Presented at the Nineteenth Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, October, 1981.

BLOCK TRUNCATION CODING ON PASM

LEAH J. SIEGEL **
EDWARD J. DELP *
T. N. MUDGE *
HOWARD JAY SIEGEL **

* University of Michigan Department of Electrical and Computer Engineering Ann Arbor, MI 48109

** Purdue University School of Electrical Engineering West Lafayette, IN 47907

ABSTRACT

Image data compression techniques are an important class of image processing algorithms. Data compression techniques are used in both the efficient storage and transmission of pictures. The implementation of an image coding algorithm known as Block Truncation Coding (BTC) on the PASM parallel processing system is presented. Two forms of BTC are considered, one where the output of the encoder is fixed length and another where the output is variable length. Both the SIMD and MIMD modes of parallelism are considered. The serial and parallel complexities of the coding algorithms are analyzed and compared. The potential for using a parallel processor to meet real-time constraints in image data compression is illustrated.

I. INTRODUCTION

Image data compression is an important discipline that is finding growing application in such diverse areas as video data transmission, the archival storage of images, the storage and transmission of weather satellite data, and remotely piloted vehicle imaging. Characteristic of image data compression problems are large data sets and a large number of arithmetic operations that in principle can be performed concurrently. In many applications real-time constraints require very high processing rates when coding (decoding) an image for data compression (reconstruction). In view of the potential concurrency in the coding problem, using a parallel processing computer having a large number of processors appears to be the most natural solution. Traditionally, the cost of such machines has prohibited their consideration, however, recent advances in VLSI circuit technology have made such architectures feasible. In paper the detailed design of a data compression algorithm to perform Block Truncation Coding (BTC) for a parallel processing computer, PASM, is presented. The rate at which BTC can be performed is shown to increase with the number of processors in approximately a linear fashion. Thus any real-time constraints can be met if a sufficient number of processors are available.

II. BLOCK TRUNCATION CODING

BTC [1,2] is a data compression scheme based on applying a block adaptive two-level (one bit) moment-preserving quantizer to image data [3]. BTC has proved to be very competitive with classical transform coding techniques, such as the Chen and Smith cosine transform algorithm, as far as image quality [4,5]. The basic BTC algorithm requantizes $n \times n$ nonoverlapping blocks of pixels in an image into two gray levels. These gray levels are chosen such that the sample

This research was supported by the National Science Foundation under Grants ECS-7909016 and MCS-8009315, by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR-78-3581, and by the Defense Mapping Agency, monitored by the United States Air Force Rome Air Development Center Information Sciences Division, under Contact No. F30602-78-C-0025 through the University of Michigan. The United States Government is authorized to reproduce and distribute reprints for governmental purposes not-withstanding any copyright notation hereon.

mean and mean square value (or sample standard deviation) are identical with the original $n \times n$ block.

One proceeds by first dividing the original picture into $n \times n$ blocks (we have used $n=4$ for our examples). Blocks are the coded individually, each into a two level signal. The levels for each block are chosen such that the first two sample moments are preserved. Let $k=n^2$ and let $x_1, x_2, \ldots, x_k$ be the values of the pixels in a block of the original picture. Let

$$m_1 = \frac{1}{k} \sum_{i=1}^{k} x_i \tag{1}$$

be the first sample moment

$$m_2 = \frac{1}{k} \sum_{i=1}^{k} x_i^2$$

be the second sample moment and let

$$\sigma^2 = m_2 - m_1^2$$

be the sample variance.

As with the design of any one bit quantizer, we need to find a threshold and two output levels for the quantizer. We have chosen the sample mean as the threshold for this application. Other choices of thresholds are discussed in [1,2]. Therefore if

$$x_i \geq m_1 \quad \text{output} = y_2 \tag{2}$$

or

$$x_i < m_1 \quad \text{output} = y_1$$

$$\text{for } i = 1, \ldots, k.$$

where

y_1 and y_2 are the low and high

output levels, respectively. The output levels y_1 and y_2 for a two-level non-parametric moment preserving quantizer are found by solving the following equations:

Let q = number of x_i's greater than m_1

We then have

$$km_1 = (k-q)y_1 + qy_2 \tag{3}$$

$$km_2 = (k-q)y_1^2 + qy_2^2$$

Equation 3 is readily solved for y_1 and y_2:

$$y_1 = m_1 - \sigma \left[\frac{q}{k-q} \right]^{\frac{1}{2}} \tag{4}$$

$$y_2 = \bar{m}_1 + \sigma \left[\frac{k-q}{q} \right]^{\frac{1}{2}}$$

Each block is the described by $\bar{m}_1$, σ , and an $n \times m$ bit plane consisting of 1's and 0's depending on whether a given pixel is above or below $\bar{m}_1$. The receiver (decoder) reconstructs the image block by calculating y_1 and y_2 from Equation 4 and placing those values in accordance with the bits in the bit plane.

For each 4x4 block the output of the encoder consists of the quantized sample mean and sample standard deviation and a 16 bit "bit plane" (represented by 1 bit/pixel). The sample mean and sample standard deviation are *jointly* quantized to 10 bits using an empirically derived quantization scheme detailed in [2]. This scheme is based on the fact that if $\bar{m}_1$ is very small or very large then σ will be small and hence we can assign fewer bits to σ. Therefore, the original 16 pixels are represented by 26 bits or 1.625 bits/pixel compared to 8 bits/pixel in the uncoded state. A variable length encoding extension of BTC exists which can yield a further compression by representing those blocks where the sample standard deviation is zero (or very small) by *only* the sample mean quantized to 8 bits. This variation on the BTC algorithm allows the average data rate to be reduced to a value between 0.5 bits/pixel and 1.625 bits/pixel. The exact rate with depend upon the number of blocks in a given image that meet the small variance condition. In empirical tests with high resolution aerial reconnaissance images the percentage of blocks meeting this condition were about 10%. For "head and shoulder" face images the number of blocks meeting the small variance condition can be as high as 30%. We are of course paying for this reduction in data rate by having the output of the encoder for each block be sometimes 26 bits and at other times 8 bits. This can cause problems when channel noise is present.

The BTC decoding algorithm consists of taking the bit plane for each block and substituting for those pixels greater than the sample mean (as given by the bit plane) the gray level value, y_2, and substituting for those less than the sample mean the gray level value, y_1, such that the sample mean and sample standard deviation are the same as the original block of pixels [1]. The reconstructed image appears slightly enhanced but does not look block-like, as would be expected, because the brightness has been preserved.

Let us quickly review the basic BTC encoding algorithm.

(1) The image is divided in small non-overlapping blocks such as 4 x 4.

(2) The first and second sample moments are computed.

(3) A bit plane is constructed such that each pixel location is coded as "one" or a "zero" depending on whether that pixel is greater than $\bar{m}_1$.

(4) The bit plane, $\bar{m}_1$, and σ are sent to the receiver.

(5) The picture block is reconstructed such that $\bar{m}_1$ and σ (alternatively $\bar{m}_2$) are preserved. That is pixels in the bit plane that are "0" are set to "y_1" and the "1"'s are set to "y_2" as in Equation 4. For example, suppose a 4 x 4 picture block is given by the following:

$$x_{ij} = \begin{matrix} 121 & 114 & 56 & 47 \\ 37 & 200 & 247 & 255 \\ 16 & 0 & 12 & 169 \\ 43 & 5 & 7 & 251 \end{matrix}$$

so

$$\bar{m}_1 = 98.75$$

$$\bar{x} = 92.25$$

$$q = 7$$

and

$$y_1 = 16.7 \approx 17$$

$$y_2 = 204.2 \approx 204$$

the bit plane is:

1	1	0	0
0	1	1	1
0	0	0	1
0	0	0	1

The reconstructed block becomes:

204	204	17	17
17	204	204	204
17	17	17	204
17	17	17	204

and the sample mean and variance are preserved.

BTC has been used successfully to code various types of pictures including aerial reconnaissance images, multilevel graphics, and video signals using a multitemporal extension of BTC. A variation of BTC has also been used in speech coding at low bit rates.

III. PASM OVERVIEW

Two types of parallel processing systems are SIMD and MIMD. SIMD (single instruction stream-multiple data stream) machines [7], such ad Illiac IV [8] and STARAN [9], typically consist of a set of N processors, N memories, an interconnection network, an a control unit. The control unit broadcasts instructions to the processors, and all enabled ("turned on") processors execute the same instruction at the same time. Each processor executes instructions on a separate data stream. The interconnection network allows interprocessor communication. An MIMD (multiple instruction stream - multiple data stream) machine [7] also typically consists of N processors, N memories, and an interconnection network, but each processor can follow an independent instruction stream (e.g., C mmp [10] and Cm* [11]). As with SIMD architectures, there is a multiple data stream. PASM is a partitionable SIMD/MIMD multimicroprocessor system being designed for image processing and pattern recognition applications [6]. It will consist of N processors which can be structured as one or more independent SIMD and/or MIMD machines, where N is a power of two and may be as large as 1024.

Figure 1 is a block diagram overview of the PASM system. The heart of the PASM system is the Parallel Computation Unit (PCU), which contains N processors, N memory modules, and an interconnection network. The Micro Controllers are a set of microprocessors which act as the control units for the PCU processors in SIMD mode and orchestrate the activities of the PCU processors in MIMD mode. Control Storage contains the programs for the Micro Controllers. The Memory Management System controls the loading and unloading of the PCU memory modules. The Memory Storage System stores these files. It consists of

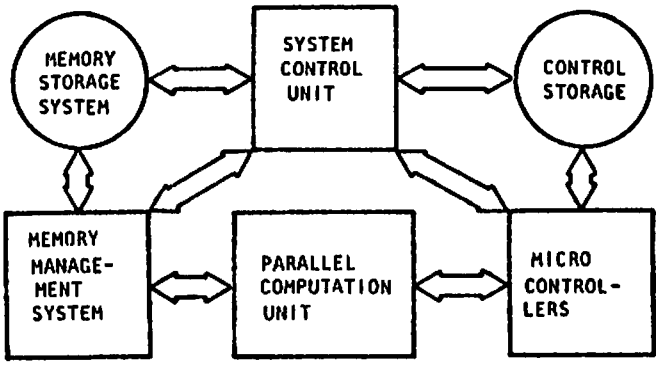

Figure 1. Block Diagram Overview of PASM.

multiple secondary storage devices connected in a fashion that will allow parallel loading/unloading. The System Control Unit is a conventional machine, such as a PDP-11, and is responsible for the overall coordination of the activities of the other components of PASM.

The organization of the PCU is shown in Figure 2. The PCU processors are microprocessors that perform the actual SIMD and MIMD computations. The PCU memory modules are used by the PCU processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. A pair of memory units is used for each PCU memory module so that data can be moved between one memory unit and secondary storage while the PCU processor operates on data in the other memory unit (double buffering). A processor and its associated memory module are termed a processing element (PE). The interconnection network provides a means of communication among the PCU's PE's. Either the Augmented Data Manipulator network [12] or the Generalized Cube Network [13] will be used in PASM.

This brief summary of the PASM organization is provided as background for the following sections. For further details of the system architecture, see the references indicated.

IV. PARALLEL BLOCK TRUNCATION CODING

Coding

Parallel implementation of BTC on PASM is divided into two cases. The fixed bit rate method is structured as an SIMD process. For the variable rate method, both SIMD and MIMD implementations are examined.

For data compression (coding) under both methods, an image having $4 \lambda N$ lines (λ N "line groups"), $\lambda \leq 1$, will be allocated to the PEs such that PE i holds line groups λ i through $\lambda (i+1)-1$, $0 < i < N$. Each PE will therefore hold complete 4x4 blocks, and will not need data from any other PEs. For smaller images (number of line groups < N), the most efficient approach to coding will be to use the partitionability of PASM, allowing several images to be processed in parallel. The partition sizes will be chosen so that each PE holds at least one line group. In both cases (number of line groups < or > N), the double buffering of the PASM

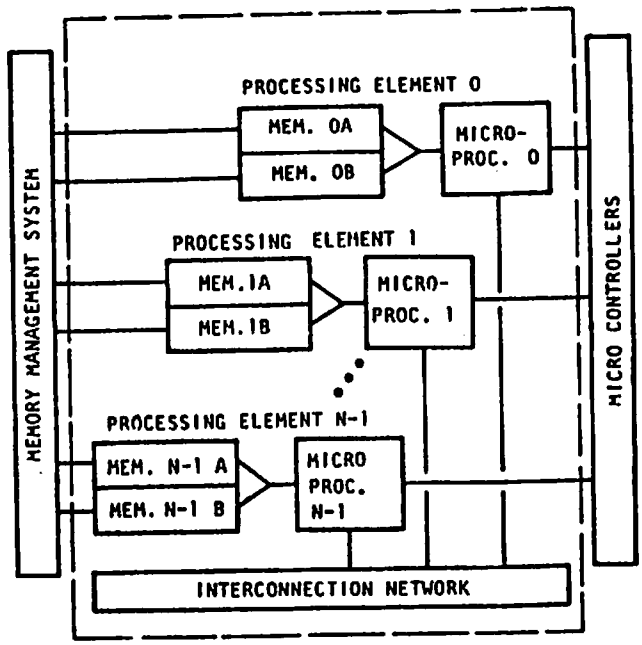

Figure 2. PASM Parallel Computation Unit.

memories will be used so that while one image is being processed, another can be obtained from the Memory Storage System.

For fixed rate coding PASM will operate in SIMD mode with each PE simply performing the BTC algorithm on its portion ("stripe") of the image. That is, all PEs will execute the same BTC algorithm in lockstep, following the same instruction stream; each PE will process a different stripe of the image. Because all PEs will code the same number of blocks and because no inter-PE communications are needed, the execution time for the SIMD algorithm on an N-PE machine will be 1/N-th that of the serial algorithm.

For the variable rate scheme, more operations are required to code a block to 26 bits than to 8 bits, so different PEs may not be performing the same coding operations. Figure 3 shows the flow of computations for the variable rate coding. The number of operations performed to generate the 8-bit code is a basically subset of the operations to generate the 26-bit code. In an SIMD machine each PE would execute the operations appropriate for the block which it is coding. This could be implemented using data conditional masks to enable and disable PEs. A mask statement of the form:

where<data condition>**do**<instructions>

would cause each PE to evaluate the <data condition>, using the data in its own

memory. Only those PEs in which the condition was true would execute the <instructions>; the remaining PEs would be disabled until the next block of instructions was broadcast from the control unit. Therefore, although less time is required to generate the 8-bit code, in an SIMD implementation the time to generate the 26-bit code would be required any time even one PE needed to generate the 26-bit code.

```
compute m1, m2, v^e
if v^e≈0 then code m1        /* 8-bit code */
else begin                   /* 26-bit code */
     compute v
     construct bit plane
     code m1 and v
end
```

Figure 3. Steps to code each block using variable rate scheme.

Consider the best and worst case performance of the SIMD implementation for a given percentage mixture of 8-bit and 26-bit blocks. In SIMD mode, each PE would be coding a block from the same column of the image, but from a different line group. The best case will occur when the data is such that whenever any PE generates an 8-bit code all PEs generate an 8-bit code. In this case, a factor of N speedup over the serial algorithm will be obtained. however, in actual images, this situation will rarely exist. The worst case performance will occur when the data is such that at any point in the algorithm, at least one PE is generating a 26-bit code. Let B be the number of blocks in the images, α the percent of 8-bit blocks, $(1-\alpha)$ the percent of 26-bit blocks, t_8 the time to generate an 8-bit code, and t_{26} the time to generate a 26-bit code. Then the serial execution time will be:

$$T_1=[\alpha t_8 + (1-\alpha)t_{26}]B.$$

The worst case N-PE SIMD execution time will be:

$$T_N=t_{26}B/N$$

and the speedup will be:

$$T_1/T_N=[\alpha t_8/t_{26} + (1-\alpha)]N. \qquad (5)$$

Four typical aerial reconnaissance photographs were analyzed for the purpose of determining the usage of the 8-bit and 26-bit codes. Evidence from these pictures indicates that $\alpha \approx 10\%$, and that for any given column, it is likely that at least one PE will need to generate a 26-bit code. The low variance blocks tended to occur primarily near the edges of the image, and were clustered in groups of four or five blocks. Therefore, except possibly for coding along the left and right edges of the picture, it is likely that for any given column of the image, at least one PE will be generating a 26-bit code. PEs generating an 8-bit code will be idle for the extra time required by the slower coding scheme. Despite this disadvantage of SIMD processing, the speedup for $\alpha = 0.1$ is greater than 90% of N.

In terms of speedup, the absolute worst case for the SIMD mode of processing will occur for an image where 1/N of the blocks generate a 26-bit code, and these blocks are distributed so that whenever one PE is generating a 26-bit code, all others need only to generate an 8-bit code. Under this pathological worst case percentage mixture and data distribution, speedup will be:

$$T_1 / T_N \approx (t_8 / t_{26}) N.$$

Because SIMD processing will rarely be able to take advantage of the shorter time needed to code a block to 8 bits, MIMD operation is considered, with each PE coding the blocks in its portion of the image asynchronously with respect to the other PEs. Best case performance will occur when the low variance blocks occur in the image in such a way that they are evenly distributed among the PEs. The 8-bit blocks need not be in the same column of the image for the savings in execution time to be realized. If each PE holds the same number of 8-bit blocks, the total execution time will be the same for all PEs, and for such images, execution time of the N-PE MIMD algorithm will be faster than the serial algorithm by a factor of N. Worst case performance will occur if all of the blocks in some PE require the 26-bit code. The coding of the image will not be completed until that PE has finished; meanwhile, the remaining PEs will be idle. The worst case speedup for a given α will be given by Equation 5. Thus the best and worst case speedups will be the same as for the SIMD algorithm. However, because the requirements on the relative placement of the 8-bit blocks is less stringent for good speedup under the MIMD approach, for realistic images the MIMD algorithm will in general be faster than the SIMD algorithm. Moreover, for any given image, the SIMD algorithm will never be faster than the MIMD algorithm. Any data distribution that will improve the speed of the SIMD algorithm will also improve the speed of the MIMD algorithm. The converse is not true. For this reason, for variable rate coding, the MIMD implementation method will be preferable. The exact difference between the SIMD and MIMD execution times for a given image will be a function of factors such as data distribution, α, and system architecture implementation details.

Decoding

For decoding under the fixed rate scheme, the code is assigned to PEs in an analogous pattern to the data assignment for coding. Each PE holds code for λ line groups, corresponding to 4 λ image lines. If SIMD processing is used, each PE decodes its portion of the image, however speedup will be slightly less than a factor of N. This is a result of the fact that among those pixels being decoded simultaneously, some PEs will need to assign the output level y_8. In an SIMD system, this could be implemented using data conditional masking. In this case, a mask would be of the form

where<data condition>**do**<instructions 1>**elsewhere do**<instructions 2>

Each PE would evaluate the <data condition> using data in its own memory. Those PEs in which the condition is true would execute <instructions 1>. Then the remaining PEs would execute <instructions 2>. The time to execute the "where-elsewhere" statement would be the sum of the times to execute <instructions 1> and <instructions 2>. Therefore, instead of one output level assignment per pixel as in the serial algorithm, it will most often be the case that time for two assignments will be needed in the SIMD algorithm. If the fixed rate decoding is implemented as an MIMD process with each PE asynchronously decoding its stripe of the image, a speedup of N over the serial algorithm will be obtained. The relative speeds of SIMD versus MIMD processing for this task would depend on the actual architecture implementation details.

The decoding operations in the variable rate method are shown in Figure 4. As in the variable rate coding, the processing of 26-bit blocks requires more

computation than the processing of 8-bit blocks. However, whereas the operations for coding to 8 bits were basically a subset of the operations to code to 26 bits, for the decoding process entirely different operations are performed for the two types of codes. In an SIMD system, this could be implemented using "where-elsewhere" statements. For the variable rate decoding, unless all PEs were decoding blocks of the same type, the time to decode one set of N blocks (i.e., one block per PE) would be the time to decode a 26-bit block plus the time to decode an 8-bit block. For this reason, for decoding in the variable rate method MIMD processing is preferable. Two allocations schemes are considered.

---

if number of bits = 26

then begin

compute output levels y_1 and y_2

decode each pixel

end

else assign value of m_1 to each pixel

Figure 4. Steps to decode each block under the variable rate scheme.

In the first MIMD decoding scheme, the coded data is assigned in a manner corresponding to the data assignment for coding (code for λ line groups per PE). Because the decoding of 26-bit blocks requires more computation than decoding of 8-bit blocks, performance will vary as it did in coding. Best and worst case performance will be analogous.

An alternative data allocation divides the coded bits (as opposed to blocks) evenly among the PEs. This scheme will reduce the possible imbalance in computation among the PEs, since PEs holding primarily 26-bit blocks will hold and decode fewer blocks than PEs holding a large number of 8-bit blocks. For example, a PE holding only 26-bit blocks will decode approximately one-third as many blocks as a PE holding only 8-bit blocks. Each PE will receive complete block encodings, i.e., the bits representing a block will not be split across two PEs. One way to accomplish this is to include with each encoded image a count of the number of 8-bit and 26-bit encodings. This information can be used to estimate the total processing time required and to guide the data allocation when the encoded image data is read into the parallel secondary storage units (or directly into the PCU memory modules, via the primary/secondary storage bus system, in a real-time application).

For images having a very irregular distribution of low variance blocks, this bit-based allocation will help to equalize the execution time of the PEs. However, the scheme may have an additional overhead of restoring the image line structure after decoding is completed. The extent of this overhead will depend on the next task for which the data will be used. If the decoded image is to be displayed on a device with a word serial input channel, there will be no additional overhead. The data from the PEs will just be read into the device in sequence, either after being written back into the secondary storage system or directly from the PCU memory modules (using the primary/secondary storage busses).

V. CONCLUSION

Ways in which the PASM parallel processing system can be used to perform two types of BTC have been presented. Both SIMD and MIMD algorithms have

been considered. Due to the fact that each block of the image is processed independently, no inter-processor communication or synchronization is needed, and so, on the average, the MIMD approach will be faster than the SIMD. The parallel algorithms described here are not limited to PASM. They could be used with any SIMD or MIMD system.

Future work will include a closer look at the processing rates demanded by real-time coding/decoding at TV frame rates, and how this impacts the algorithms used by the memory management system (see Figure 2).

VI. REFERENCES

[1] E. J. Delp and O. R. Mitchell, "Image Compression Using Block Truncation Coding," *IEEE Trans. on Communications*, Vol. COM-27, No. 9, Sept. 1979, pp. 1335-1342.

[2] O. R. Mitchell and E. J. Delp, "Multilevel Graphics Representation Using Block Truncation Coding," *Proceedings of the IEEE*, Vol-68, No. 7, July 1980, pp. 868-873.

[3] E. J. Delp and O. R. Mitchell, "Some Aspects of Moment Preserving Quantizers," IEEE Communications Society's International Conference on Communications (ICC), June 1979, Boston, pp. 7.2.1-7.2.5.

[4] O. R. Mitchell, S. C. Bass, E. J. Delp, T. W. Goeddel and T. S. Huang, "Image Coding for Photoanalysis," *Proceedings of Society for Information Display*, Vol. 21, No. 3, 1980, pp. 279-292.

[5] W. H. Chen and C. H. Smith, "Adaptive Coding of Monochrome and Color Images," *IEEE Trans. on Communications*, Vol. COM-25, No. 11, Nov. 1977, pp. 1285-1292.

[6] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A Partitionable SIMD/MIMD System for Image Processing and Pattern Recognition," *IEEE Trans. on Computers*, Vol. C-30, Dec. 1981.

[7] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, Vol. 54, Dec. 1966, pp. 1901-1909.

[8] W. J. Bouknight, et al., "The Illiac IV system," *Proc. IEEE*, Vol. 60, Apr. 1972, pp. 369-388.

[9] K. E. Batcher, "STARAN parallel processor system hardware," *AFIPS 1974 NAT'l. Comp. Conf.*, Vol. 43, May 1974, pp. 405-410.

[10] W. A.Wulf and C. G.Bell, "C.mmp - a multi-miniprocessor," *AFIPS 1972 FJCC*, Dec. 1972, pp. 765-777.

[11] R. J. Swan, S. H. Fuller, and D. . Siewiorek, "Cm*: a modular multi-microprocessor," *AFIPS 1977 Nat'l. Comp. Conf.*, June 1977, pp. 637-644.

[12] H. J.Siegel and R. J. McMillen, "Using the augmented data manipulator network in PASM," *Computer*, Vol. 14, Feb. 1981, pp. 25-33.

[13] H. J.Siegel and R.J. McMillen, "The multistage cube: a versatile interconnection network," *Computer*, to appear.