way. Each cell first splits its histogram mapping into two smaller mappings by splitting the mapping's domain into even and odd subsets. Then the last bit is truncated from each element in the

domains of both new mappings. (For example, if the original mapping was $\{(0,m_0), (1,m_1), (2,m_2), (3,m_3), (4,m_4), (5,m_5), (6,m_6), (7,m_7)\}$, then the two new mappings would be $\{(0,m_0), (1,m_2), (2,m_4), (3,m_6)\}$ and $\{(0,m_1), (1,m_3), (2,m_5), (3,m_7)\}$.) Call these the "even" and "odd" mappings, respectively.

Now every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots b_1 0 c_{2l-1}c_{2l-2}\cdots c_0$$

sends its *odd* mapping to the cell at address

$$b_{2k-1}b_{2k-2}\cdots b_1 1 c_{2l-1}c_{2l-2}\cdots c_0,$$

and every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots b_1 1 c_{2l-1}c_{2l-2}\cdots c_0$$

sends its *even* mapping to the cell at address

$$b_{2k-1}b_{2k-2}\cdots b_1 0 c_{2l-1}c_{2l-2}\cdots c_0.$$

Each cell which receives an even mapping adds (element-wise) this mapping to its own even mapping. Similarly, each cell which receives an odd mapping adds that mapping to its own odd mapping. Notice that the histograms stored in each cell now have domains half the size of what they were previously.

Each cell now splits its histogram into even and odd mappings as before. Then every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots 0 b_0 c_{2l-1}c_{2l-2}\cdots c_0$$

sends its *odd* mapping to the cell at address

$$b_{2k-1}b_{2k-2}\cdots 1 b_0 c_{2l-1}c_{2l-2}\cdots c_0.$$

and every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots 1b_0c_{2l-1}c_{2l-2}\cdots c_0,$$

sends its *even* mapping to the cell at address

$$b_{2k-1}b_{2k-2}\cdots 0b_0c_{2l-1}c_{2l-2}\cdots c_0.$$

Mappings are again combined as described above. At this point each cell contains a mapping whose domain is one fourth the size of the domain of its original mapping. Furthermore, a cell at address

$$b_{2k-1}b_{2k-2}\cdots b_2c_{2l+1}c_{2l}c_{2l-1}\cdots c_0,$$

where

$$b=b_{2k-1}b_{2k-2}\cdots b_2$$

is the address of the level $l+1$ block B containing the cell

$$c=c_{2l+1}c_{2l}c_{2l-1}\cdots c_0$$

will now contain the mapping from

$$g_{s-1}g_{s-2}\cdots g_{2l+2}$$

to the number of cells in block B containing gray-level value

$$g_{s-1}g_{s-2}\cdots g_{2l+2}c_{2l+1}c_{2l}c_{2l-1}\cdots c_0.$$

For levels at or above $s/2$, the algorithm becomes a simpler special case of the algorithm for the lower levels. Assuming the current level is $l \geq s/2$, we combine histograms in the following way. Every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots b_10c_{2l-1}c_{2l-2}\cdots c_0$$

sends its histogram value to the cell at address

$$b_{2k-1}b_{2k-2}\cdots b_1 1 c_{2l-1}c_{2l-2}\cdots c_0$$

and vice-versa. Each cell then adds its received value to its stored value. Then every cell with address of the form

$$b_{2k-1}b_{2k-2}\cdots 0 b_0 c_{2l-1}c_{2l-2}\cdots c_0$$

sends its value to cell

$$b_{2k-1}b_{2k-2}\cdots 1 b_0 c_{2l-1}c_{2l-2}\cdots c_0$$

and vice-versa, and each cell then again adds its received value to its stored one, completing the computation for level $l+1$.

Once the top level is reached, any cell with address of the form

$$c_{2L-1}c_{2L-2}\cdots c_s g_{s-1}g_{s-2}\cdots g_0$$

will contain the total number of cells having gray-level value

$$g_{s-1}g_{s-2}\cdots g_0.$$

9. Efficiently Storing the Histogram Mappings

Suppose mappings are stored as sets of ordered pairs indicating which elements in the domain map to non-zero values, and what those values are. Then we see that the number of ordered pairs stored in a given cell is bounded above by two quantities as follows.

First, the histogram over a block with 4^l cells cannot have more than 4^l gray-levels mapping to non-zero values, since there are at most this many distinct gray-levels represented somewhere in the block. So the number of pairs stored in

a cell at level l cannot exceed 4^l.

Secondly, if each cell only stores pairs for those gray-level values which end in the local address of that cell within its block, then if the local address of the cell is $2l$ bits long and the gray-level values are s bits long, the number of pairs stored in a cell cannot exceed 2^{s-2l}.

Thus the maximum number of pairs stored in a cell will be $\sqrt{2^s}$, that is, the square root of the range of the gray-levels, and this maximum will occur when $l = s/4$. However, we also notice that the total of the histogram values in the pairs in some cell at level l cannot exceed 4^l, since the block containing this cell has only 4^l cells contributing to the histogram. We can use this fact at levels near $s/4$ to encode the mappings in cells at these levels actually using fewer bits than are required by the ordered-pair representation, as will be demonstrated in the example below.

Let us consider the case of a pyramid with 2^{20} cells at the bottom level, each cell containing an eight bit gray-level value. At level 0 of the pyramid, each processor block contains a single cell. The histogram mapping for any cell then is given by the single pair $(g,1)$, or really just the value g, requiring eight bits.

At level 1 of the pyramid, each processor block contains four cells. The histogram mapping for a block is stored as four pieces, one in each cell. The piece in each cell can be given as a mapping from six-bit gray-level values to three-bit counts. The mappings are from six-bit values because the remaining two bits of the gray-level values in the mappings in any cell are the same as the last two bits of the hypercube address of that cell. The mappings are to three-bit values

because within a block there can be no more than four cells with the same gray-level value. Also, no mapping piece can have a domain with size greater than four, since there are at most four distinct gray-level values in a block of four cells. Thus four pairs in each cell, each pair requiring nine bits, will hold the mapping for the block. (Note that we could actually hold the mapping with merely one pair per cell. But then we would lose the relationship between the gray-level values stored at a given cell and the hypercube address of that cell, which is essential in obtaining the speed of the algorithm. Furthermore, we will see that 36 bits per cell are enough to hold all of the histogram information for any level of the pyramid.)

We now skip to level 3. At this level, each block contains 64 cells. Any cell with local address

$$c = c_5 c_4 \cdots c_0$$

will contain the histogram information for all gray-level values of the form

$$g_7 g_6 c_5 c_4 \cdots c_0.$$

Of course, there are only four gray-level values of this form. Also, within a block at this level, no more than 64 cells can have the same gray-level value. Thus the mapping in cell c can be given as four ordered pairs, with the first element of each pair being a two-bit gray-level value (the remaining six bits given implicitly by c), and the second element of each pair being a six-bit count. (Actually, the mapping could simply be stored as an array of four six-bit numbers, so the first element of each pair is really unnecessary.)

At levels 4 and above, any cell holds the histogram value for only one gray-level value. That gray-level value is given by the last eight bits of the cells hypercube address, so the cell need only contain a count, which can require no more than 20 bits.

The most difficult case is that at level 2, where each block contains 16 cells. Consider a block B containing a cell with local address

$$c = c_3 c_2 c_1 c_0$$

This cell must contain the histogram information over B for all gray-level values of the form

$$g_7 g_6 g_5 g_4 c_3 c_2 c_1 c_0.$$

There are 16 such gray-level values, and for each of them there is a histogram value which can be anywhere from 0 to 16, since it is possible that all cells in B have the same gray-level value. However, we notice that the sum of these histogram values cannot exceed 16, since together they represent part of a histogram for a block of 16 cells. So we can encode the mapping as a 32-bit binary number containing exactly 16 1s, and in which the number of 0s between the n^{th} and $n+1^{\text{st}}$ 1s gives the mapping value for n. The 32 bit encoding of this mapping seems reasonable, since from a theoretical standpoint, no encoding for the mapping could use fewer than 29 bits.

So we see that in this case the histogram information at any level of the pyramid can be stored using no more than 36 bits per cell.

10. Summary and Comments

This paper has presented a $\log n$ time algorithm for the computation of the histogram of a collection of values distributed one per cell across a hypercube multiprocessor. It uses the technique of regarding the hypercube as a multiplexed implementation of a "fat" pyramid, in which the processor at each pyramid node is larger than the processors at that node's children.

The algorithm requires at most $\sqrt{2^s}$ space per cell where s is the number of bits required to store one of the histogram domain values, although much less storage may be necessary if appropriate encodings of intermediate results are used. Thus it is most appropriate for use where s is less than $2L$, that is, when the possible range of the values is less than the total number of values. Machine vision applications involving arrays of tens or hundreds of thousands of gray-level values quantized to a few hundred or thousand possible levels are natural applications for this algorithm.

In some cases it is desirable for the resulting histogram array to be stored such that values adjacent in the histogram array are located in adjacent processors. Consider the case of a two-dimensional array of gray-scale values, and regard this array as a mapping from position to value. We see that the gray-scale values are the *dependent* values of this mapping. But notice that their role changes to that of *independent* values when considering the histogram itself as a mapping. We were able to have the (dependent) gray-scale values of adjacent (independent) positions reside in adjacent processors by first transforming the coordinates for a given position to Gray Code values, interleaving (or concatenat-

ing) these transformed coordinates, and storing the corresponding gray-scale value in the processor whose address is given by the interleaved (or concatenated) Gray Code values.

In the computed histogram, the gray-scale values play the role of independent values of a mapping whose dependent values are indexed by processor addresses. This suggests by analogy how to obtain the desired adjacency of the (dependent) histogram values. Specifically, consider what happens if we transform each gray-scale value to its corresponding Gray Code value, before performing the histogram computation. Then, when the histogram computation is complete, the histogram value mapped to by some given gray-scale value can be found by converting that gray-scale value to a Gray Code value, and then accessing a cell whose address ends in the Gray Code value. Note that two sequential gray-scale values have corresponding Gray Code values differing in exactly one bit position, and so the histogram values mapped to by these two gray-scale values will be found in adjacent cells.

This idea generalizes easily to histograms with multi-dimensional domains, where it may be desired that the histogram values corresponding to adjacent values in the domain be found in adjacent cells. In this case, each of the elements in the histogram domain tuple in each cell is first converted to a Gray Code value. Then these Gray Code tuples are interleaved (or concatenated), and the histogram of the resulting values is computed. Now, in order to access the histogram value for a given domain tuple, the same transformation originally applied to the domain tuples is applied to the given tuple, and the corresponding

cell accessed. We see here that two adjacent domain tuples, that is, two tuples differing by one in one of their elements, will have converted (interleaved Gray Code) values differing in one bit position, and so will map to histogram values which are found in adjacent cells.

References

1. T.G. Chan and Y. Saad, "Multigrid Algorithms on the Hypercube Multiprocessor," *IEEE Transactions on Computers,* vol. C-35, no. 11, pp. 969–977, November 1986.

2. S.L. Tanimoto, "Sorting, Histogramming, and Other Statistical Operations on a Pyramid Machine," in *Multiresolution Image Processing and Analysis,* ed. A. Rosenfeld, pp. 136–145, Springer-Verlag.

7	4	12	28	20	52	60	44	36
6	5	13	28	21	53	61	45	37
5	7	15	31	23	55	63	47	39
4	6	14	30	22	54	62	46	38
3	2	10	26	18	50	58	42	34
2	3	11	27	19	51	59	43	35
1	1	9	25	17	49	57	41	33
0	0	8	24	16	48	56	40	32
	0	1	2	3	4	5	6	7

Figure 1

Hypercube To Grid Mapping

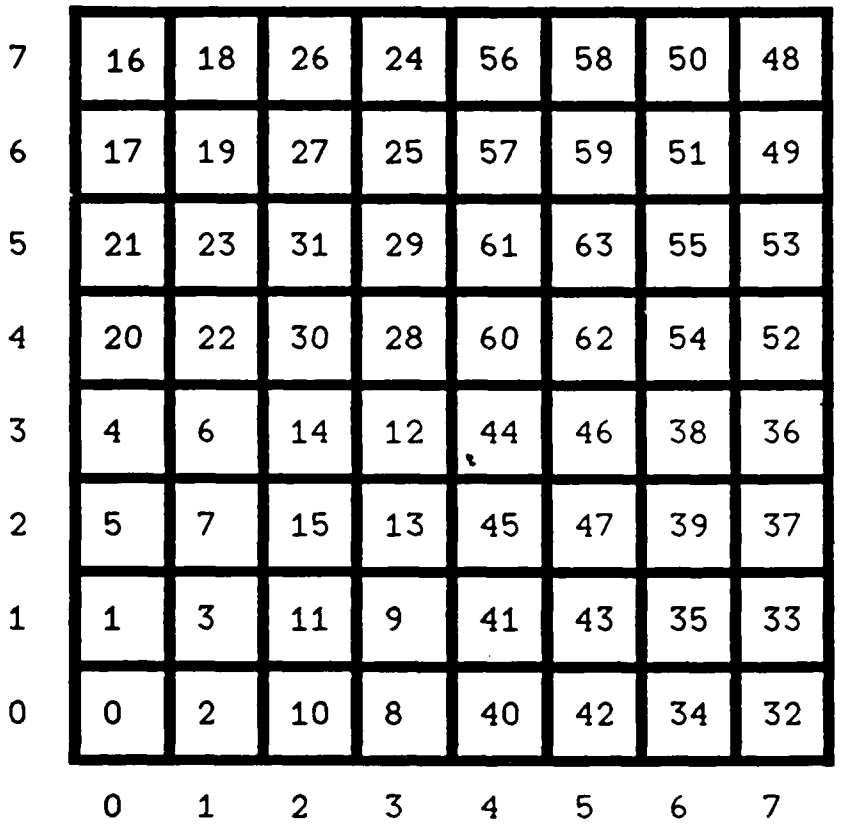


Figure 2

Hypercube To Fat Pyramid Mapping

Division Of Cells Among Nodes At Pyramid Level 0

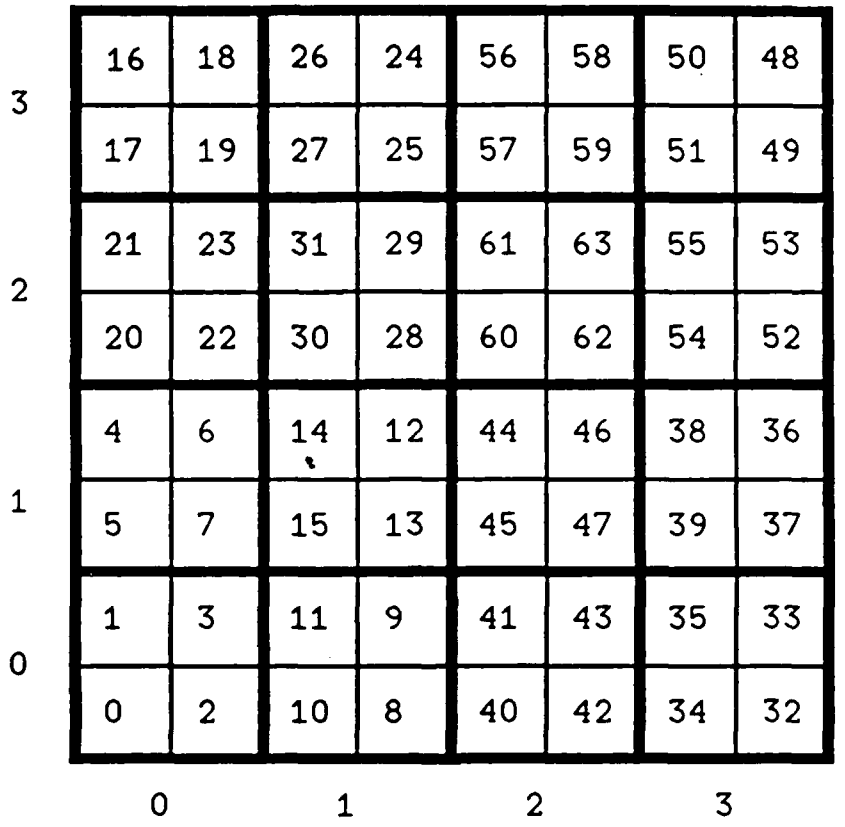


Figure 3

Hypercube To Fat Pyramid Mapping

Division Of Cells Among Nodes At Pyramid Level 1

	16	18	26	24	56	58	50	48
	17	19	27	25	57	59	51	49
1	21	23	31	29	61	63	55	53
	20	22	30	28	60	62	54	52
	4	6	14	12	44	46	38	36
	5	7	15	13	45	47	39	37
0	1	3	11	9	41	43	35	33
	0	2	10	8	40	42	34	32
		0				1		

Figure 4

Hypercube To Fat Pyramid Mapping

Division Of Cells Among Nodes At Pyramid Level 2

END

9-87

DTIC