An Associative Processor Array Designed for Computer Vision

A.W.G. Duller

R.H. Storer

A.R. Thomson

M.R. Pout

E.L. Dagless

Dept. of Electrical and Electronic Engineering
University of Bristol
Bristol BS8 1TR, UK.

The architecture of a new associative processor array chip, GLiTCH, is outlined. The suitability of such arrays for image processing and computer vision work is stressed, and a general purpose vision processing module, currently being designed, is described. Effective programming of bit-serial processor arrays requires an efficient library of low-level routines, especially for operations such as data routing and multi-bit arithmetic. The current state of a number of these primitive operations is described and they are used to solve two image processing tasks.

The use of processor arrays for vision processing is now well established, see [3] for example, and mapping one pixel of a low level image to each processor in a large processor array is an obvious way of utilizing very large scale fine-grained parallelism. Due to the general regularity of SIMD (Single Instruction Multiple Data) arrays they are ideal for VLSI design. The work to be described has as its long term goal a system which will allow associative processor arrays to be defined, simulated and then produced in silicon automatically from the high level definition data. In this way the design cycle time for processor arrays could be reduced to the point where application specific designs could be fabricated to solve a particular problem. The research that is described in this paper is the preliminary work that is required to achieve such a goal. At present the results consist of the basic simulation tools, VLSI designs and a test chip which is about to be manufactured.

We begin with a general discussion of associative processors and their use of content-addressable memory. The design decisions that have been taken to produce the current implementation are explained. The position of associative processors in a taxonomy of processors is explained and the innovations that have been made to the associative processor array concept are outlined.

The most effective use of processor arrays requires an efficient library of routines to perform low-level operations. The operations considered in this paper are generalised data movement routines and reduction type functions which produce single results from an array of data. The multi-bit arithmetic functions are not considered but are available in the current simulation system. The importance of these library

Figure 1: Floorplan of a GLiTCH chip

routines is demonstrated by their impact on two image processing applications: histogramming and convolution with Laplacian of Gaussian (LoG) operators of varying sizes. It is shown that even for low-level tasks such as these highly efficient solutions can be achieved.

Finally a list of applications is given which have been implemented on the current GLiTCH simulation.

ASSOCIATIVE PROCESSOR ARRAYS

In common with most fine-grain SIMD architectures, associative processor arrays (APA) [7] contain a large number of simple processing elements (PEs) executing instructions broadcast from a single controller. APAs are distinguished from other SIMD arrays by the type of local memory linked to each processor. In an APA content addressable memory (CAM) is used which means that individual PEs are identified by the contents, or part of the contents, of their local memory rather than by a co-ordinate system which specifies their physical location [8]. In a content addressable memory, such as those used in page translation tables or cache systems, only the first data item matched is returned to the host processor. However, in an APA each row of CAM has a processor connected to it which allows all of the matched data items to be processed in parallel. A qualifying pattern is broadcast by the controller, to identify a subset of PEs which obey the instructions that follow. In addition writing data back to local memory can be performed several bits at a time.

SIMD architectures such as the DAP [9], are sometimes termed Logic in Memory Arrays, since the total processing power of the machine is distributed through the total memory; each processor using its own RAM for local storage. This type of architecture increases the memory access bandwidth by widening the data path. The use of CAM in each PE increases the memory access bandwidth further by improving their access time to local memory, in a similar way to cache memory in a uniprocessor system. In an APA the processing power is further distributed by placing the matching logic within the memory cells themselves. The result is that each PE can 'fetch' two one-bit arguments and add them in a single machine cycle. For searching functions the advantage is obvious, since a search for a multi-bit argument also takes only one cycle.

Since VLSI CAM is larger and consumes more power than RAM it is quite clear that with currently conceivable technology the amount of CAM that could be accessed directly by each PE is likely to remain insufficient, on its own, for many applications. This is especially true for the complex vision processing tasks that make use of large databases, or high precision arithmetic computation. Thus the use of a combination of RAM and CAM in each processor in the way described above is one way of producing machines capable of rapid execution of these tasks.

THE GLITCH CHIP DESIGN

GLiTCH (Goes Like The Clappers, Hopefully) is an APA designed for computer vision (see Figure 1). A VLSI design means that a large number of PEs can be made available cheaply on a single chip. However, this can mean that a large number of pins are needed to avoid a data transfer bottleneck at the chip boundary.

The current design of GLiTCH took the following factors into consideration:-

- The number and complexity of the processing elements determine the range of algorithms that can be implemented.
- Processing elements need to communicate with each other; the most efficient interconnection scheme in terms of bandwidth conflicts with the need for high packing density and reasonable pin count.
- Input and output of the array data must not be a limiting factor on performance.
- The design has to be testable.

The factors listed above are conflicting, and so some form of compromise had to be reached.

The use of a large number of single-bit processing elements is attractive since it allows flexibility in the precision of the data used in any computation. In addition 'clever' algorithms for arithmetic routines such as square root and logarithm can compute results a single bit at a time [4].

Figure 2: A GLiTCH based vision system

As images are two dimensional, processor arrays for image processing are often mesh connected, each processor directly connected to its 4, 6 or 8 immediate neighbours. Unfortunately, closely packed processing elements and a limited number of pins to a chip package mean that complex interconnection schemes become difficult to implement. For N^2 PEs on a chip, $4N$ pins are required for a 4-connected mesh. Furthermore, the mesh chosen may not be the optimum connectivity for all the problems the machine is applied to.

The adopted solution for GLiTCH (more details of the design decisions taken can be found in [1]) achieves a high PE density and low pin count by connecting the processing elements only in one dimension. By having a barrel shift mechanism built into the CAM array, any 4, 6 or 8 connected mesh can be simulated within the chip, a single bit is passed between any virtual neighbours in one to three machine cycles.

The initial GLiTCH chips will contain 64 processing elements, each with 68 bits of local CAM. Similar chips with more PEs are possible without increasing the pin count.

The data input and output problem is alleviated by threading a data shift register through all of the processing elements (see below). In this way data I/O can be performed in parallel with computation.

Because of the regularity of the structures required by an APA the design cycle has been relatively short compared with that of a uniprocessor of similar dimensions. Designing future APA's targetted at specific applications should be even faster since the basic cells are already available.

A GLITCH BASED VISION SYSTEM

A computer vision system based on the GLiTCH chip will consist of an array of chips with supporting hardware, as shown in figure 2. Any number of GLiTCH chips can be connected in a linear array.

Threaded through the array is the 8-bit data shift register (DSR) which brings digitised video or other array data from an input framestore and takes processed

array data to the output framestore. The DSR can perform this operation while the PEs are processing a previously loaded section of the array data. Only 16 machine cycles are then needed to exchange the data in the DSR with 8 bits from each PE's CAM.

Program instructions for the chips from a microcode store are delivered by an instruction sequencer which also provides addresses for program data (match and write patterns) in the 32 bit wide data store and may branch on the outcome of a number of condition flags collected from the processor array.

Scalar processing is provided by a host transputer which can transfer data to and from the program data stores. A scalar register can shift and test a scalar value from the program data store for scalar-vector arithmetic operations.

At present the transputer is acting only as a convenient building block which allows a collection of GLiTCH chips to be controlled. However, as is shown in figure 3 the vision module that will eventually be developed is one that has a dedicated controller dealing with the GLiTCH. The transputer in the system enables a number of these modules to be easily linked to allow the dissemination of results to a global processor or to neighbouring modules. In this way it is envisioned that low and intermediate level image processing will be performed in the vision module using Inmos A110 chips and GLiTCH arrays. The data from these modules can then be passed to a global processor which allows the integration of results. The higher level modules may also include GLiTCH arrays since there is evidence that the architecture can perform maximal join graph matching efficiently.

Inter-chip Data Routing

Between each chip and its neighbours is a custom routing/support chip controlling a 32 bit inter-chip data path which allows fast transfer of tag bits between distant PEs: a system with 16 GLiTCH chips moves 16×32 bits of data per 50ns machine cycle, or a data movement bandwidth of 1.28 Gigabytes per second. When used with the chip's internal barrel shifter, this helps to offset the disadvantages of 1D connectivity.

Note that there is no degradation in data movement bandwidth as the number of PEs per chip increases, because the distance (in terms of chips) the data has to move in one cycle will decrease proportionately.

Other logic in the support chip identifies the one chip in the array which is producing data during a read operation providing efficient 'random access' of information in the array [10].

It is possible to attach RAM to each support chip which can then be used as secondary store for the GLiTCH chips, CAM data can be temporarily transferred to this RAM to make room for higher resolution processing within the PEs. A full-custom or semi-custom routing chip including fast RAM may be developed in the future.

LOW LEVEL OPERATIONS

To enable algorithms to be evaluated for the GLiTCH architecture a simulation system has been written which allows array modules containing variable numbers of PEs. A library of routines is being developed, including data movement, multi-bit integer and floating point arithmetic and reduction operators, which facilitates the efficient description of algorithms at a level above assembly language.

The most obvious disadvantage with a linear array of processing elements is the low communication bandwidth between distant PEs. To some extent with the GLiTCH architecture this problem is reduced by the provision of the barrel shifting mechanism. However, for some basic algorithms there can still be a large disparity between the amount of data routing that has to be performed and the amount of computation. Data routing has to be seen as an overhead in this kind of architecture. For this reason, reduction operators and general data movement algorithms are considered below.

Reduction operators, those that take an array as input and produce a result which is of reduced rank, by their nature make poor use of processor arrays in general. Examples of reduction operators are summation, maximum, minimum, first non zero in an array etc. The first non zero in an array is catered for by a hardware network and there is no reason to explore this further. Operations such as max or min can easily be performed using a number of match cycles and are considered trivial.

Summation Operators

Assuming that one value is resident in each PE, "cascaded addition" can be used to perform the addition. The operation at the j th stage of the computation is to shift the current result vector by $2^{(j-1)}$ processing elements and then to perform a component by component addition between the unshifted and shifted vectors. This results in the required sum being accumulated in a time

$$O(\log_2(N)).$$

This result assumes that there are at least N processors available to perform the calculation. For 32-bit data values and varying numbers of GLiTCH chips the results are as follows.

System size (chips)	Processing time (ms)	Communication time (ms)
4	0.04 (44%)	0.05 (56%)
16	0.05 (33%)	0.10 (67%)
64	0.06 (18%)	0.27 (82%)
512	0.075 (4%)	1.72 (96%)
1024	0.08 (2%)	3.37 (98%)

The results show that the ratio of processing time to data routing time is extremely poor which suggests that improvements can be made to the algorithm. The first algorithm investigated is similar to that used on DAP

Figure 3: Vision processing module

for summation [4]. The basic idea is to halve the length of each operand in the sum at each stage, dividing the computation between more and more PEs. This counteracts the reduction in the number of utilised PEs due to the cascaded addition. Finally, each PE contains only one bit of the partial results and the rest of the computation is achieved by parallel additions and carry propagations between PEs. The strategy for achieving this is important, since some methods result in the bits of the final value having to be collected from over the entire array. An efficient version of this method gives the following execution times.

System size (chips)	Processing time (ms)	Communication time (ms)
4	0.023 (61%)	0.015 (39%)
16	0.044 (64%)	0.016 (36%)
64	0.033 (61%)	0.021 (39%)
512	0.041 (38%)	0.067 (62%)
1024	0.044 (27%)	0.118 (73%)

Using similar techniques other routines are possible for summing a single bit column or several single bit columns. If several bit columns are to be summed independently then the computation can be spread out over the available PEs; an application of this is the histogram algorithm described in a later section.

Data Movement Routines

It has been mentioned that the time spent in routing data between PEs is an overhead that must be minimised. The theory of data routing put forward by Flanders [6] allows regular routing to be performed easily by defining the routing in terms of a mapping vector

which manipulates the addresses of the data items. The implementation of these changes to the mapping vector can be broken down into two operations; one which swaps two bits of the mapping vector, the other which inverts a single bit.

To implement these on GLITCH required only selected shifting operations to be performed. These have been written and are in the process of being optimised for the architecture.

LAPLACIAN OF GAUSSIANS FILTERING

The use of Laplacian of Gaussian filtering (LoG) for edge detection is known to require the use of large masks, i.e 31x31. Huertas and Medioni [5] have shown that the 2-D LoG mask could be decomposed into two 1-D masks with a corresponding reduction in the amount of computation. In addition the resulting masks are symmetric which allows a greater reduction in the computation.

Two methods for implementing LoG masks have been used. Both methods use the symmetry of the masks by multiplying the mask weight by each pixel and then shift-accumulating in both directions.

The first method loads the image into the array in row order and for the row mask the shifts required are $O(\text{mask_size}/2)$. For the column mask the shift lengths become $O(\text{patch_size} * \text{mask_size}/2)$.

In order to reduce the shifting required for larger masks another method was considered which again loaded the image in row order. However, before the column mask is applied the data reordering algorithms are applied to produce the data in column order. This allows relatively short shifts to be used for the column mask. The results

Figure 4: Execution times for 64 chips - solid line = shifting, dotted line = data rearrangement

Figure 5: Execution times for 16 chips - solid line = shifting, dotted line = data rearrangement

must then be reordered to row ordering (although this may be achieved automatically when the data is read out into the framestore). The results show (see figures 4 and 5) that the shifting method executes faster for all but the largest masks considered. This is because the time required for the row to column ordering transition remains a constant overhead irrespective of the mask size.

For both methods the zero crossings are obtained using the predicates suggested in [5], which only requires communications with the eight nearest neighbours. The predicate matching can be performed using multi-bit matches which each require a single clock cycle.

HISTOGRAMMING

Two methods of performing this operation have been considered. The first is to assign each bin of the histogram to a PE. The PEs are labelled with the pixel value of the bin which they accumulate. The image is held external to the array and each pixel is sequentially matched against the field containing the label of each bin. This match can be performed in a single clock cycle by the associative memory. The PE that has been marked then increments its histogram field. After $O(N^2)$ operations the histogram resides in the array, one bin per PE. This is not very efficient since incrementing the bins takes a large number of cycles compared to the matching and is not a parallel operation as it occurs in only one PE. A simple extension is to distribute each bin over a number of neighbouring PEs, the PEs associated with a particular bin being distinguished by marking them with different patterns in their subset CAM. In this way several pixels can be matched before the increment stage is performed. For a 16 chip system this requires $\approx 15\text{ms}$. This time is data dependent.

The second method uses one of the primitive summation operators and operates on a patch of the image at a time. The patch is held in the memory of the PEs, one pixel per PE. Every possible pixel value is matched in turn and the number of responses is summed each time. The summation operation used is one that produces the sums of several bit columns simultaneously by spreading the summation calculations across the array. In this way the parallelism at each stage is kept as high as possible. For a 1024 chip system this requires $\approx 5\text{ms}$ although a variant of the algorithm which holds several pixels per PE is possible which allows the chip requirement to be reduced.

FURTHER APPLICATIONS

GLITCH has been applied to a number of other image processing algorithms including the following;

- FFT [2],
- Convolution [1],
- Hough Transform [2],
- Voronoi diagrams,
- Image resampling [2],
- Image Generation [11].

CONCLUSIONS

The first GLITCH chips are expected during the Summer of 1989. Due to funding constraints these chips will only contain a small number of processing elements, but will allow the design to be verified before fabrication of the full device. It is envisaged that the full chips will be incorporated into a transputer controlled test bed

where their performance over a range of image processing tasks can be measured.

ACKNOWLEDGEMENTS

We wish to acknowledge the financial support of the Science and Engineering Research Council, BP International Limited and the MOD at RSRE for this work.

References

- [1] Duller A.W.G., Storer R., Thomson A.R. & Dagless E.L. "An Associative Processor Array for Image Processing" *Image and Vision Computing* Vol. 7 No. 2 May 1989, Butterworth Press.
- [2] Duller A.W.G., Storer R., Thomson A.R., Pout M.R. & Dagless E.L. "Image Processing Applications using an Associative Processor Array" To be presented at AVC89.
- [3] Hunt D.J. "The ICL DAP and its application to image processing" *Languages and Architectures for Image Processing* (Duff M.J.B., Levialdi S. eds), Academic Press, London, 1981.
- [4] Gostick R.W. "Software and algorithms for Distributed-Array Processors" *ICL Technical Journal*, May 1979.
- [5] Huertas A. & Medioni G. "Detection of intensity changes with sub-pixel accuracy using Laplacian-Gaussian masks" *IEEE Trans. PAMI-8* No. 5 September 1986.
- [6] Flanders P.M. "A unified approach to a class of data movements on an array processor" *IEEE Trans. Comp.* Vol. c-31 No. 9 September 1982.
- [7] Foster C.C. *Content Addressable Parallel Processors* Van Nostrand Reinhold, New York 1976.
- [8] Hockney R.W. and Jesshope C.R. *Parallel Computers* Adam Hilger Ltd., Bristol, 1981.
- [9] Reddaway S.F. "DAP - a distributed array processor" *1st Annual Symposium on Computer Architecture*, (IEEE/ACM), Florida, 1973.
- [10] Blair G.M. & Denyer P.B.
"Content addressability: an exercise in the semantic matching of hardware and software design" *IEE Computers and Digital Techniques* Vol. 136 Pt. E No. 1 pp 41-47 January 1989.
- [11] Storer R., Duller A.W.G. & Dagless E.L.
"Image Generation with an Associative Processor Array" *Proceedings of CG International 88*, published as *New Trends in Computer Graphics*, Ed. Magnenat-Thalmann N. & Thalmann D., (Springer-Verlag).