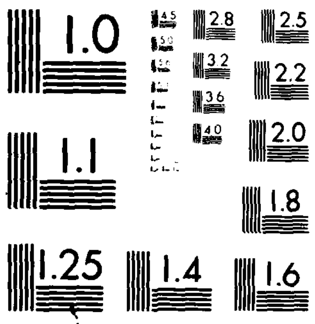
AD-A098 121   MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB   F/G 9/2
IMAGE PROCESSING ON MPP.(U)
FEB 81 T KUSHNER, A Y WU, A ROSENFELD   AFOSR-77-3271
UNCLASSIFIED TR-1007   AFOSR-TR-81-0367   NL

END
DATE
FILMED
5-81
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

TR-1007 February 1981
AFOSR-77-3271

IMAGE PROCESSING ON MPP: 1

Todd Kushner
Angela Y. Wu *
Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

The Massively Parallel Processor (MPP) is a 128 by 128 array of processing elements that communicate with their horizontal and vertical neighbors by shifting data one bit at a time. This paper describes the efficient use of MPP for various types of image processing operations, including point and local operations, discrete transforms, and computation of image statistics. A comparison between MPP and ZMOB (a system consisting of 256 microprocessors) is also presented.

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

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 report.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) AFOSR TR-81-0367	2. GOVT ACCESSION NO. AD-A098 121 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) IMAGE PROCESSING ON MPP	5. TYPE OF REPORT & PERIOD COVERED (9) Interim Rept.
	6. PERFORMING ORG. REPORT NUMBER (14) TR-1007
7. AUTHOR(s) (10) 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 (16) 2304/A2 61102F
11. CONTROLLING OFFICE NAME AND ADDRESS Math. & Info. Sciences, AFOSR/NM, Bolling AFB, Washington, DC 20332 (17) A2	12. REPORT DATE (11) February 1981
	13. NUMBER OF PAGES 32
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 35	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
Pattern recognition
Parallel processing
Cellular computers
MPP

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

The Massively Parallel Processor is a 128 by 128 array of processing elements that communicate with their horizontal and vertical neighbors by shifting data one bit at a time. This paper discusses the efficient use of MPP for various types of image processing operations, including point and local operations, discrete transforms, and computation of image statistics. A comparison between MPP and ZMOB (a system consisting of 256 microprocessors) is also presented.

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

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

411044

1. Introduction

1.1 MPP

The Massively Parallel Processor (MPP) is a 128 by 128 array of processing elements (PEs) that communicate with their horizontal and vertical neighbors by shifting data one bit at a time. For a description of the MPP design see [1]. In the following paragraphs we outline only a few basic features of MPP that are needed in designing image processing algorithms for it.

Each image processing algorithm implemented on MPP will consist of two phases: computation and communication. To support the computational aspect of parallel algorithms, each PE, while being a "bit-slice" processor, is capable of supporting a complete conventional instruction set. Each PE has a bit addressable local memory of 1024 bits and a number of fast registers to support arithmetic and interprocessor communication.

Parallel algorithms generally require interprocessor communication: to accomplish this, every PE can synchronously shift data to its north, south, east, or west neighbor. (At the array edges, processor passing may "wrap around" to the PEs at the other end of the row or column.) When loading data from the host machine, a 128-long bit vector may be passed to the 128 edge processors all at once, which may in turn shift it across the image while the rest of the image is loaded. In the current configuration this data loading occurs over a UNIBUS from a VAX host.

1.2 Image Processing on MPP

This paper deals with the efficient useof MPP for performing various types of image processing operations, including point and local operations, discrete transforms, and computation of image statistics. The aim is to make the fullest possible use of MPP's parallelism, so as to achieve a speedup by a factor proportional to the number of PEs (128^2 = 16,384). We also compare MPP processing with performing operations on the host VAX itself, as well as with processing on ZMOB (a system consisting of 256 microprocessors that communicate via a fast shift-register bus). A more detailed treatment of image processing on ZMOB can be found in [2].

2. Point Operations

A point operation on an image maps the value of each pixel into a new value, independent of the values of other pixels. The image is divided equally among the PEs; 1 pixel/ processor for a 128 by 128 image, 4 pixels/processor for a 256 by 256 image, 16 pixels/processor for a 512 by 512 image, and so on. Images much larger than 512 by 512 cannot be held in the 1024 bits of local memory available to each PE. The PEs are loaded with the image data from the host VAX over the UNIBUS, the point operation is performed, and the results are returned to the host VAX.

To compute the amounts of time needed to perform point operations on MPP and on the VAX, let C_m and C_v be the times for an MPP PE and for the VAX, respectively, to perform the given operation on one pixel. In an N by N image, there are N^2 pixels: thus, C_vN^2 and $C_mN^2/16,384$ are the times to perform the point operation on the VAX and MPP (with its 16,384 processors), respectively.

However, in the case of the MPP, there is also the amount of image loading and unloading time to consider. On the MPP, data is loaded from the host VAX, via the UNIBUS, to a staging area of the MPP, where the data is input simultaneously to 128 edge PEs, 128 bits at a time. Letting r be the rate at which a byte of data is transferred on the UNIBUS (400nsec., and p be the rate at which a bit of data is passed between PEs, let us compute how long it takes to load a 128 by 128 (say) image of byte-long pixels: 1) from the VAX to MPP staging area via the UNIBUS, and 2) from the MPP staging area to the PEs (a concurrent process). Via the UNIBUS it takes $128 \times 128 \times r$, or 6.534msec. From the staging area to the PEs, it takes $128 \times 128 \times 8$ bits $\times 1/128$ (number of bits passed simultaneously) $\times p$, or 1.024μsec. Thus, the UNIBUS is the rate-limiting step of the MPP image loading process, and the total time to load and unload is $rN^2 + rN^2 = 2rN^2$.

In summary, on the VAX, the time to perform the operation on the entire image is C_vN^2, while the time to perform it on the MPP is $2rN^2 + C_mN^2/16{,}384$. If $32{,}768r + C_m < 16{,}384C_v$, using the MPP is faster than using the VAX.

With local operations, the situation is more complicated because information must be shared between neighboring processors. The next section will discuss the amount of time it takes to perform local operations, using different neighborhood geometries. A comparison with performing an (iterated) operation on the host VAX will also be given. Due to the limited local memory of MPP PEs, the focus of the discussion will be the one pixel per PE case.

3. Local operations

Each iteration of a local operation consists of two steps: a neighbor-passing step, and a computation step involving the gathered neighborhood. Several types of local neighborhoods are commonly used, and these (with the steps involved in passing neighbors) are outlined in Figure 1. Every passing sequence involves the exact number of neighbors required, except for the 8-neighbor connected component case, where one extra neighbor transfer occurs (due to the interconnection structure of MPP). In all, eight pixels are passed in the 8-neighbor case; four pixels in the 4-neighbor case; three pixels in the 2x2 case; five pixels in the 8-neighbor connected component case; and two pixels in the 4-neighbor connected component case. In the following paragraphs we analyze a specific case, the 8-neighbor local operation, and give a comparison between the performance of MPP and of the host VAX itself.

When is using MPP better than simply using the host VAX? In other words, when does the overhead of using MPP (loading and unloading an image via the UNIBUS) offset the time saved in performing an (iterated) local operation? To answer this, we must first obtain formulas for computation times on VAX and MPP.

We will assume a 128 x 128 image, thus one pixel per MPP PE. The relevant parameters are:

N = length of image side = 128

p = time to pass one bit between MPP PEs

m = number of bits per pixel (8, for 256 grey levels)

C_m = time to compute one local operation on MPP

C_v = time to compute one local operation on VAX

n = number of iterations of the local operation

r = time to pass one pixel over the UNIBUS

On the VAX, the time to compute n iterations of a local operation taking C_v time per pixel is

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

On MPP, the computation must be split into three states: Loading (L_m), processing (P_m), and unloading (U_m). As we have already seen, the loading of the MPP PEs is limited by the amount of time it takes to transfer the image pixels over the UNIBUS (loading of the PE. from that point is much faster). Loading and unloading times are the same:

$$L_m = U_m = rN^2$$

There are two stages for each iteration of a local operation on MPP: communication and computation. For an eight-neighbor operation with one pixel/PE, the pass time is 8mp per iteration, and the compute time is C_m per iteration. Thus,

$$P_m = 8nmp + nC_m$$

In summary, the total time for MPP processing is $T_{MPP} = L_m + U_m + P_m$, or

$$T_{MPP} = 2rN^2 + 8nmp + nC_m$$

Given that the VAX takes some fraction α of the time that an MPP PE does for the given local operation (α will vary), how time-consuming must that local operation be (on MPP, say) before it is worth moving to MPP for processing? Let $C_v = \alpha C_m$, and solve:

$$\begin{aligned} T_{VAX} &= T_{MPP} \\ \alpha n C_m N^2 &= 2rN^2 + 8nmp + nC_m \\ C_m &= \frac{2rN^2 + 8nmp}{\alpha n N^2 - n} \end{aligned}$$

Tables 1 and 2 show typical results for the realistic values

$N = 128$

$m = 8$

$p = 3\text{x}10^{-7}$sec. (300nsec/bit PE transfer rate)

$r = 4\text{x}10^{-7}$sec. (400nsec/byte UNIBUS transfer rate)

Table 1 gives minimum MPP computation times for $T_{VAX} = T_{MPP}$; Table 2 gives minimum times for $T_{VAX} = 10T_{MPP}$.

We can see from these tables that MPP will usually be advantageous over, and often more than ten times faster than, the VAX, since one to ten microseconds is the minimum for MPP PE operations. For short once-iterated operations, MPP will be IO-bound: for C_m between 10^{-7} and 10^{-3} sec., the fractional overhead in transferring the image between the VAX and MPP is over 90%; at $C_m = 10^{-2}$ sec., the overhead is 57%; at $C_m = 10^{-1}$ sec., the overhead is 12%; and, at higher C_m values or for more than one iteration, the overhead drops well below 1%. Generally, more than one iteration of a local operation must be performed before MPP is useful.

In the case where we have several pixels per PE (N by N image, $N > 128$), the situation is different. For local operations on images larger than 128 by 128, the general formula for the computation time is $C_m mN^2/P$ and for the communication time is $(4(N/\sqrt{P}) + 4)$ (the number of points bordering the size N^2/P subregion) times mp. Thus, with increasing N (within the constraint of the limited PE local memory), the computation time rises by the square and the communication time rises linearly with N; consequently, the calculation becomes more CPU bound. In any case, the small amount of memory per PE limits the number of pixels that can be handled by a PE. The values of a pixel and its eight neighbors already take up a significant fraction of this memory (72 bits, or about 7%). To handle a 2x2 block of pixels and their neighbors (a 4x4 block in all) requires nearly twice this, and a 3x3 block with neighbors (5x5 in all) requires 40% of the memory. It would be difficult to handle much larger blocks.

4. Computation of image statistics

In this section we consider some MPP tasks involving computation of image statistics - in particular, the computation of image histograms and co-occurrence matrices on MPP.

4.1 Histograms

The histogram algorithm for MPP consists of two main steps: histogramming the columns of the image (creating a histogram for the pixels in each column with the "buckets" for each gray-level residing along with the pixels in the PEs of each row), and totalling the row so that the (e.g.) left-most column of PEs contains the final histogram for the image. For simplicity, the method described below is designed for one pixel and one histogram bucket per PE--a 128 x 128 image, and 128 (i.e., seven-bit) gray levels.

a) Histogramming columns

The method for histogramming the columns of the image involves passing the gray-levels cyclically (and synchronously) around the PEs of that column, using the "wraparound" feature of the MPP when passing pixels between processors. The goal is to have the processor in row $\underline{i}$ of the given column contain a count of the number of occurrences of gray level $\underline{i}$ in that column. In this example, each PE sets aside an eight-bit counter for the histogram "bucket" and cycles the seven-bit gray-levels through each of the 128 PEs in the column. Whenever a gray-level corresponding to the row number of the PE passes through, the counter in that PE is incremented by 1. This method is extensible to more than 128 gray levels; the processors simply multiply their responsibility for gray levels (e.g., two each for 256 or four each for 512 gray levels); this is similarly true for larger images. Letting N = the number of processors in the column (128) and m = the number of gray levels (128, in this example), the complexity of this part of the algorithm is $\theta(n\log m)$. See Figure 2 for an example of an eight-long column (and eight gray levels).

b) Totalling rows

Totalling the rows to obtain the final histogram is done in a somewhat more complicated fashion. The method is to pass the counters derived from the column histogramming step leftward and sum them at each level. This summing may be done bit-by-bit (by adding two bits and saving the carry for the next round), since they must be passed bitwise anyway, to save time. The least significant bit (LSB) is passed leftward first, and this is added to the LSB of the held counter (with the carry saved in a special register); the LSB of the resulting number is passed at the next step. This continues until the final LSB propagates to the leftmost column, where it is added to that column's counter and results in the LSB of the final bucket count. Meanwhile, the next-to-last bit propagates leftward after the LSB, being added to the next-to-last bit, and the carry from the LSB addition, in the same fashion, until it propagates to the left column.

Since larger and larger counts are being formed as the column totals merge, the counters of each column must be extended to accommodate these sums. For column N (numbering from 1 at the right to 128 at the left), that column's counter must be extended to $(\lfloor \log_2 N \rfloor + 8)$ bits. So that the algorithm may work in proper synchrony, every bit of each counter must be passed upward, even leading zeros. Figure 3 presents a worked out example for a row of length 6.

Letting N = the number of processors in a row of the processor array (128 on MPP), it takes N steps to propagate the LSB to the left column. It then takes $(2\log_2 N - 1)$ steps to pass the rest of the $(2\log_2 N)$-bit counter maintained by the PE in the second-to-left column. Thus, this part of the algorithm takes $\theta(N + \log_2 N)$ steps.

The total complexity of histogramming on the MPP is $\theta(n\log_2 m)$ (m the number of gray levels) from the first part plus $\theta(N + \log_2 N)$ in the second part, which totals to $\theta(N\log m)$.

c) Time requirements

The first step, column histogramming, involves cycling N m-bit pixels through the column PEs, comparing the pixel value to the row number and (potentially) incrementing a counter at each step (note that on an SIMD machine such as MMP, a step such as this incrementing takes just as much time whether it occurs or not, since the instruction(s) must be sent to each processor anyway; they are simply disarmed if necessary). Thus, at each of N steps, an m-bit pass, an m-bit compare, and an (n +1)-bit add occur; thus, the time taken for column histogramming is:

$$T_{col} = N(mp + mc + (n + 1)a)$$

Here

N = length of image side = 128

$n = \log_2 N = 7$

m = number of bits per gray level = 7 (128 gray levels)

p = time to pass one bit between MMP PEs (300nsec.)

a = time, per bit, to add two numbers on MPP (300nsec.)

c = time, per bit, to compare two numbers on MPP (400nsec.)

r = time to pass one pixel over the UNIBUS (400nsec.)

The MPP instruction timing will vary, depending on the exact programming of the algorithm.

For the second step, row totalling, there are (N + 2n - 1) steps where one bit is passed and one addition takes place; thus, the time taken for row totalling is:

$$T_{row} = (N + 2n - 1)(p + a)$$

To this is added the time to load and unload the image, which is:

$$T_{load} = T_{unload} = rN^2$$

The total time for histogramming a 128 by 128 image (128 gray levels) on MPP is thus

$$T_{MPP} = N(mp + mc + (n + 1)a) + (N + 2n - 1)(p + a) + 2rN^2$$
$$\simeq 0.001019 \text{ (compute)} + 0.0098304 \text{ (load and unload)}$$
$$\simeq 0.0108494 \text{ sec.}$$

On the VAX, histogramming requires the time it takes to update one histogram bin (say t_v) times the number of pixels in the image, N^2. Thus the time to histogram an image on VAX is

$$T_{VAX} = N^2 t_v$$

For the 300nsec. cycle time of the VAX, t_v will typically be 1 to 10 microseconds, depending on how the program is coded (assembly versus C). Thus, on the VAX, histogramming a 128 by 128 image will take about 0.0016384 to 0.016384 seconds. This is 15% to 1.5 times the total MMP time, or 1.6 to 16 times the MPP computation time alone. Thus, MPP seems to offer only a marginal, if any, improvement over using the VAX for this task.

4.2 Co-occurrence matrices

A co-occurrence matrix is essentially a "histogram" of the occurrences of pairs of gray levels; if there are M different gray levels, it is an M by M matrix. To compute the co-occurrence matrix of an image, the neighbor of each point at some displacement δ is obtained, and the appropriate entry (gray-level$_1$, gray-level$_2$) of the matrix incremented by one. On the MPP, this would be analogous to the histogram algorithm presented earlier: the M by M matrix would be treated as a size M^2 histogram; each processor would be responsible for M/128 rows of the matrix; the points are circulated around the columns, each

PE updating appropriate entries of its rows; finally, these columns are passed leftward and totalled.

However, since there are only 1024 bits (128 bytes) available in the local memory of each PE, the largest number of values which can be accommodated is 128 (with no room to space) or, practically, 64. Thus co-occurrence matrix computation on MPP should be done for matrices of small size, e.g., 8 by 8.

5. Two-dimensional discrete transforms

On MPP, the following method calculates the two-dimensional Fourier transform (or other similar discrete transform) of an N by N image in $\theta(N)$ time. The process is composed of two steps: the discrete transform of the image row-wise, then the discrete transform column-wise. To transform the rows, each processor computes the first complex term it will use in its summation, multiplies it by the pixel value, and stores the result in a register. Then each pixel is shifted circularly, the second term is calculated, multiplied, added to the counter, and so on. This process is repeated similarly for the columns. Each takes N steps, thus the algorithm takes $\theta(N)$ time. However, while this method does well on 128 by 128 images (one pixel per PE), the processors quickly run out of local memory with larger images.

MPP is also very limited in its ability to perform geometric operations on images, primarily due to memory constraints. Due to the fixed geometry of the processors and the synchronous nature of their intercommunication, unless each processor can hold the block of data it needs to calculate the values of the output pixels, there is no "smooth" way of getting the needed data to its destination in a parallel fashion.

6. Comparison of MPP and ZMOB

Tables 3 and 4 show the performance of MPP and ZMOB, respectively, at various basic image processing tasks. The MPP table uses bits as the basic image units, whereas the ZMOB table uses pixels. These tables include total complexity measures for computation time, communication time, and memory requirements as a function of image size (N, the diameter), number of processors (P), the number of gray levels (M), and various constants. Tables 5 and 6 restate this information for the histogramming algorithm, based on the relations of P and M to N. Note that a factor of $\theta(N^2)$, due to the UNIBUS image loading and unloading step, appears in each communication complexity formula, separated by parentheses from the inter-processor communication complexity.

If the number of processors in ZMOB is regarded as proportional to the image diameter (N), and the number of processors in MPP as proportional to image size (N^2), then we see in Tables 3 and 4 how computational complexity decreases, but intercommunication complexity increases, when the relative number of processors assigned to a task increases. A comparison of the actual timings of a histogram algorithm, in Tables 1 and 2, and Tables 3 and 4 in [2], show that in reality, the machines are quite close in their utility relative to the VAX.

7. Concluding remarks

Due to the inflexible intercommunication structure in MPP, certain algorithms are constrained to have a value or values propagate from one end of the array to the other, and thus have an unavoidable factor of N, or 8N for one-byte data, built into their complexity. In addition, other algorithms, where communication does not occur in a tightly orchestrated way, become intractable. The severely limited local memory space is also a difficulty in considering certain algorithms or certain (practical) image sizes. Nevertheless, MPP still manifests significant speed advantages, particularly when it is used for point and local space-domain operations or for transform-domain filtering. It will be a powerful tool for image processing and analysis.

Appendix

Image reconstruction on MPP and ZMOB

The two methods of image reconstruction which will be discussed for implementation on MPP and ZMOB are the Filtered Back Projection and Fourier reconstruction methods. The former basically involves taking each point of a density projection and "smearing" its value, divided by an appropriate measure of width, across the image. This is repeated for each projection, its points being smeared additively, with suitable (pre-and) post-processing of the image to compensate for the spread function of the back projection process. The latter method involves taking the Fourier transform of each projection and, by applying the Fourier Slice Theorem (which states that the transforms of the projections are the values of the central cross sections, at the same orientations, of the transformed image), using them as values from which to interpolate the Cartesian-grid representation of the transformed image, from which the reconstructed image is derived by inverse transformation.

On the MPP, the first method, filtered back-projection, is difficult due to the non-linear nature of the reconstruction process. The problem may be restated thus: for any point in the image, what points from each projection must be used to get (interpolate) that projection's contribution to the final value? Since the projections are at various orientations, this becomes a geometric operation problem which, except for the two-projection situation, is of a form that the fixed geometry of the MPP cannot easily handle.

In the Fourier reconstruction method, while rows of processors may be able to transform the projections, and the projections, once in place among the appropriate processors, may be fairly readily interpolated (and the image inverse transformed by the method in Section 5), it is not clear how to smoothly get the transformed projection points to the processors where they belong.

For image reconstruction on ZMOB, there is an attractive way to implement the filtered back-projection method. Given P processors and projections, the circular image is partitioned into 2P sectors, and each processor is assigned two opposite sectors, such that each projection bisects each pair of sectors. For an N by N image, each sector pair will contain approximately $\frac{\pi N^2}{4P}$ points (about 50 for a 32 by 32 image with 16 partitions). Each processor is then loaded with the projection data assigned to it. Each point in the sectors will add to a running sum, as the back-projected contribution from that projection, an interpolated value depending on where a line from the point, normal to the projection, falls on the projection. After the first projection is processed, each processor passes those values to its next neighbor, then again to the neighbor two over, and so on (note that in later rounds, the normal each point drops onto the projection takes into account the ray number it is working on).

To calculate the computational, communication, and space complexity of this algorithm, define the following variables:

N = image diameter (N by N image) (and projection length)

P = number of processors (and projections)

p = time to pass one point between processors

C_{int} = time to process one image point (interpolate and sum)

r = time to load one point into ZMOB via the UNIBUS

The computation time is the time for each point in one processor's allocation of the image (2 sectors) to be processed, for each projection:

$$T_{comp} = P\left(\frac{\pi\left(\frac{N}{2}\right)^2}{P}\right)C_{int}$$
$$= \tfrac{1}{4}\pi N^2 C_{int}$$

The communication time will consist of two parts: the time to pass projections between processors, and the time to load the projection data (via the UNIBUS, as shown earlier to be the rate-determining step). Thus,

$$T_{comm} = PN_p + 2rN^2$$

Finally, the amount of memory required is that for the projection and the portion of the image:

$$\text{Memory size} = \frac{\pi\left(\frac{N}{2}\right)^2}{P} + N$$
$$= \frac{\pi N^2}{4P} + N$$

To find how well this algorithm compares to commercial algorithm timings (around 10 sec.), using the following representative values:

$N = 512$ (512 by 512 image, at 1mm resolution)

$P = 256$

$p = 10^{-5}$sec. (10μsec./byte ZMOB transfer rate)

$r = 4x10^{-7}$sec. (400nsec./byte UNIBUS transfer rate)

we get:

$T_{comm} = 1.31 + 0.210 = 1.52$sec.

$T_{comp} = 205776C_{int}$

and for: C_{int} = (1μsec., 10μsec., 100μsec.)

we get: T_{comp} = (0.206sec., 2.06sec., 20.6sec.)

for a total time of: (1.73sec., 3.58sec., 22.1sec.)

For the range of C_{int} values used, which should be realistic since many of the values used in projection normal computation and interpolation may be precomputed instead of computed "on-the-fly", the timings for ZMOB image reconstruction should be very attractive compared to commercial systems.

References

1. K.E. Batcher, Design of Massively Parallel Processor, IEEE Trans. Computers C-29, September 1980, 836-840.

2. T. Kushner, A.Y. Wu, and A. Rosenfeld, Image Processing on ZMOB, TR-987, Computer Science Center, University of Maryland, College Park, MD, December 1980.

$m \backslash \alpha$	1	1/2	1/4	1/8	1/16	1/32	1/64
1	$8.01\text{x}10^{-7}$	$1.60\text{x}10^{-6}$	$3.21\text{x}10^{-6}$	$6.41\text{x}10^{-6}$	$1.28\text{x}10^{-5}$	$2.57\text{x}10^{-5}$	$5.15\text{x}10^{-5}$
2	$4.01\text{x}10^{-7}$	$8.02\text{x}10^{-7}$	$1.61\text{x}10^{-6}$	$3.21\text{x}10^{-6}$	$6.43\text{x}10^{-6}$	$1.29\text{x}10^{-5}$	$2.58\text{x}10^{-5}$
4	$2.01\text{x}10^{-7}$	$4.02\text{x}10^{-7}$	$8.05\text{x}10^{-7}$	$1.61\text{x}10^{-6}$	$3.22\text{x}10^{-6}$	$6.45\text{x}10^{-6}$	$1.29\text{x}10^{-5}$
8	$1.01\text{x}10^{-7}$	$2.02\text{x}10^{-7}$	$4.05\text{x}10^{-7}$	$8.10\text{x}10^{-7}$	$1.62\text{x}10^{-6}$	$3.24\text{x}10^{-6}$	$6.50\text{x}10^{-6}$
16	$5.12\text{x}10^{-8}$	$1.02\text{x}10^{-7}$	$2.05\text{x}10^{-7}$	$4.10\text{x}10^{-7}$	$8.20\text{x}10^{-7}$	$1.64\text{x}10^{-6}$	$3.29\text{x}10^{-6}$
32	$2.62\text{x}10^{-8}$	$5.24\text{x}10^{-8}$	$1.05\text{x}10^{-7}$	$2.09\text{x}10^{-7}$	$4.19\text{x}10^{-7}$	$8.39\text{x}10^{-7}$	$1.68\text{x}10^{-6}$
64	$1.37\text{x}10^{-8}$	$2.73\text{x}10^{-8}$	$5.47\text{x}10^{-8}$	$1.09\text{x}10^{-7}$	$2.19\text{x}10^{-7}$	$4.38\text{x}10^{-7}$	$8.78\text{x}10^{-7}$

Thresholds (in seconds, MPP PE computation time) for MPP preferability to the VAX

TABLE 1

$m \backslash \alpha$	1	1/2	1/4	1/8	1/16	1/32	1/64
1	8.02×10^{-6}	1.60×10^{-5}	3.21×10^{-5}	6.44×10^{-5}	1.29×10^{-4}	2.61×10^{-4}	5.34×10^{-4}
2	4.01×10^{-6}	8.03×10^{-6}	1.61×10^{-5}	3.23×10^{-5}	6.48×10^{-5}	1.31×10^{-4}	2.67×10^{-4}
4	2.01×10^{-6}	4.03×10^{-6}	8.07×10^{-6}	1.62×10^{-5}	3.25×10^{-5}	6.57×10^{-5}	1.34×10^{-4}
8	1.01×10^{-6}	2.03×10^{-6}	4.06×10^{-6}	8.13×10^{-6}	1.63×10^{-5}	3.30×10^{-5}	6.74×10^{-5}
16	5.12×10^{-7}	1.02×10^{-6}	2.05×10^{-6}	4.11×10^{-6}	8.27×10^{-6}	1.67×10^{-5}	3.41×10^{-5}
32	2.62×10^{-7}	5.24×10^{-7}	1.05×10^{-6}	2.10×10^{-6}	4.23×10^{-6}	8.54×10^{-6}	1.74×10^{-5}
64	1.37×10^{-7}	2.74×10^{-7}	5.48×10^{-7}	1.10×10^{-6}	2.21×10^{-6}	4.46×10^{-6}	9.11×10^{-6}

Thresholds (in seconds, MPP PE computation time) for 10-fold speedup over the VAX when using MPP

TABLE 2

	Computation	Communication	Memory
<u>Point operations</u>	$C_m N^2/P$	$2r_B N^2 \log M$	$(N^2/P)\log M$
<u>Local operations</u> (8-neighbors) i = # iterations	$iC_m \log M N^2/P$	$(4p_B i(N\sqrt{P} + 1) +$ $2r_B N^2)\log M$	$(N\sqrt{P} + 2)^2 \log M$
<u>Histogramming</u> c_m^1 = 1st phase time/bit c_m^2 = 2nd phase time bit	$c_m^1 N^2 M \log M/(P\sqrt{P})$ $+ c_m^2(N + 2\log N-1)$ $\cdot M/\sqrt{P}$	$N^2 p_B \log M$ $+ p_B(N + 2\log N-1)$ $\cdot M/\sqrt{P} + 2r_B N^2 \log M$	$(N^2/p + \log(N^2)$ $\cdot M\sqrt{P})\log M$
<u>Co-occurrence matrices</u> (ℓ_1,ℓ_2) = displacement c_m^1 = 1st phase time/bit c_m^2 = 2nd phase time/bit	$c_m^1 N^2 M^2 \log M/(P\sqrt{P})$ $+ c_m^2(N + 2\log N-1)$ $\cdot M^2/\sqrt{P}$	$2N^2 p_B \log M/\sqrt{P}$ $+p_B(N + 2\log N-1)$ $\cdot M^2/\sqrt{P} + (N/\sqrt{P} + \ell_1)$ $\cdot(N/\sqrt{P} + \ell_2) r_B \log M$ $+ N^2 r_B \log M$	$((N/\sqrt{P} + \ell_1)(N\sqrt{P} + \ell_2)$ $+ \log(N^2)M^2/\sqrt{P})\log M$
<u>Discrete transforms</u>	$C_m 2N$	$2Np_B \log M + 2r_B N^2 \log M$	$(N^2/P)\log M$

N^2 = size of (N by N) image	C_m = time to compute one operation, per <u>bit</u>, on MPP
P = number of processors	r_B = UNIBUS transfer rate, per <u>bit</u>
M = number of gray levels	p_B = PE intercommunication rate, per <u>bit</u>

TABLE 3: MPP SUMMARY (MEASURES PER <u>BIT</u>)

	Computation	Communication	Memory
Point operations	$C_z N^2/P$	$2rN^2$	N^2/P
Local operations (8-neighbor) i = # iterations	$iC_z N^2/P$	$4pi(N/\sqrt{P} + 1)$ $+ 2rN^2$	$(N/\sqrt{P} + 2)^2$
Histogramming	$C_z N^2/P$	$Mp(P-1)\log(N^2/P)/$ $(P\log M) + 2rN^2$	$N^2/P + (M\log(N^2/P)$ $+ \log(N^2))/\log M$
Co-occurrence matrices (ℓ_1, ℓ_2) = displacement	$C_z N^2 P$	$M^2 p(P-1)\log(N^2/P)/$ $(P\log M) + (N/\sqrt{P}$ $+ \ell_1)(N/\sqrt{P} + \ell_2) r$ $+ rN^2$	$(N/\sqrt{P} + \ell_1)(N/\sqrt{P} + \ell_2)$ $+ (M\log(N^2/P) + \log(N^2))/$ $\log M$
Discrete transforms	$2C_z N^2 \log N/P$	$p(N^2/p - N^2/P^2)$ $+ 2rN^2$	N^2/P

N^2 = size of (N by N) image	C_z = time to compute one operation, per pixel, on ZMOB
P = number of processors	r = UNIBUS transfer rate (per pixel)
M = number of gray levels	p = conveyor belt transfer rate (per pixel)

TABLE 4: ZMOB SUMMARY (MEASURES PER PIXEL)

M \ P	$\theta(N^2)$	$\theta(N)$	$\theta(C)$
$\theta(N)$	$N + \log N$	$N\sqrt{N}\log N + N\sqrt{N} + \sqrt{N}\log N$	$N^3\log N + N^2 + N\log N + N$
	$N\log N + N + \log N[(+N^2\log N)]$	$N\sqrt{N}\log N + N\sqrt{N} + \sqrt{N}\log N[(+N^2\log N)]$	$N^2\log N + N^2 + N\log N + N(+N^2\log N)$
	$\log N + \log^2 N$	$N\log N + \log^2 N\sqrt{N}$	$N^2\log N + N\log^2 N$
$\theta(C)$	$\frac{1}{N} + \frac{\log N}{N}$	$\sqrt{N} + \frac{\log N}{\sqrt{N}} + \frac{1}{\sqrt{N}}$	$N^2 + N + \log N$
	$N + \frac{\log N}{N}\left[(+N^2)\right]$	$N\sqrt{N} + \sqrt{N} + \frac{\log N}{\sqrt{N}} + \frac{1}{\sqrt{N}}\left[(+N^2)\right]$	$N^2 + N + \log N(+N^2)$
	$\frac{\log N}{N}$	$N + \frac{\log N}{\sqrt{N}}$	$N^2 + \log N$

TABLE 5 MPP Histogramming
complexity: Computation

Communication

Memory

M \ P	$\theta(N^2)$	$\theta(N)$	$\theta(C)$
	C	N	N^2
$\theta(N)$	$\frac{N}{\log N}(+N^2)$	$N(+N^2)$	$N(+N^2)$
	$\frac{N}{\log N}$	N	$N^2 + N$
	C	N	N^2
$\theta(C)$	(N^2)	$\log N(+N^2)$	$\log N(+N^2)$
	$\log N$	$N+\log N$	$N^2+\log N$

TABLE 6: ZMOB histogramming complexity:
Computation

Communication

Memory

Neighborhood	Step	Pass Direction	No. of Pixels Passed	Result
8-neighbor	1	Up	1	$\underline{x}$ x
	2	Right	2	x$\underline{x}$ xx
	3	Down	2	xx x$\underline{x}$ xx
	4	Left	3	xxx x$\underline{x}$x xxx
4-neighbor	1	Up	1	$\underline{x}$ x
	2	Right	1	x$\underline{x}$ x
	3	Down	1	x x$\underline{x}$ x
	4	Left	1	x x$\underline{x}$x x
2 x 2	1	Down	1	x $\underline{x}$
	2	Right	2	xx x$\underline{x}$
8-component	1	Right	1	x$\underline{x}$
	2	Left	1	x$\underline{x}$x
	3	Down	3	xxx x$\underline{x}$
4-component	1	Down	1	x $\underline{x}$
	2	Right	1	x x$\underline{x}$

Figure 1. MPP passing sequences for various types of neighborhoods

Row \ Step	1	2	3	4	5	6	7	8
1	2/0	3/0	4/0	3/0	0/0	6/0	7/0	6/0
2	3/0	4/0	3/0	0/0	6/0	7/0	6/0	2/1
3	4/0	3/1	0/1	6/1	7/1	6/1	2/1	3/2
4	3/0	0/0	6/0	7/0	6/0	2/0	3/0	4/1
5	0/0	6/0	7/0	6/0	2/0	3/0	4/0	3/0
6	6/1	7/1	6/2	2/2	3/2	4/2	3/2	0/2
7	7/1	6/1	2/1	3/1	4/1	3/1	0/1	6/1
8(0)	6/0	2/0	3/0	4/0	3/0	0/1	6/1	7/1

In entry a/b, a = value passing through, b = counter contents. The values are cyclically shifted upward. Each counter adds 1 when the value passing through it is equal to its row number. In this example, there are 8 PEs and 8 gray levels.

Figure 2. Column histogramming example

Step	Row Contents					
0	11	01	00	10	11	10
1	11	01	00	10	11	1$\underline{0}$
2	11	01	00	11	1'$\underline{0}$	$\underline{0}$0
3	11	01	01	1$\underline{0}$	1$\underline{0}$0	00
4	11	01'	1$\underline{0}$	1$\underline{0}$0	$\underline{0}$00	00
5	11	1'$\underline{0}$	1$\underline{0}$0	$\underline{0}$00	000	00
6	11	1'$\underline{0}$0	$\underline{0}$00	000	000	00
7	11	1$\underline{0}$00	000	000	000	00
8	1011	$\underline{0}$000	000	000	000	00

In each entry, bits that have just been passed are underlined; primes denote positions of carry bits

Figure 3. Row totalling example

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 81-0367 — 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) IMAGE PROCESSING ON MPP	5. TYPE OF REPORT & PERIOD COVERED INTERIM
	6. PERFORMING ORG. REPORT NUMBER TR-1007
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 2304/A2 61102F
11. CONTROLLING OFFICE NAME AND ADDRESS Math. & Info. Sciences, AFOSR/NM Bolling AFB Washington, DC 20332	12. REPORT DATE February 1981
	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
Pattern recognition
Parallel processing
Cellular computers
MPP

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

The Massively Parallel Processor is a 128 by 128 array of processing elements that communicate with their horizontal and vertical neighbors by shifting data one bit at a time. This paper discusses the efficient use of MPP for various types of image processing operations, including point and local operations, discrete transforms, and computation of image statistics. A comparison between MPP and ZMOB (a system consisting of 256 microprocessors) is also presented.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DA

FILM

5