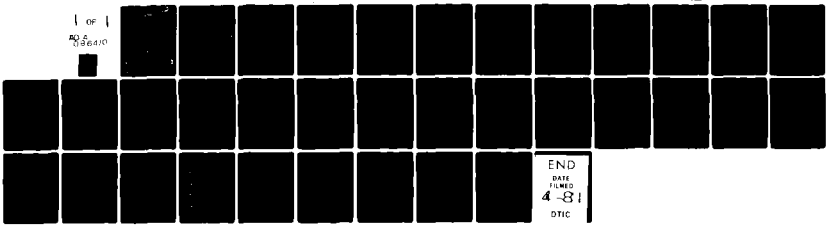
AD-A096 410 MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB F/G 9/2
IMAGE PROCESSING ON ZMOB.(U)
DEC 80 T KUSHNER, A Y WU, A ROSENFELD AFOSR-77-3271
UNCLASSIFIED TR-987 AFOSR-TR-81-0143 NL



AFOSR-TR- 81-0143

LEVEL II

COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

Approved for public release;
distribution unlimited.

(18) AFOSR

(17) TR-81-0143

(9) Technical rept.

(12)

LEVEL II

(14) TR-987

(15) AFOSR-77-3271

(11) December 1980

(11) 34

(6) IMAGE PROCESSING ON ZMOB

(10) Todd Kushner
Angela Y. Wu
Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, Maryland

(10) 2304

(17) A2

ABSTRACT

ZMOB is a multi-microprocessor system consisting of 256 Z80A microprocessors that communicate via a fast cyclic shift-register bus. This paper discusses the efficient use of ZMOB for various types of image processing operations, including point and local operations, discrete transforms, geometric operations, and computation of image statistics.

AIR FORCE ... (AFSC)
NOTICE ...
... and is
... AFR 190-12 (7b).
... Officer

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Sherry Palmer in preparing this paper.

*Also with the Department of Mathematics, Statistics, and Computer Science, American University, Washington, D.C.

1. Introduction

1.1 ZMOB

ZMOB [1] is a multi-microprocessor system with a ring-like inter-processor communication system called the "conveyor belt". The current configuration has 256 processors and is capable of executing a total of about 100 million instructions per second. This section explains features of the conveyor belt architecture exploited in processor communication.

The conveyor belt allows any processor to communicate with any other, at a speed so great that it is unnoticeable to the processor. Asynchronously, processors may compute data and pass intermediate results among each other. The conveyor belt also supports tightly-synchronized ("lock-step") parallel image processing algorithms by allowing processors to all communicate data in an organized way (e.g., a "pass-right" sequence) or rapidly pass blocks of data among one another (termed "burst mode").

However, in instruction-level lock-step mode (where absolute synchronous timing is crucial), not all patterns of data exchange can occur during an infinitesimal communication step. For example, no processor can receive data from more than one processor or send data to more than one specific processor at the same time.

Communication with the host VAX can occur simultaneously between all the processors (i.e., the VAX can send data to or receive data from all 256 processors at once), and all processors' computations may be synchronized by a command from the VAX. Thus, heavy simultaneous conveyor belt traffic must be organized and orchestrated carefully, and the VAX can provide the means to put the processors in synch to begin that activity properly.

1.2 Image Processing on ZMOB

This paper deals with the efficient use of ZMOB for performing various types of image processing operations, including point and local operations, discrete transforms, geometric operations, and computation of image statistics. The aim is to make the fullest possible use of ZMOB's parallelism, so as to achieve a speedup by a factor proportional to 256, the number of processors. To this end, we consider how the image data should be partitioned among the processors, and how the operations should be segmented into computation and communications steps. We also compare ZMOB processing with performing operations on the host VAX itself.

2. Point and Local Operations

A point operation on an image computes a new value for each pixel as a function of the old value, independent of the values of other pixels. To perform such an operation on ZMOB, the image is divided into 256 parts in any convenient way; each ZMOB processor receives one part from the host VAX and operates on its pixels; and the results are returned to the host VAX.

Let C_z and C_v be the times for a ZMOB processor and for the VAX, respectively, to perform the given operation on one pixel. Let N^2 be the number of pixels in the image, and let r be the time required to pass one pixel from the VAX to ZMOB or vice versa via the UNIBUS. Then the time required to perform the operation on the entire image in the VAX is $C_v N^2$, while the time required to perform it on ZMOB is $2rN^2 + C_z N^2/256$. Evidently, if $512r + C_z < 256C_v$, using ZMOB is advantageous.

The situation is more complicated when we deal with local operations, in which the result for a given pixel depends on the values of the pixel and a set of its neighbors. Here, if we partition the image into disjoint parts, exchange of information between ZMOB processors is necessary, and the amount of exchange depends on the shapes of the parts. Alternatively, we can divide the image into overlapping parts, such that for every pixel there exists a processor that contains the pixel and its neighbors. This

makes data exchange unnecessary when the local operation is performed only once; but if the operation must be iterated, as is often the case, the amount of overlap needed may become excessive.

Section 2.1 discusses the optimal choice for the shapes of the parts, and concludes that square blocks are best, at least for all the standard types of neighborhoods used in local operations. Section 2.2 discusses the amount of overlap and shows that the least possible overlap is always optimal. Section 2.3 discusses the relative merits of performing an (iterated) operation on ZMOB or on the host VAX itself.

2.1 Optimal Region Shape

Iterated local operations performed on ZMOB involve cycling between two states: computation, where each processor performs calculations on data in its local memory, and communication, where some or all processors pass information between themselves in synchrony. With iterated local operations, this information will lie at the border of the image subregion contained in each processor. In the following sections, we discuss the following question: given that at each iteration processors must pass appropriate border information, what is the optimal image subregion shape?

2.1.1 Optimal Rectangle: Strips vs. Squares

The first question is whether squares are the best rectangles; intuitively, this is so, because (in the 8-neighbor case) a one-thick border around the region must be passed at each iteration, and a square has the smallest perimeter of any rectangle having the same area (Fig. 1). More rigorously, let A be the subregion area and ℓ be the rectangle length, so that A/ℓ is the rectangle height. Then C, the cost of passing the region perimeter, is proportional to

$$C(\ell) = 2A/\ell + 2\ell + 4$$

To optimize for ℓ, we differentiate and set to 0:

$$\frac{d}{d\ell}C(\ell) = \frac{-2A}{\ell^2} + 2$$

$$\frac{2A}{\ell^2} = 2$$

$$\ell^2 = A$$

Since $\ell^2 = A$ describes a square, it is the optimal rectangle.

2.1.2 The 8-Neighbor Case: Comparison of Square, Diamond, Triangular, and Circular Shaped Regions

Now that squares have been shown to be the best rectangles for local operations, diamonds, triangles, and circular regions will be compared for efficiency too (without regard for the potential difficulty of performing the subdivision). The statistics compared will be the perimeter-to-area ratio, the fraction of overhead spent passing data instead of the real work, computing. Figure 2 graphically illustrates the border size calculation, and Table 1 contains the perimeter/area ratios.

As presented, the data shows squares better than triangles better than diamonds. Circles, in the limit, are as good as squares, but for realistic values come out worse (in Fig. 2, the circle has area 49, perimeter 40, with the equivalent square's perimeter 32), not to mention the image subdivision problem. Again, squares are best.

2.1.3 Other Neighborhoods

a. The 4-Neighbor Case

Figure 3 illustrates the algebraic relationship between perimeter size and area for square, diamond and triangular regions in the 4-neighbor case. Since the neighborhood is symmetrical, the triangle produces the same result in any orientation. Table 2 shows that a square is superior to either a triangular or diamond region.

b. The 2 x 2 Neighborhood

This neighborhood shape is used in the Roberts gradient and in shrinking and shifting operations. Figure 3 and Table 2 again show that a square region subdivision is best for these operations.

c. Other Asymmetric Neighborhoods

The two types of neighborhoods used in the standard connected component labeling operation, for 8- and 4-connectedness, are shown in Figure 3. Once more, Table 2 shows that a square is best.

2.2 Optimal Region Overlap

Once the region shape has been decided, the next question is how to best coordinate the cycling between computation and communication in the course of iterated local operations. In particular, additional border information is required for each iteration of a local operation. Is it best to pass several layers of border information at once and then compute on them, or just one layer at a time?

The answer is: just one layer at a time. Intuitively, the more layers we pass at each stage, the larger each successive layer gets (Fig. 4). Likewise, the amount of computation grows for each successive layer of border points added to the region (Fig. 5). Thus, we cannot gain by passing more than one layer at a time between computations, and the pass one layer - compute one iteration strategy is optimal.

More formally, let i = the total number of iterations to be performed, j = the number of iterations to be performed at each step (the variable to be optimized), p = the time to pass one point, t = the time to perform one local operation, and n = the side length of the square we are dealing with. The communication time for one round is

$$4(nj+j^2)p$$

The computation time for one round is

$$t \sum_{k=0}^{j-1} (n+2k)^2, \text{ or (as a polynomial in j)}$$

$$t(\frac{4}{3}j^3+(2n-2)j^2+(n^2-2n+\frac{2}{3})j)$$

Adding these together, and multiplying by i/j, which is the total number of rounds to complete i iterations, gives us a total time of

$$\frac{4}{3}\,tij^2+2i(t(n-1)+2p)j + (ti(n^2-2n+\frac{2}{3}) +4inp)$$

Differentiating, setting to zero, and solving for j gives

$$j = -(\frac{3}{4}(n-1)+\frac{3}{2}\,\frac{p}{t})$$

A *negative* optimal value of j implies that we should use the minimum legal value, and j should be 1.

2.3 Timing: VAX vs. ZMOB Computation Tradeoff

When is it better to use ZMOB rather than simply using the host VAX? In other words, when does the overhead of using ZMOB (loading and unloading an image to/from the processors via the conveyor belt) offset the time saved in performing the (iterated) operation? To answer this, we must first obtain formulas for computation times on VAX and ZMOB. The variables will be:

N = length of image side (area $=N^2$)
P = number of processors in ZMOB (256)
p = time to pass one pixel between ZMOB processors
C_z = time to compute one local operation on ZMOB
C_v = time to compute one local operation on VAX
n^v = ZMOB square region side ($n^2=N^2/P$)
m = number of iterations of local operation
r = time to pass one pixel over the UNIBUS

2.3.1 Vax and ZMOB Computation Times

On the VAX, the time to compute m iterations of a local operation which takes C_v time per pixel is

$$T_{VAX} = mC_vN^2$$

On ZMOB, the computation must be split into three stages: loading (L_z), processing (P_z), and unloading (U_z).

a. Loading and Unloading

Each processor in ZMOB may be loaded simultaneously from the VAX over the conveyor belt; each processor's subregion of $N^2/256$ points is loaded at the transfer rate of the conveyor belt, p. However, the loading time is limited by the time it takes to pass the entire N^2 image points between the VAX and ZMOB over the UNIBUS; this occurs at the UNIBUS transfer rate(r). Loading and unloading times are the same:

$$L_z=U_z=rN^2$$

b. Processing

There are two stages for each iteration of ZMOB processing: communication and computation. Pass time is (4n+4)p per iteration, and compute time n^2C_z per iteration; thus

$$P_z = (4n+4)mp + n^2mC_z$$

In summary, the total time for ZMOB processing is $T_{ZMOB} = L_z+U_z+P_z$, or

$$T_{ZMOB} = 2rN^2 + (4n+4)mp + n^2mC_z$$

2.3.2 VAX vs. ZMOB Tradeoff

Given that the VAX takes some fraction α of the time that ZMOB does for the given local operation (α will vary), how time-consuming must that local operation be (on ZMOB, say) before it is worth moving the image to ZMOB for processing? Let $C_v=\alpha C_z$, and solve (letting $n^2 = N^2/P$):

$$T_{VAX} = T_{ZMOB}$$

$$\alpha mC_zN^2 = 2rN^2+(4n+4)mp+n^2mC_z$$

$$C_z = \frac{2rN^2 + 4(N/\sqrt{P}+1)mp}{mN^2(\alpha-1/P)}$$

Tables 3 and 4 show typical results for the realistic values

$N = 512$
$P = 256$
$p = 10^{-5}$ sec. (10μsec/byte ZMOB transfer rate; a conservative estimate)
$r = 4 \times 10^{-7}$ sec (400 nsec/byte UNIBUS transfer rate)

Table 3 gives minimum ZMOB computation times for $T_{VAX} = T_{ZMOB}$, and Table 4 gives minimum times for $T_{VAX} = 10T_{ZMOB}$.

We can see from these tables that, since the smallest value for C_z is the time required for one Z80 instruction or about one microsecond (10^{-6} sec.), ZMOB will almost always be advantageous and will often be more than ten times faster than the VAX. We can also see this in the following list of fractional overhead values (the ratio of ZMOB loading and unloading time to the total processing time) for a (one-iteration) local operation: 98.9% when C_z equals 10^{-6} sec. (around one instruction), 94.8% at $C_z=10^{-5}$ sec., 66.9% at $C_z=10^{-4}$ sec., 17.0% at $C_z=10^{-3}$ sec., and 2.0% at $C_z=10^{-2}$ sec.; the ratio drops well below 1% for more than one iteration or larger C_z values. Thus, even for once-performed local operations, ZMOB loading and unloading overhead is relatively small, and since $C_z/256 \ll C_v$ (usually), the use of ZMOB will ordinarily be advantageous.

3. Two-dimensional Discrete Transforms

The method described below calculates the two-dimensional Fourier transform (or other similar discrete transforms) of an N by N image in O(N log N) time. Each processor is assigned a subregion of consecutive rows of the image. The process is composed of three steps: a row-wise fast Fourier transform (FFT) by each processor; transposition of the image (matrix) between processors; and a (now) column-wise FFT. Executing the FFT on each row held by the processor is straightfoward and performed in O(N log N) time. Transposition of the image to perform the column-wise transform is accomplished as follows: each processor is destined to receive a portion of each row during the course of the transposition, with one portion remaining in the processor. Processor i passes the portion to go to processor i+1, which can be determined by computation, during the first communication round; this quantity may be several elements (and several bytes per element). During the second round, processor i+2 receives its portion from processor i, and so on, until 255 rounds have been completed. Each processor now contains one or more columns. The process is illustrated in Figure 6. Each row and column takes O(N log N) time to be transformed; each processor contains N/256 rows or columns, but since

N is bounded by the image size that ZMOB can realistically hold, this can be regarded as a constant. The transposition process takes O(N) time to transfer the elements of one or more rows to other processors. Thus the entire algorithm takes time proportional to 2N log N + N time, or O(N log N).

4. Geometric Correction

The problem of performing geometric correction of an image in parallel using ZMOB involves each processor receiving information about the input image from other processors for each point in the output subregion assigned to that processor. The value of each output point is computed by interpolation from the values of a set of input points surrounding an ideal input point, usually having non-integer coordinates, whose position is defined by the inverse of the given coordinate transformation. For example, for bilinear interpolation we use a 2 by 2 neighborhood of the ideal input point, while for cubic spline interpolation we use a 4 by 4 neighborhood.

One desirable condition for efficient geometric correction on ZMOB is to have each interpolation neighborhood reside entirely within one processor, so that no more than one need be consulted to obtain an output image value. This may be insured by providing suitable overlaps between the subregions handled by the processors (e.g., a one-row border for a 2 by 2 neighborhood). However, there is no way of guaranteeing, in general, that we can compute the output values in such a way that, at each step, each processor needs information from a different processor. As a result, the communication between processors will not be evenly distributed, and it becomes impossible to give an

exact estimate of the time required. Only in the special case where the pixel displacement is bounded by some distance d, it becomes possible to provide an overlap between processors proportional to d, thus allowing each processor to compute its portion of the output image without consulting other processors.

5. Computation of Image Statistics

5.1 Image Histogram Algorithm

We first consider the problem of creating the grey-level histogram of an image in ZMOB (either freshly loaded from the VAX or already present after a series of previous image operations). In the algorithm to be described below, the goal is for each of the 256 processors to contain the frequency of occurrence of one of the values of the (eight-bit) grey level, for an image of arbitrary size (though with an upper bound, within the constraints of local memory).

The method is divided into two steps: local histogram creation and histogram merging. During histogram creation, each processor creates a 256-bucket histogram for its sub-portion of the total image, the image area being divided into 256 equal parts (the strategy for partitioning is irrelevant and no overlapping is necessary). Each bucket may be of some appropriate size, say 16 or 24 bits, which will accommodate the largest possible value, or perhaps the highest bit may be reserved as a bucket overflow indicator. Each processor also has a different (and larger) bucket, corresponding to its processor I.D. and to the grey level that it will be counting, that it is responsible for totalling during the next step.

During the histogram merging phase, each processor will pass the contents of each histogram bucket (other than its own) to the appropriate processor for totalling during 255 communications rounds. Each processor already has the initial count for its own bucket. During the first round, processor i passes the contents of bucket i+1 (module 256) to processor i+1 for totalling. On the second round, bucket i+2 is passed to processor i+2, and so on. After all 255 bucket values belonging to other processors are passed, they are disregarded and the processor's own final value is returned to the VAX.

5.2 Co-occurrence Matrix Computation

The problem of computing co-occurrence matrices is very similar to that of histogramming. Each co-occurrence matrix element is a frequency of a pair of grey levels occurring at a particular distance and orientation from one another, just as each element of the histogram (vector) is the frequency of a single grey level occuring at any pixel. The one difference is that a co-occurrence matrix is potentially much larger (the square of the total number of grey levels). Usually, the range of grey levels used is more restricted than in the histogram case -- e.g., we use only the upper five or six bits of the grey value. Another difference is that the geometry of the pixel pair calls for the use of appropriate overlap when storing the image subregions in the processors (see Figure 7). In particular, if each pixel is compared with one m units horizontally and n units vertically displaced, we can use square subregions with m columns and n rows of overlapping. This obviates the need to request information from other processors during the course of the computation, at a great savings of time with a small cost of extra memory used. The process then proceeds similarly to the histogramming algorithm: each processor computes a co-occurrence matrix for its subregion; each processor is assigned 1/256th of the matrix elements (arbitrarily)

to total; and through 255 rounds of communication, each processor sends each other processor its portion of the matrix to total, and receives the other 255 values for its own matrix portion. Each matrix portion will probably consist of several matrix elements, each potentially several bytes long. After the totalling is completed, each processor communicates its portion to the VAX where the final matrix is assembled.

6. Concluding Remarks

We have seen that ZMOB should have substantial speed advantages in many image processing situations. In particular, we have outlined efficient ZMOB communication/computation schemes for point and local operations (with particular reference to how the data should be partitioned among the processors), discrete transforms, geometric operations (in some cases), and computation of statistics. These schemes demonstrate that efficient use of ZMOB's parallelism is possible for essentially all basic image processing and analysis tasks.

Reference

1. Chuck Rieger, John Bane, and Randy Trigg, A highly parallel multiprocessor, TR-911, July 1980, Department of Computer Science , University of Maryland, College Park, Maryland.

Region Shape	Perimeter/Area
square	$\frac{4m + 4}{m^2}$
triangle	$\frac{8m + 8}{m^2 + m}$
diamond	$\frac{3m + 3}{m^2 + 1}$
circle $\left(\lim_{m \to \infty}\right)$	$\frac{4m + 4}{m^2}$

TABLE 1

	Square	Triangle	Diamond
4-neighbor	$\frac{4}{m}$	$\frac{2(3m + 1)}{m^2 + m}$	$\frac{4(m + 1)}{m^2 + 1}$
2 x 2	$\frac{2m + 1}{m^2}$	$\frac{2(2m + 1)}{m^2 + m}$	$\frac{2(2m + 1)}{m^2 + 1}$
NW, N, NE, and W neighbors	$\frac{3m + 1}{m^2}$	$\frac{2(3m + 1)}{m^2 + m}$	$\frac{5m + 3}{m^2 + 1}$
N and W neighbors	$\frac{2}{m}$	$\frac{4}{m + 1}$	$\frac{3m + 1}{m^2 + 1}$

TABLE 2

m \	1	1/2	1/4	1/8	1/16	1/32	1/64
1	8.08×10^{-7}	1.62×10^{-6}	3.27×10^{-6}	6.65×10^{-6}	1.37×10^{-5}	2.94×10^{-5}	6.87×10^{-5}
2	4.07×10^{-7}	8.16×10^{-7}	1.65×10^{-6}	3.34×10^{-6}	6.91×10^{-6}	1.48×10^{-5}	3.46×10^{-5}
4	2.06×10^{-7}	4.13×10^{-7}	8.33×10^{-7}	1.69×10^{-6}	3.50×10^{-6}	7.50×10^{-6}	1.75×10^{-5}
8	1.05×10^{-7}	2.12×10^{-7}	4.27×10^{-7}	8.67×10^{-7}	1.79×10^{-6}	3.84×10^{-6}	8.96×10^{-6}
16	5.53×10^{-8}	1.11×10^{-7}	2.24×10^{-7}	4.54×10^{-7}	9.39×10^{-7}	2.01×10^{-6}	4.70×10^{-6}
32	3.02×10^{-8}	6.05×10^{-8}	1.22×10^{-7}	2.48×10^{-7}	5.13×10^{-7}	1.10×10^{-6}	2.56×10^{-6}
64	1.76×10^{-8}	3.53×10^{-8}	7.13×10^{-8}	1.45×10^{-7}	2.99×10^{-7}	6.41×10^{-7}	1.50×10^{-6}

Minimum threshold for local

operation computation time C_z (in seconds)

for ZMOB usefulness

TABLE 3

m \	1	1/2	1/4	1/8	1/16
1	8.38×10^{-6}	1.75×10^{-5}	3.82×10^{-5}	9.37×10^{-5}	3.43×10^{-4}
2	4.22×10^{-6}	8.79×10^{-6}	1.92×10^{-5}	4.71×10^{-5}	1.73×10^{-4}
4	2.13×10^{-6}	4.45×10^{-6}	9.72×10^{-6}	2.39×10^{-5}	8.75×10^{-5}
8	1.09×10^{-6}	2.28×10^{-6}	4.98×10^{-6}	1.22×10^{-5}	4.48×10^{-5}
16	5.73×10^{-7}	1.19×10^{-6}	2.61×10^{-6}	6.40×10^{-6}	2.35×10^{-5}
32	3.13×10^{-7}	6.52×10^{-7}	1.42×10^{-6}	3.50×10^{-6}	1.28×10^{-5}
64	1.82×10^{-7}	3.80×10^{-7}	8.31×10^{-7}	2.04×10^{-6}	7.48×10^{-6}

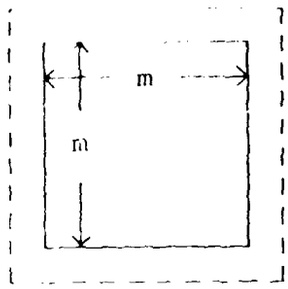
Minimum threshold for local

operation computation time C_z (in seconds)

for 10-fold speedup using ZMOB.

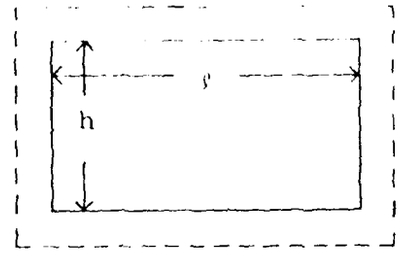
(Note: beyond a 4:100 ratio of C_v:C_z [$v < 10/P$], ZMOB cannot approach a 10-fold speedup)

TABLE 4



<u>square</u>

perimeter = 4m+4

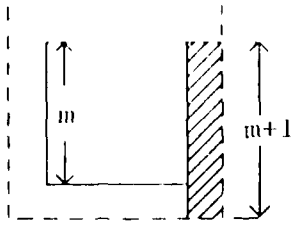


<u>rectangle</u>

perimeter = 2h+2ℓ+4

Area = m^2=hℓ

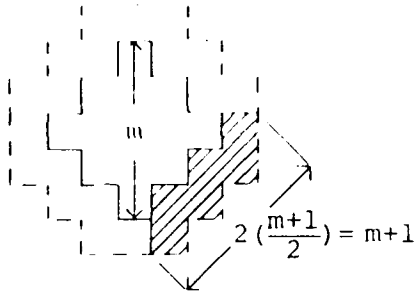
Figure 1.



square

area = m^2

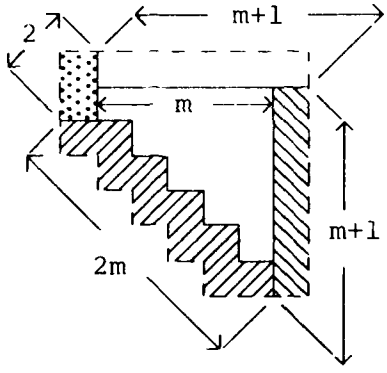
perimeter = $4(m+1) = 4m+4$



diamond

area = $\frac{m^2+1}{2}$

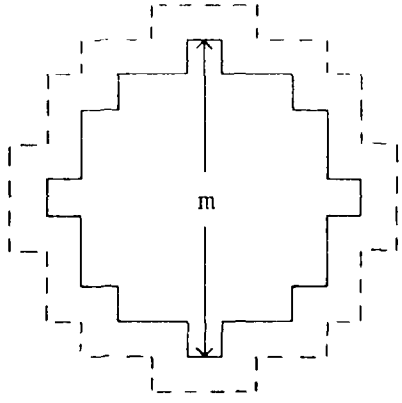
perimeter = $4(m+1) = 4m+4$



triangle

area = $\frac{m(m+1)}{2}$

perimeter = $2(m+1)+2m+2$
$= 4m+4$



circle

area = $\lim_{m\to\infty} \pi(\frac{m}{2})^2$

perimeter = $\lim_{m\to\infty} \pi(m+1)$

Figure 2.

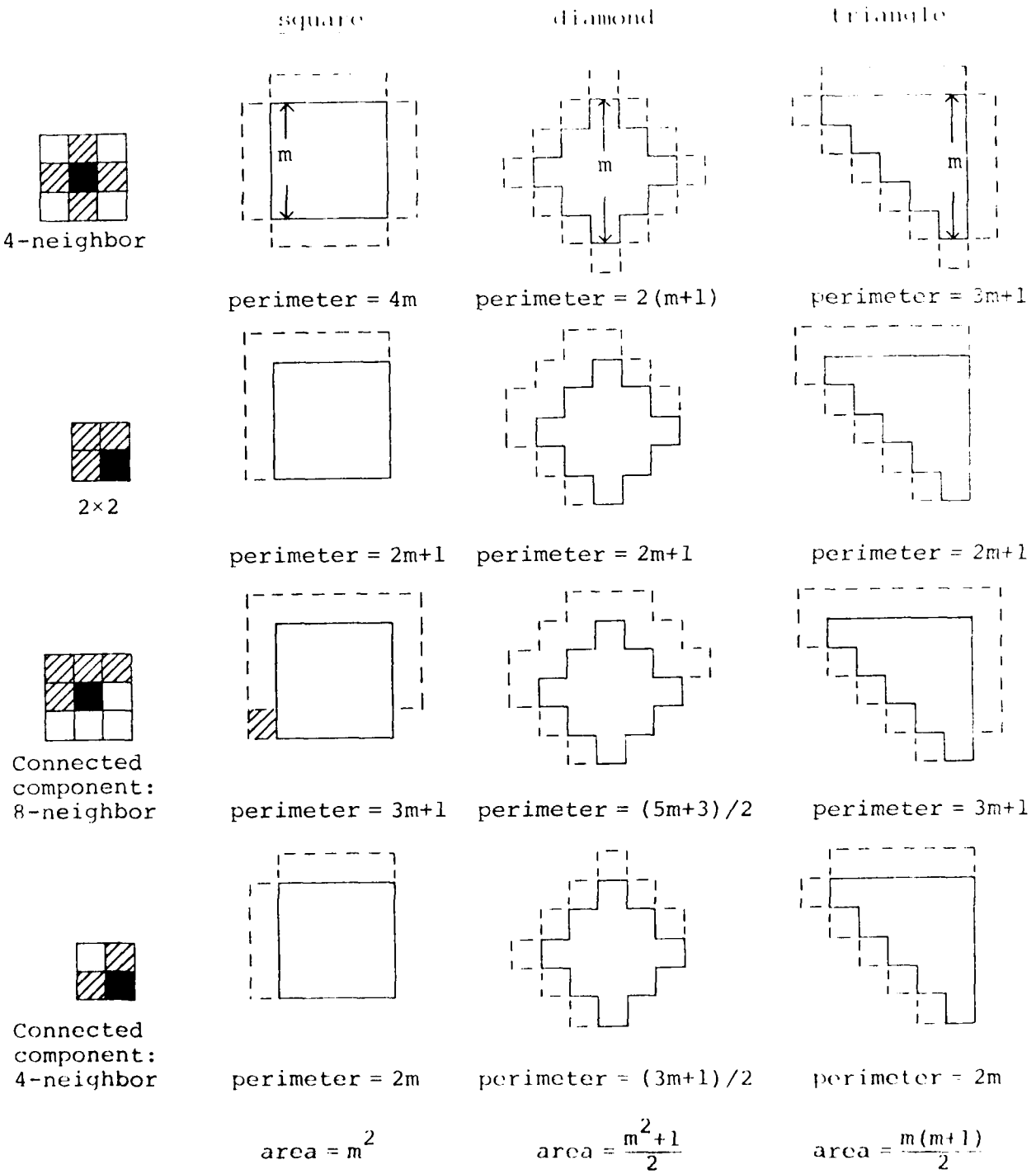
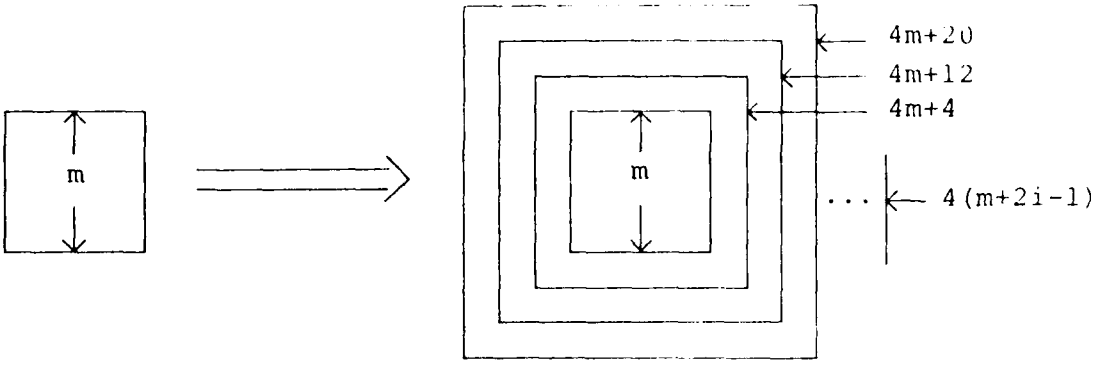
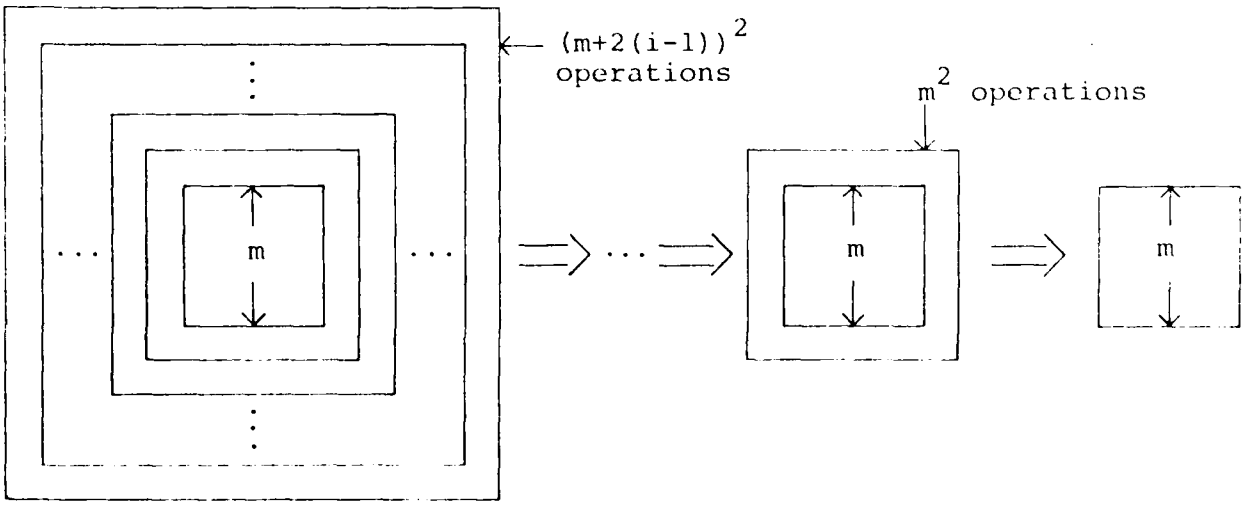

square
diamond
triangle
4-neighbor
m
m
m
perimeter = 4m
perimeter = 2(m+1)
perimeter = 3m+1
2×2
perimeter = 2m+1
perimeter = 2m+1
perimeter = 2m+1
Connected component: 8-neighbor
perimeter = 3m+1
perimeter = (5m+3)/2
perimeter = 3m+1
Connected component: 4-neighbor
perimeter = 2m
perimeter = (3m+1)/2
perimeter = 2m
area = m^2
area = $\frac{m^2+1}{2}$
area = $\frac{m(m+1)}{2}$

Figure 3.



Communication

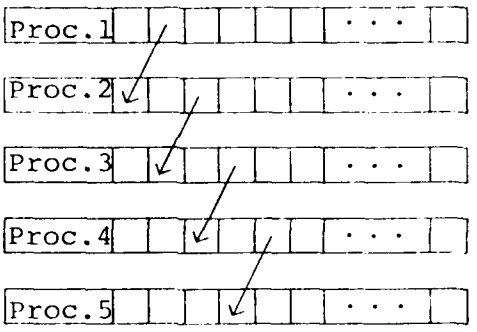
Figure 4.



Computation

Figure 5.

Round 1



Round 2

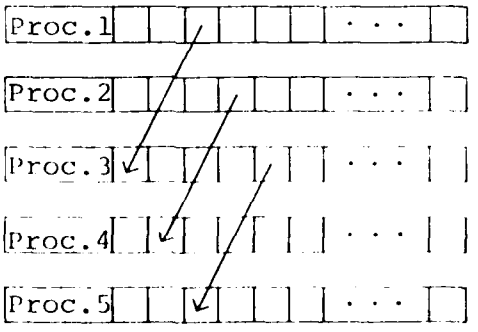


Figure 6.

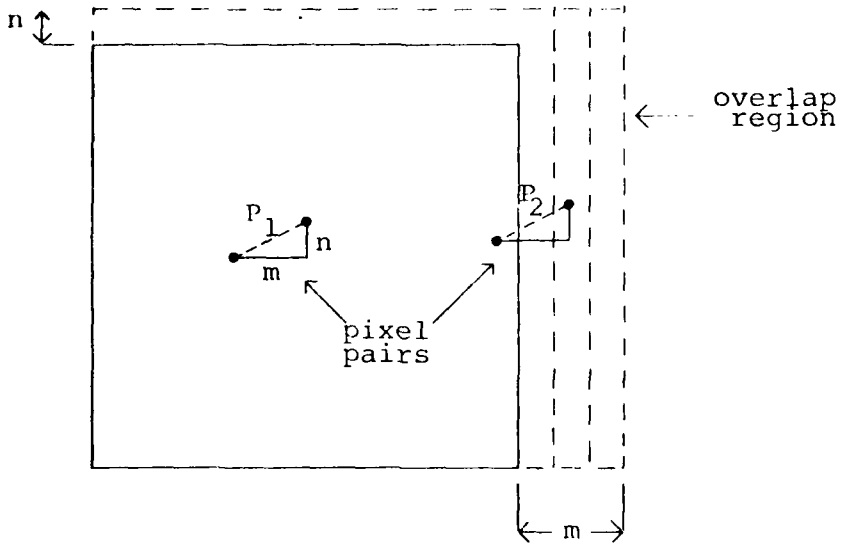


Figure 7.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR- 81-0143	2. GOVT ACCESSION NO.: AD-A096 420
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): IMAGE PROCESSING ON ZMOB	5. TYPE OF REPORT & PERIOD COVERED: INTERIM
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Todd Kushner, Angela Y. Wu, Azriel Rosenfeld	8. CONTRACT OR GRANT NUMBER(s): AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, MD 20742	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Math. & Info. Sciences, AFOSR/NM, Bolling AFB, Washington., DC 20332	12. REPORT DATE: December 1980
	13. NUMBER OF PAGES: 32
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Parallel processing
ZMOB

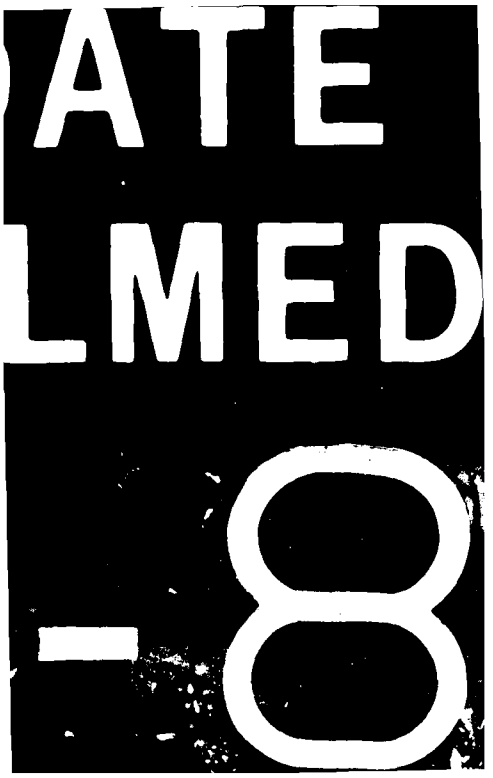
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ZMOB is a multi-microprocessor system consisting of 256 Z80A microprocessors that communicate via a fast cyclic shift-register bus. This paper discusses the efficient use of ZMOB for various types of image processing operations, including point and local operations, discrete transforms, geometric operations, and computation of image statistics.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


ATE
LMED
-8