JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 9, 297–303 (1990)

Design and Analysis of a Multiprocessor for Image Processing

J. R. PARKER AND T. R. INGOLDSBY

Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N-1N4

I. INTRODUCTION

Image processing is concerned with the manipulation of pictorial data, usually by computer. An image is generally represented as a two-dimensional array of numbers (pixels) each of which corresponds to a brightness or color value at some point in the image. There are many operations that are commonly performed on images in order to enhance them for some specific purpose: filtering, level transformations and thresholding, edge and region enhancements, feature recognition, statistical measurements, and so on. A characteristic of many of these operations is *shift invariance*. Operations such as edge enhancement and noise reduction are usually applied to small regions of the object image at one time. In this case, modifying one portion of the image has little or no effect on other portions some distance away. The processing of each region is independent of all of the others and can be carried out in any order. If many processors are available, then each region could be operated on concurrently with a resulting savings in the time needed [2, 3, 5, 6, 10, 16, 18, 19].

Some operations can be applied to rows or columns of an image: the Fourier transform is an example. To compute a two-dimensional Fourier transform, the transforms of the rows of the image are computed, and then the transforms of the resulting columns are computed. A row or column can be thought of as a rectangular region of the image, and again the regions can be processed concurrently if the processors are available.

Operators that perform contrast enhancement functions usually require only the gray level at a single pixel to compute a result. These are *point operations*, and concurrency is again possible, although it is unlikely that one processor per pixel could be allocated. What could be done is to use arbitrary rectangular regions, chosen to make the number of regions equal to the number of processors. Indeed, it is likely that this will be done for all of the operations discussed above. Unless a computer system has been designed with a particular operation in mind, it would be improbable that the number of processors available would exactly equal the number of regions to be processed. While enormous time savings can be had by using architectures designed around specific algorithms, it is the common bus architectures for multiprocessing that are mostly commonly built in practice, due to the ease of construction and low cost. In this paper, a multiprocessor system is described that has been designed with image processing as its specific task, while attempting to minimize cost and complexity.

II. COMMON BUS ORGANIZATION

Figure 1 shows an example of a common bus multiprocessor. Each processor connects to the same bus and can access devices and memory connected to the bus. In addition, each processor may have its own local memory. For example, consider a bus that can transfer 10 Mbyte/s. In this case, assuming no local memory, the upper limit on speed of the system is 10 MIPS in an absolutely ideal case: no bus contention, no data needed, and short instructions. The same system using a local program memory only, with no access to common data, has no upper limit to speed; every added processor increases the amount of work done. In practical situations, both shared memory and local memory are used in an attempt to reduce contention for the bus [1].

Bus contention is the root cause of most of the trouble with common bus systems. Special hardware is used to connect each processor to the bus in order to request bus control, and if the bus is busy, waiting occurs. Two things can be done to reduce contention: reduce traffic, and control transfers from a central location.

Given that each processor has its own program memory, reducing bus traffic in an image processing application has an obvious implementation. Each pixel should be sent only once across the bus to a processor, and then possibly be sent back. This is not always possible when dealing with rectangular regions of an image, since the edges of the regions will need to be transmitted many times. What must be done to cut down on the redundant traffic is to allow each processor to possess its own data memory as well, and at least enough to hold any region that might reasonably be expected. A good size is two times the amount of storage needed to store either a row or column of the image, whichever is larger. This permits Fourier transform calculations on each row or column, for example.

Control of bus transfers centrally offers a number of advantages, the most important of which is the elimination of

0743-7315/90 $3.00

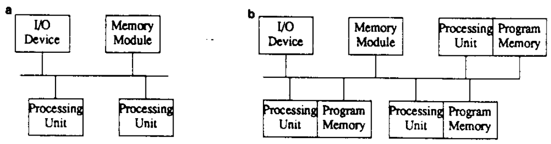

FIG. 1. Common bus multiprocessor systems. (a) Common bus multiprocessor. (b) Common bus multiprocessor with local code memory.

contention, as such. In this configuration, the master control unit (MCU) is responsible for transferring data from the data memory, or frame buffer, into the local program memories of the processors. The processors do not request data; the MCU sends a region into local data memory and then allows the processor to proceed. The MCU then repeats this process with the next processor, and so on. When the first processor has finished, it sets a bit that the MCU can see; now there are two possibilities. Either the MCU can load *all* processor memories and then start over again, looking for set completion bits, or it can start over again when the *first* processor sets its completion bit. In the latter case, it can be seen that no more processors will actually be used than the MCU can reasonably deal with, and this number will depend on the region size and algorithm being implemented.

One other modification can be made to improve things further. When a processor has completed its task, the MCU must unload the local data memory into the global frame buffer and load a new region into the local memory. While this is happening, the processor involved is idle. If two buffers are used, and both are loaded initially, then the MCU simply swaps buffers and starts the processor immediately, then unloads the processed buffer while the processor works on the other. This doubles the local memory requirement while reducing the idle time of the processors to nearly nil.

When the system is started there is an initialization time during which some processors will be idle. This corresponds to a pipeline startup time. There is a similar time when all processors have finished, again corresponding to the time taken to unload a pipeline. Other than at these two moments, none of the processors being used will be idle, and the bus will be working at near maximum capacity. This method of handling the local data memories will be referred to as a *distributed cache* memory [4, 17].

III. A SAMPLE MULTIPROCESSOR DESIGN

The basic idea behind the Vista processor, a common bus multiprocessor based on the distributed cache concept, is to break a two-dimensional signal, or image, into regions, and to let a high-speed satellite processor deal with each region. As far as possible, "off-the-shelf" components were to be used in the design, since this would decrease the cost and increase ease of construction and maintenance. Vista's purpose is to perform any signal processing operation on a two-dimensional signal, and so it must be user programmable and not designed to perform a particular transformation.

The major features of interest of this system are the use of multiple identical special-purpose satellite processors, the lack of proper shared memory, the extensibility to any number of processors, and the potential for fault tolerance. Instead of shared memory, a distributed cache memory system is used, so that each satellite processor operates on its own pages in its own memory, which can be moved in and out of a global image memory upon demand. Memory contention is not an issue, but the maximum bus transfer speed limits the speed of the cache memory.

The Vista processor consists of a central multiplexer processor (MCU) connected on a common bus with N signal processors. In our design, the multiplexer function is performed by an Intel 8086 MPU, and the bus is the Intel Multibus. Since many systems use the Multibus, the Vista processors should be easily connected to other systems (Sun, Wicat, Dandilion, etc.) and used as a peripheral processor. Fig. 2 shows an outline of the Vista.

The Vista image processing system uses a number of processor units referred to here as *Tachyons.* These are independent high-speed computers specially suited for computation-intensive applications, such as image processing. Each Tachyon is based on a digital signal processor, currently the Texas Instruments TMS32010 [20]. Any available multiprocessor could be used here, but the TMS has a number of advantages [9].

The TMS32010 is a special-purpose microprocessor capable of operating at 5 MIPS. This speed is due, at least in part, to a modified Harvard architecture which allows instruction and data fetches to be carried out effectively simultaneously. The designers of the TMS32010 have provided instructions that make it possible to move data in or out of the on-chip data RAM from either off-chip program memory or I/O ports. With the exception of these special transfer instructions, most other instructions are single-cycle (200-ns) operations.

Throughput is further improved by means of a carefully tailored instruction set which makes common signal processing operations execute in a single cycle. For example, it

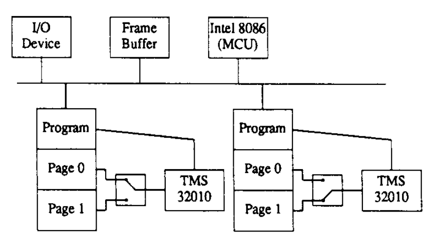

FIG. 2. Outline of the Vista structure.

is possible to multiply two values, accumulate a result, and move another result to memory all in a single cycle. Unfortunately, the price paid for these features is program and data memory space very limited in size. Program memory, located off-chip, is 4K words and data memory, located on-chip, a mere 144 words. Because of the high execution speed of the TMS32010, the off-chip program RAM must be fast, static, expensive memory. The TMS32010 is not suited to the processing of large data sets without a supporting architecture. The Tachyon architecture is one such scheme for dealing with this problem.

The Tachyon processor units had as their design goals the ability to process image data quickly using a wide variety of algorithms at a relatively modest cost. This was achieved by a method that may be considered as either a sophisticated double-buffering system or a paged memory system, something like the FLIP [8]. The 4K-word program address space of the TMS32010 was divided into two areas, one dedicated to storage of programs, constants, tables, etc., and the other for storage of small blocks of image data. A data block size of 1K words was chosen, which allows for significantly sized sections of an image to be stored. For example, depending on the image resolution approximately one to four rows or columns of pixels could be stored. This would be useful in algorithms such as the fast Fourier transform, in which it is customary to process individual rows and columns of pixels in an image.

Although program memory is divided into two areas, it is physically organized as three banks of memory, one bank of 3K and two banks of 1K. Either one or two banks may be active (connected to the TMS32010) at a given time, one of which must be a 1K bank. The banks which are not active, i.e., connected to the TMS32010, are instead connected to the MCU (and optionally a DMA controller) via the Multibus. The MCU is able to load a program into the 3K memory bank and then command the TMS32010 to execute it. While the TMS32010 is processing the data found in one data bank, the master CPU can unload already processed data and load a new portion of the image to be analyzed into the other data bank. Since the task of the MCU is relatively simple it can service a number of Tachyon units quickly enough to keep them constantly busy.

Philosophically, the Tachyon method may be viewed in two ways. From the point of view of the system master, it is a simple double-buffer arrangement. From the perspective of the TMS32010, however, it appears much more as a virtual paged memory system: the Tachyon pages data from a small, fast static RAM area to a large, inexpensive frame buffer.

While the hardware for the Tachyon system was being constructed, the system was simulated. This served two purposes. First, the viability of the system could be tested, including some details of interconnection, and approximate timings could be obtained. Second, software could be written for the system before the hardware was complete and thus speed up the entire implementation process by using concurrency.

IV. ANALYSIS AND SIMULATION

In order that the Vista system be considered successful, an improvement in performance over a single-processor system must be evident when performing image processing tasks. It is also important to be able to predict with some confidence the performance measures of interest for Vista acting on a particular problem. It is for these reasons that the Vista system was simulated and analyzed carefully both during and after construction [14].

There are three phases to a computation on the Vista machine. First, the processors are loaded with subimages; this corresponds to the startup phase of a pipeline and will be referred to as the *load cycle.* For each processor the first buffer is loaded, the processor is started, and then the second buffer is loaded. Since each subimage takes T_b seconds to be loaded and there are N_p processors each with two buffers, the load cycle will require $2N_pT_b$ units of real time. The way that the MCU normally operates is to continue to load additional processors until there are none remaining or until the first processor loaded completes its task. If the time needed for one processor to finish its computation is T_t then the first processor actually works for T_t time units during the load cycle; the second processor works for $T_t - 2T_b$ units, and so on. The total work done in the load cycle is $N_p(T_t - T_b(N_p - 1))$ processor units.

During the second phase of computation the MCU cycles through all of the processors and restarts them, unloads the completed buffer, and reloads the buffer with new data. This continues until no data remain and is called the *reload* cycle. One subcycle involves reloading all N_p processors once and requires $2N_pT_b$ units of real time. Since all processors are working during this cycle, a total of N_pT_t processor units of work is done per subcycle. There will be C subcycles needed to complete a task, where C is a rational number that can be calculated.

The final phase is called the unload cycle, during which the final results are collected. The first half of the unload cycle, $U1$, involves restarting the processors and unloading a buffer; the processors are still active. The second half, $U2$, involves unloading the last buffer from each processor with the processor idle. The analysis of the unload cycle is more involved, and details may be found in [12]. However, the unload subcycle $U1$ appears to require $N_pT_b + W$ time units, where W is the MCU idle time during $U1$. Cycle $U2$ always requires T_t time units.

Hence, the total time needed by a Vista system to solve some problem is

$$2N_pT_b + 2CN_pT_b + (N_pT_b + W) + T_t,$$

where C is the number of reload cycles required. Both C and W can be calculated given buffer sizes and transmission times. The conditions for which this formula is correct are

1. $C \geqslant 1$,
2. $T_t - 2T_b N_p \leqslant 0$,
3. $k \geqslant \lceil (T_t + T_b)/2T_b \rceil$,

which are the normal operating conditions for Vista. In more simple terms:

1. There are many more blocks of data to be processed than there are processors.
2. The time needed by a processor to complete one calculation is greater than the time needed to transmit one block to the processor.
3. The Vista processor has a minimum number of processors available to dedicate to the problem.

There are at least seven other cases to be examined in detail and new expressions could be derived for each case. Rather than do this a Simula [21] simulation of Vista was constructed that would predict execution times in all cases. An instruction-level model was also built so that actual Vista code could be executed [11].

V. PROGRAMMING VISTA

A five-point median filter, a Sobel operator, and a two-dimensional FFT [15] will be used to illustrate the use of the Vista system. The MCU runs the same program whatever problem is to be solved. This program is basically the data distribution program described above and is equivalent to the call

call DO_VISTA (image, N, M, n, m, ovn, ovm),

to which we pass the image to be processed, its size (rows, columns), the size of the subimage to be sent to the processors, and the overlap of the subimages in each direction. For example, the median filter requires an overlap of one row and one column since the last row of a subimage will be the first row of the next. For the FFT, no overlap is needed; a subimage will be a row or a column.

All other processors run the same program, which is sent across the bus and loaded concurrently. For the median filter, the program of Fig. 3 was used. The program for the Sobel operator is seen in Fig. 5. The TMS Fortran compiler [13] is by no means optimal, and this program generates 467 words.

To perform the median filter, the original 256×256 image is broken up into overlapping subimages of 16×16 pixels. There are 324 of these, and each one requires 65.2 ms to process on one of the processors. The way our system is arranged at the moment (no DMA), a subimage requires 2.91 ms to transmit.

The FFT of the same image is performed by sending all rows, then all columns, to the processors. The FFT program is hand-coded assembler, needing 22 ms per processor.

The Sobel operator is very much like a convolution, and the timings can be interpreted to be approximately the times needed to perform a convolution with a 3×3 matrix. To compute the Sobel operator, subimages of size 19×53 were sent; this most nearly fills up to Tachyon "page." The 256×256 image then requires 75 subimages, each needing 143 ms to processes.

VI. RESULTS

The Vista system actually constructed consists of one MCU, one processor, and one 500K frame buffer. No DMA device is used, so the MCU must perform the actual data reads and writes. This is not ideal but is sufficient for assessment purposes.

The Vista system performs the FFT in 12.6 s, which compares with 92 s on a VAX 11/780. A second processor

```
c ***************************************************************
c  TMS Fortran version of VISTA 5-point median filter
c
        common datin, datout
        integer i,j,k,n,m,temp(5),ii,t, p
        integer datin(256), datout(256)
        equivalence (datin, 16#bb8#), (datout,16#c68#)
c
c P is index into DATIN, a two dimensional array. Initially P points
c to DATIN(2,2); it runs to the second last element, skip 2, and
c starts again in the second column of the next row. The last row
c is skipped altogether. This process simply skips the outer layer
c of pixels.
c
        p = 18
        j = 2
 10      continue
         do 100 i=2,15
c
c Load TEMP, in data memory, with data from input sub-image,
c which resides in program memory (one of the transfer buffers)
c
          temp(1) = datin(p)
          temp(2) = datin(p-1)
          temp(3) = datin(p+1)
          temp(4) = datin(p-16)
          temp(5) = datin(p+16)
c
c Remove the smallest two data points, let k=3rd smallest value.
c
          n = 5
 30       k = temp(1)
          ii = 1
          do 50 m = 1,n
           if (temp(m) .ge. k) goto 50
           ii = m
           k = temp(m)
 50       continue
          t = temp(ii)
          temp(ii) = temp(n)
          temp(n) = t
          if (n .eq. 3) goto 60
          n = n-1
          goto 30
 60       continue
          datout(p) = k
          p = p+1
 100     continue
c
         p = p+2
         j = j + 1
         if (j .le. 15) goto 10
        end
```

FIG. 3. The Tachyon program for the median filter.

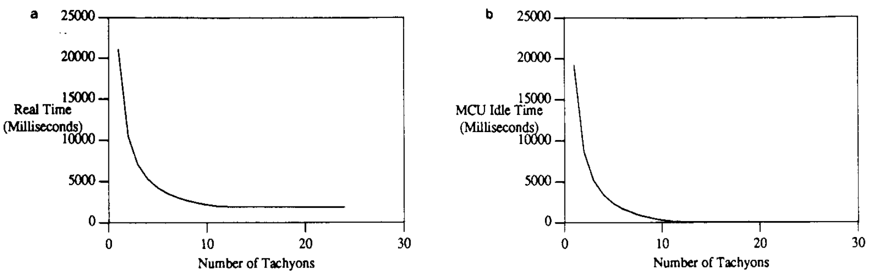

FIG. 4. Performance measures for Vista/5-point median filter. (a) Number of Tachyons vs MCU real time. (b) Number of Tachyons vs MCU idle time.

would cut the time in half, to 6.3 s. Our prototype would allow four processors at most to be fully utilized, at which point the FFT would take 3.2 s. Note that this is a restriction of the actual prototype, and not of the Vista architecture: the prototype lacks a DMA controller.

The simulation model in this paper would predict a time

```
c ***************************************************************
c        TMS Fortran version of VISTA Sobel edge detector
      c
      integer i,j,in, out,a0,a1,a2,a3,a4,a5,a6,a7,a,aa,x,y
      integer nn,mm,datin(1007), xrows(38)
      common nn,mm,datin
      equivalence (nn, 16#0c00#)
      equivalence (mm, 16#0c01#)
      equivalence (datin, 16#0c02#)
c
c    A0  A1  A2  Sobel edge detector uses the pixel
c    A7   f  A3  number scheme shown to the left. The
c    A6  A5  A4  values X=(A2+2A3+A4)-(A0+2A7+A6)
c                and    Y=(A0+2A1+A2)-(A6+2A5+A4)
c
      in = 1
      out = 19
c...
c...      do 20 j = 2, mm-1
c... Construct a DO loop - the compiler generated ones are slower.
      j = 1
    1  j = j + 1
      if (j .ge. mm) goto 21
c...
        xrows(in+17) = 53+j
        aa = j
c...
c...        do 10 i=2, nn-1
c...
        i = 1
    2   i = i +  1
        if (i .ge. nn) goto 11
c...
          a = aa + 53
c...A0 is image(i-1,j-1) = -1 row and -1 col = -53-1 = a-54
          a0 = datin(a-54)
          write (1) a0
c...A1 = image(i-1,j) = -1 row = a-53
          a1 = datin(a-53)
          write (1) a1
c...A2 is image(i-1,j+1) = -1 row and +1 col = -53 + 1 = a-52
          a2 = datin(a-52)
          write (1) a2
c...
c...A3 is image(i,j+1) = this row, +1 col = a+1
          a3 = datin(a+1)
          write (1) a3
c...A4 is image(i+1,j+1) = +1 row and +1 col = +53+1 = a+54
          a4 = datin(a+54)
          write (1) a4
c...A5 is image(i+1,j) = +1 row = a+53
          a5 = datin(a+53)
          write (1) a5
c...A6 is image(i+1,j-1) = +1 row and -1 col = +53-1 = a+52
          a6 = datin(a+52)
          write (1) a6
c... A7 is image(i,j-1) = this row, -1col = a-1
          a7 = datin(a-1)
          write (1) a7
c...
          x = (a2+a3+a3+a4) - (a0+a7+a7+a6)
          y = (a0+a1+a1+a2) - (a6+a5+a5+a4)
          write (1) x
          write (1) y
c...
          if (x .lt. 0) x = -x
          if (y .lt. 0) y = -y
c...
          xrows (in + i-2) = x+y
          write (1) x+y
          aa = a
c...
c... 10 CONTINUE
  10    goto 2
  11    continue
c...
        if (j .eq. 2) goto 15
c...
c... Copy the OUT row into the DATIN array.
c...
        a = xrows(out+17)
        i = 1
 13       i = i + 1
          if (i .ge. nn) goto 14
c...           do 12 i=2, nn-1
          datin(a) = xrows(out+i-2)
          a = a + 53
          goto 13
 14     continue
c...
c... Let OUT=IN and IN=OUT, and repeat.
c...
 15     continue
        i=in
        in=out
        out=i
c...
c... 20 CONTINUE
 20 goto 1
 21 continue
c...
    a = xrows(out+17)
    do 30 i=2,nn-1
      datin(a) = xrows(out+i-2)
      a = a + 53
 30 continue
    end
```

FIG. 5. TMS Fortran program for the Sobel operator.

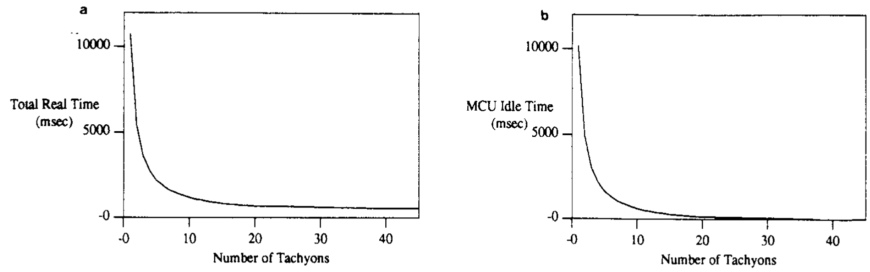

FIG. 6. Performance measures for the Vista/Sobel operator implementation. Simulation of the Tachyon multiprocessor: (a) Execution times-entire image; (b) MPU idle time vs number of Tachyons.

of 11.28 s to perform the FFT. The difference may be explained, in part, by the fact that the TMS processor has several instructions that cannot be timed accurately. These instructions cause the internal pipeline to be cleared, increasing the execution unpredictably by up to a factor of four. Because the prototype Vista has only one TMS processor the analytic model was not applied. Transforming columns has a larger MCU overhead due to greater complexity in indexing an element; the array is stored in row major order.

Much of the problem with the prototype is caused by the lack of DMA capability on the bus used (Multibus [7]). Assuming a bus rate DMA controller as part of the MCU, the figures become more impressive. Each processor can transform a row or column in 0.022 s, and transmission of the entire image across the bus and back requires 0.05 s. Thus, the best that Vista can do is to use 112 processors and compute the FFT in 0.1 s. This is an effective computation rate of 350 MIPS using a common bus.

The results for the five-point median filter are somewhat less impressive for a number of reasons. The main reason is that the filter operates on a two-dimensional subimage of a larger image, and all pixels on the boundary of a subimage must be transmitted at least twice. As well, the program for the filter was written in FORTRAN instead of the hand-coded assembler used for the FFT. Some of the performance data, mainly obtained through simulation, appear in Fig. 4.

The time needed to complete the filter on a 256×256 image using only one processor was 21.2 s, as opposed to 11.2 s on a VAX. When two processors are used, the time drops to 10.6 s, and for the best case of 23 processors the job is complete in 1.88 s; idle time for this case was estimated at only 0.04% of the total. Note that 1.88 s is the time needed by our prototype for bus transmission alone.

The Sobel operator can be computed in 0.57 s using 37 processors at best. With only one processor 1.07 s is needed, and this drops to 0.118 s at a reasonable 10 processors. Again, overlap between subregions causes the retransmission of pixels, which causes extra overhead. Figure 6 summarizes the results.

Given sufficient processors, image processing problems can be executed in nearly the time taken to transmit the image across the bus and back. Only when there are too few processors does the MCU have to wait. If the problem can be broken up into operations on subimages then the Vista system can be applied to it; operations such as rotations and circular harmonic transforms cannot be made to work on subimages and so are more difficult to compute using Vista.

The currently implemented system, using the very quick TMS32010, requires a lot of board space due to the need to use very fast chips; for example, the fastest memory chips we could find had a small capacity, and so the chip count increased. Still, it would be possible to put four TMS Tachyons on one Multibus board. If another processor were used (e.g., 6809) then eight or ten Tachyons could occupy a board. Custom chips would also decrease the chip count. It is possible to place the multiplexer logic onto one VLSI chip, and while work on this has not proceeded very far, it is clear that the chip count will drop significantly.

REFERENCES

1. Abu-Sufah, W., Hasmann, H., and Kuck, D. On input/output speedup in tightly coupled multiprocessors. *IEEE Trans. Comput.* **C-35**, 6 (June 1986).
2. Ahuja, N. Multiprocessor pyramid architectures for bottom up image analysis. *IEEE Trans. Pattern Analy. Mach. Intell.* **PAMI-7**, 1 (Jan. 1985).
3. Armstrong, J. L. Programming a parallel computer for robot vision. *Comput. J.* **21**, 3 (Aug. 1978).
4. Bhuyan, L. On the performance of loosely coupled multiprocessors. *Proc. 11th Annual Symposium on Computer Architecture*, Ann Arbor, MI, June 1984.
5. Briggs, F. A., Fu, K. S., Hwang, K., and Patel, J. H. A shared resource multiple microprocessor system for pattern recognition and image processing. In Fu, K. S., and Ichikawa, I. (Eds.). *Special Computer Architectures for Pattern Processing.* CRC Press, Cleveland, OH, 1982.
6. Gannon, D. B., and Van Rosendale, J. On the impact of communications complexity on the design of parallel numerical algorithms. *IEEE Trans. Comput.* **C-33**, 12 (Dec. 1984), 1180–1194.
7. IEEE. Proposed microcomputer system bus standard. IEEE Computer Society Subcommittee Microcomputer System Bus Group, Oct. 1980.
8. Luetjen, K., and Gemmer, P. FLIP: A flexible multiprocessor system for image processing. *Proc. Fifth International Conference on Pattern Recognition*, Miami, Dec. 1980.

9. NEC Electronics Digital Signal Processor Data Sheet. 1982 Microcomputer Division Catalog; NEC PD7720, p. 551.
10. Ni, L. M., and Jain, A. K. A VLSI systolic architecture for pattern clustering. *Trans. Pattern Anal. Mach. Intell. IEEE* **PAMI-7,** 1 (Jan. 1985).
11. Parker, J. R. The TMS 32010 emulation system. Department of Computer Science Report, University of Calgary.
12. Parker, J. R. Analysis and simulation of a common bus multiprocessor. Department of Computer Science Report, University of Calgary.
13. Parker, J. R. A subset Fortran compiler for a modified Harvard architecture. *ACM SIGPLAN Notices* **21,** 9 (Sept. 1986).
14. Parker, J. R., and Ingoldsby, T. R. Common bus multiprocessor design for signal processing applications. Department of Computer Science Report, University of Calgary.
15. Pavlidis, T. *Algorithms for Graphics and Image Processing.* Comput. Sci. Press, Rockville, MD. 1982.

Received August 18, 1988

16. Preston, K., Jr., and Uhr, L. (Eds). *Multicomputers and Image Processing.* Academic Press, New York, 1982.
17. Rudolph, L., and Segall, Z. Dynamic decentralized cache schemes for MIMD parallel processors. *Proc. 11th Annual Symposium on Computer Architecture.* Ann Arbor, MI, June 1984.
18. Selfridge, D. B., and Mahakian, S. Distributed computing for vision: Architecture and benchmark test. *IEEE Trans. Pattern Recognition Mach. Intell.* **PAMI-7,** 5 (Sept. 1985), 623–626.
19. Siegel, L. J., Siegel, H. J., and Feather, A. E. Parallel processing approaches to image correlation. *IEEE Trans. Comput.* **C-31,** 3 (Mar. 1982), 208–217.
20. Texas Instruments. TMS32010 user's guide. Texas Instruments Ltd, Dallas TX, 1983.
21. Unger, B. W., and Parker J. R. An operating system simulation and implementation language. *Proc. Conferences on Simulation, Measurement, and Modeling of Computer Systems,* Boulder, CO, 1979.