Image Coding Using the Multimicroprocessor System PASM[1]

T. N. Mudge*
Edward J. Delp*
Leah J. Siegel**
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

Data compression techniques are used in both the efficient storage and transmission of pictures. The implementation of a particular image coding algorithm known as Block Truncation Coding (BTC) on the PASM parallel processing system is reviewed as representative of a class of image coding algorithms. 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 number of processors needed to meet the real-time constraints in image data compression is illustrated using timing specifications for currently available state-of-the-art components.

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. In view of the potential concurrency in the coding problem, using a parallel processing computer having a large number of processors appears to be a natural solution. Traditionally, the cost of such machines has prohibited their consideration, however, recent advances in VLSI circuit technology have made such architectures feasible. Most of the special purpose computer architectures for image processing have addressed the area of image enhancement and, in particular, pattern recognition and scene analysis [1]. The use of special purpose architectures for image data compression has not been as widely discussed.

This paper is an extension of work previously reported in [2]. It will examine the real-time implementation of parallel Block Truncation Coding (BTC) algorithms and, given current technology, estimate the number of processors needed to meet the real-time constraint of video rates (60 frames per second). Data rates as low as 1.625 bits/pixel can be achieved with no appreciable distortion.

The detailed design of a data compression algorithm to preform BTC on a parallel processing computer, PASM, is reviewed. The rate at which BTC can be performed is shown to increase with the number of processors at approximately a linear fashion thus any real-time constraints can be met if a sufficient number of processors are available.

Block Truncation Coding

BTC [3,4] is a data compression scheme based on applying a block adaptive two-level (one bit) moment-preserving quantizer to image data [5]. BTC has proved to be very competitive with classical transform coding techniques from the standpoint of image quality [6,7]. The basic BTC algorithm requantizes nxn nonoverlapping blocks of pixels in an image into two gray levels. These gray levels are chosen such that the sample mean and mean square value (or sample standard deviation) are identical with the original nxn block.

One proceeds by first dividing the original picture into nxn blocks (we have used $n=4$ for our examples). Blocks are then 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:

$$\overline{m}_1 = \frac{1}{k} \sum_{i=1}^{k} x_i \qquad (1)$$

be the first sample moment. Let:

$$\overline{m}_2 = \frac{1}{k} \sum_{i=1}^{k} x_i^2 \qquad (2)$$

be the second sample moment and, let:

[1] 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. F3602-81-C-0193 through the Southeastern Center for Electrical Engineering Education. The United States Government is authorized to reproduce and distribute reprints for governmental purposes not-withstanding any copyright notation herein.

$$\bar{\sigma}^2 = \bar{m}_2 - \bar{m}_1^2 \qquad (3)$$

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 [3,4].) Therefore, if:

$$x_i \geq \bar{m}_1 \quad output = y_2 \qquad (4)$$

or, if:

$$x_i < \bar{m}_1 \quad output = y_1 \qquad (5)$$

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

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 $\bar{m}_1$; we then have:

$$k\bar{m}_1 = (k-q)y_1 + qy_2 \qquad (6)$$

$$k\bar{m}_2 = (k-q)y_1^2 + qy_2^2 \qquad (7)$$

Equations 6 and 7 are readily solved for y_1 and y_2:

$$y_1 = \bar{m}_1 - \bar{\sigma}\left[\frac{q}{k-q}\right]^{\frac{1}{2}} \qquad (8)$$

$$y_2 = \bar{m}_1 + \bar{\sigma}\left[\frac{k-q}{q}\right]^{\frac{1}{2}} \qquad (9)$$

Each block is then described by $\bar{m}_1$, $\bar{\sigma}$, and an nxn 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 Equations 8 and 9 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 [4]. This scheme is based on the fact that if $\bar{m}_1$ is very small or very large then $\bar{\sigma}$ will be small and hence we can assign fewer bits to $\bar{\sigma}$. 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.

In summary then, 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. The reconstructed image appears slightly enhanced but does not look block-like, as would be expected, because the brightness has been preserved. An example of coding a representative image block is presented in [3,8].

PASM Overview

Two types of parallel processing systems are SIMD and MIMD. SIMD (single instruction stream-multiple data stream) machines [9], such ad Illiac IV [10] and STARAN [11], 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 [9] 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 [12] and Cm* [13]). 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 [14]. 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

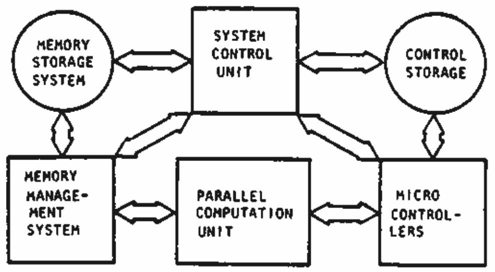

Figure 1. Block Diagram Overview of PASM.

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 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 [15] or the Generalized Cube Network [16] 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.

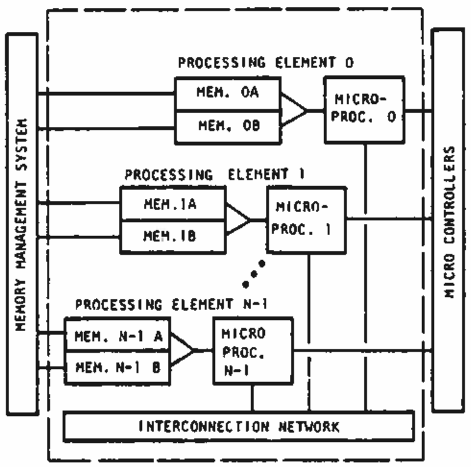

Figure 2. PASM Parallel Computation Unit.

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 (λN "line groups"), $\lambda \geq 1$, will be allocated to the PEs such that PE i holds line groups $\lambda(i-1)+1$ through λi, $0 < i < N$. 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 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.

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

```
/* compute $\overline{m}_1$, $\overline{m}_2$, $\overline{\sigma}^2$ */
if $\overline{\sigma}^2 \approx 0$
then   code $\overline{m}_1$            /* 8-bit code */
else begin
                         /* 26-bit code */
    compute $\overline{\sigma}$
    construct bit plane
    code $\overline{m}_1$ and $\overline{\sigma}$
end
```

Figure 3. Steps to code a block: variable rate scheme.

in the algorithm, at least one PE is generating a 26-bit code. Let B be the number of blocks in the image, α the fraction of 8-bit blocks, $(1-\alpha)$ the fraction 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:

$$T_1 = [\alpha t_8 + (1-\alpha)t_{26}]B. \qquad (10)$$

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

$$T_N = t_{26}B/N \qquad (11)$$

and the speedup will be:

$$T_1/T_N = [\alpha t_8/t_{26} + (1-\alpha)]N. \qquad (12)$$

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:

$$T_1/T_N \approx (t_8/t_{26})N \qquad (13)$$

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. 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 12. 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.

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 4λ 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_1 and others will need to assign the output level y_2. 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 allocation schemes are considered.

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 is discussed in [2].

```
if
number of bits = 26
    then begin
        compute output levels $y_1$ and $y_2$
        decode each pixel
      end
    else
assign value of $\overline{m}_1$ to each pixel
```

Figure 4. Steps to decode a block: variable rate scheme.

Proposed Processor Architecture

In this section we will outline a possible architecture for the PE's that make up PASM's PCU. Based on this architecture we will estimate the number of processors (i.e., the value of N) necessary to perform BTC at video data rates if the block size is 4x4. As our benchmark we will assume an image of 512x512 single byte pixels is to be processed in approximately 16 ms (this corresponds to video data rates with no interlacing). Furthermore, we will assume currently available component technology to implement the PE's. The implementation of the interconnection network will be deferred--its use is not central to the BTC algorithms.

A possible PE architecture is comprised of the following components:

(1) A 64Kx16 bit dynamic memory.

(2) A third generation 16 bit microprocessor--our figures will be based on Motorola's 68000 with a 10MHz clock.

(3) A high speed bipolar 4Kx16 bit static memory--our analysis will be based on the 40 ns of Fairchild's F93453 4Kx1 bit static memory chip.

(4) A high speed multiply-accumulate chip (MAC)--our analysis will be based on the 115 ns multiply-accumulate time of TRW's TDC1010 MAC chip.

The dynamic memory is organized as a two 32K buffers to allow the double buffering described earlier. The 68000 manages the PE's internal activity and interaction with the rest of the system including the double buffering. The 68000 has a 16 Mbyte address space. The dynamic RAM occupies the low 128K bytes and the static memory occupies the next 8K bytes of address space. An additional four words (8 bytes) follow the static memory area. The remaining space is unused.

The MAC chip is intended to perform most of the computation. Intermediate results are stored in the static memory--it functions as a high speed scratch pad. Associated with the static RAM are four 16 bit registers that are in the address space of the 68000 (see above). Two of these indicate the starting addresses of two vectors of contiguous words in the static RAM whose inner product is to be formed by the MAC chip (the vectors may or may not have memory locations in common). The third register indicates the address at which to place the result. The fourth register indicates how long the vector operations are to be--for well formed vector operations only one "length" register is needed. The 68000 can initiate the following operations by the MAC chip on data in the scratch pad:

[M1]Inner products between two vectors; time = $200l$ ns.

[M2]Element-wise multiplies between two vectors; time = $200l$ ns.

[M3]Summation of elements of one vector; time = $200l$ ns.

Where l = number of words in the vector. The timing figures assume that the movement of data between the scratch pad and the MAC chip is overlapped. Loading up a location in the scratch pad must be done as a memory-to-memory data move by the 68000. Typically, the time to do memory-to-memory moves is 2-3 μs (assuming a 10MHz clock).

Examination of the coding and decoding algorithms (see Equations 1 through 9) indicates that coding requires the most computation and the time to perform it is thus the critical factor. Therefore, we shall examine it in further detail with reference to the above proposed PE architecture.

To code a 4x4 block requires the following operations:

(1) To form $\overline{m}_1$ requires 15 adds, and one divide. The adds can be done by the MAC chip (M3 with $l=15$) in 3 μs. The divide is a 4 bit right shift. The shifting must be done by the 68000 on a double word (the output of the MAC chip). This will take about 4 μs. Thus for one block we can compute $\overline{m}_1$ in about 8 μs.

(2) To form $\overline{m}_2$ requires 16 multiplies, 15 adds, and one divide. Again, using the MAC chip to form the sum of squares (M1 with $l=16$) and the 68000 to perform the divide by shifting results in a compute time of about 8 μs.

(3) To form $\bar{\sigma}^2$ requires one multiply, and one subtract. Again, using the MAC chip (M1 with $l=2$) results in a compute time of about 0.4 μs. The joint encoding of the sample mean and sample standard deviation requires that up to 7 bits of the standard deviation be determined from the above computed value of the sample variance, i.e., a square root operation must be performed. The square root can be performed by table lookup followed by a single iteration using the Newton-Raphson algorithm. The table lookup requires that 512 words of the scratch pad be loaded with precomputed data. The most significant 8 bits of the variance are used to reference two words in the table. The first contains a 4 bit approximation to the standard deviation and the second contains an approximation to the reciprocal of the standard deviation. These can be combined using the MAC chip (one multiply and accumulate--M1 with $l=1$). The 68000 is also required to perform a 1 bit right shift (divide by two). Adding the above steps together results in a total compute time to extract the sample standard deviation is about 2 μs.

(4) To form the bit plane requires 16 comparisons. This can be accomplished using the MAC chip as a subtract unit (M3 with $l=1$ repeated 16 times). The resulting compute time is about 3 μs.

(5) One last timing consideration needs to be taken into account, and that is the time required to copy the block into the scratch pad and to copy $\overline{m}_1$, $\overline{m}_2$, and

the bit plane back to the dynamic memory for transmission. This copying will be done by the 68000 and will take about 8 μs.

Assuming the image is already in the PCU's memory and thus directly addressable by the microprocessors, the above list of operations together with associated housekeeping operations will take about 30 μs. Since a 512x512 image contains 16K 4x4 blocks one PE will take about 0.5 s to perform the BTC algorithm on a complete frame. This figure assumes a worst case situation in which every block must be coded into 26 bits. Therefore, to meet our real-time constraint of video data rates will require N=32 (to the nearest power of 2) PE's.

Conclusion

This paper used the example of Block Truncation Coding to illustrate how parallel processing might be used to help meet real-time constraints in image coding, such as the need to process at video data rates. A PE architecture that formed the basis of the multimicroprocessor system PASM was outlined. Its proposed architecture was made up from inexpensive off-the-shelf state-of-the-art components. Timing figures derived from manufacturers specifications indicated that a system with 32 PE's would meet the video data rate requirement. The hardware for such a system would, therefore, be inexpensive. The use of multiple processors will impact overall system cost much more significantly at the software level.

References

[1] P. E. Danielsson, and S. Levialdi, "Computer architectures for pictorial information systems," *IEEE Computer*, Vol. 14, Nov. 1981, pp. 53-67.

[2] L. J. Siegel, E. J. Delp, T. N. Mudge, and H. J. Siegel, "Block truncation coding on PASM," *Proceedings 19th Annual Allerton Conference on Communication, Control, and Computing*, Oct. 1981, pp. 891-900.

[3] E. J. Delp, and O. R. Mitchell, "Image compression using block truncation coding," *IEEE Trans. on Communications*, Vol. COM-27, Sept. 1979, pp. 1335-1342.

[4] O. R. Mitchell, and E. J. Delp, "Multilevel graphics representation using block truncation coding," *Proceedings of the IEEE*, Vol-68, July 1980, pp. 868-873.

[5] E. J. Delp, and O. R. Mitchell, "Some aspects of moment preserving quantizers," *Proceedings of IEEE Communications Society's International Conference on Communications (ICC)*, May 1979, pp. 7.2.1-7.2.5.

[6] 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, July 1980, pp. 279-292.

[7] W. H. Chen, and C. H. Smith, "Adaptive coding of monochrome and color images," *IEEE Trans. on Communications*, Vol. COM-25, Nov. 1977, pp. 1285-1292.

[8] D. J. Healy, and O. R. Mitchell, "Digital video bandwidth compression using block truncation coding," *IEEE Trans. on Communications*, Vol. Com-29, Dec. 1981, pp. 1809-1817.

[9] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, Dec. 1966, pp. 1901-1909.

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

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

[12] W. A. Wulf and C. G. Bell, "C.mmp - A multiminiprocessor," *Proceedings of the AFIPS 1972 FJCC*, Dec. 1972, pp. 765-777.

[13] R. J. Swan, S. H. Fuller, and D. Siewiorek, "Cm*: A modular multi-microprocessor," *Proceedings of the AFIPS 1977 Nat'l. Comp. Conf.*, June 1977, pp. 637-644. Vol. 14, Feb. 1981, pp. 25-33.

[14] 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, pp. 934-947.

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

[16] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *IEEE Computer*, Vol. 14, Dec. 1981, pp.65-76.