BENCHMARKING AN ASSOCIATIVE PROCESSOR ARRAY FOR VISION

M. Pout* and A.W.G. Duller **

The DARPA Image Understanding Benchmarks are used to measure the suitability of the GLiTCH associative processor architecture for a range of vision related tasks. In particular the tasks of edge detection, connected component labelling and the Hough transform are discussed in some detail. Results are then presented for two different array sizes for all of the image understanding tasks.

1. Introduction

One way of assessing the performance of a computer is by means of benchmark programs. However, traditional benchmarks are largely irrelevant for computer vision, since vision requires many different kinds of tasks, which require far more than just arithmetic. To overcome this problem a number of vision benchmarks have been suggested, such as the Abingdon Cross (Preston, 1986), the Tanque Verde Suite, the DARPA Image Understanding Benchmark (Rosenfeld, 1987) and the DARPA Integrated Image Understanding Benchmark (Weems et al, 1988). Various problems were encountered with the early benchmarks since they had no prescribed method and the results depended as much on the ingenuity of the programmer as the "power" of the architecture.

In order to assess the GLiTCH system, it was decided to implement the first set of DARPA benchmarks, since it appeared a somewhat smaller task, and also because the second set specified the use of IEEE standard 32-bit floating point arithmetic, which would prove difficult to implement in a 64-bit CAM version of GLiTCH.

2. Associative Processor Architecture

In this section a short overview of the GLiTCH architecture will be given. The basic GLiTCH architecture is targetted at solving low and medium level vision tasks with an emphasis on low cost high speed processing which was easily extendable.

The design uses the SIMD paradigm and consists of 64 identical processing elements (PE's) on each chip and uses Content Addressable Memory (CAM) as local

* Deptartment of Electrical and Electronic Engineering, University of Bristol, Queen's Building, University Walk, Bristol BS8 1TR, UK.

** School of Electronic Engineering Science, University of Wales, Bangor, Gwynedd LL57 1UT, UK.

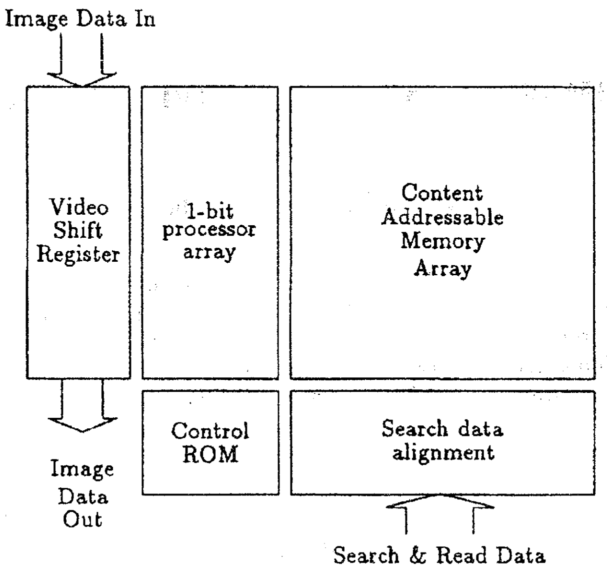

Fig. 1. Floorplan of a GLiTCH chip.

storage for each PE. A GLiTCH chip floorplan is shown in Figure 1. The basic parameters of the architecture are:

- 64 processing elements (PE) per chip
- 1-bit ALU in each PE
- 1-bit registers in the ALU
- 64 bits of data CAM per PE
- 4 bits of subset CAM per PE
- 1-D direct connectivity between PE's

Long distance communications supported by barrel shifter overlaid on the CAM array and an external routing network

The GLiTCH design is an SIMD array, combining the massive parallelism of a VLSI processor array with a content addressable memory capability. The array consists of a number of GLiTCH chips, a routing network, extra RAM for the processors and a microsequencer-based controller (Fig. 2). The GLiTCH program in the microcode store consists of; flow control operations obeyed by the sequencer; scalar operations passed to the transputer; and array operations applied to the GLiTCH chips. Associative operations allow the simultaneous comparison or writing of multiple data bits in all processors with a pattern supplied from the data store. Data may be read from a single processing element (PE) back to the transputer. Arithmetic and logic operations can be performed within each PE with status flags fed back to the sequencer on a reply bus. The data routing network shown can be used to pass each PE's "tag" bit between

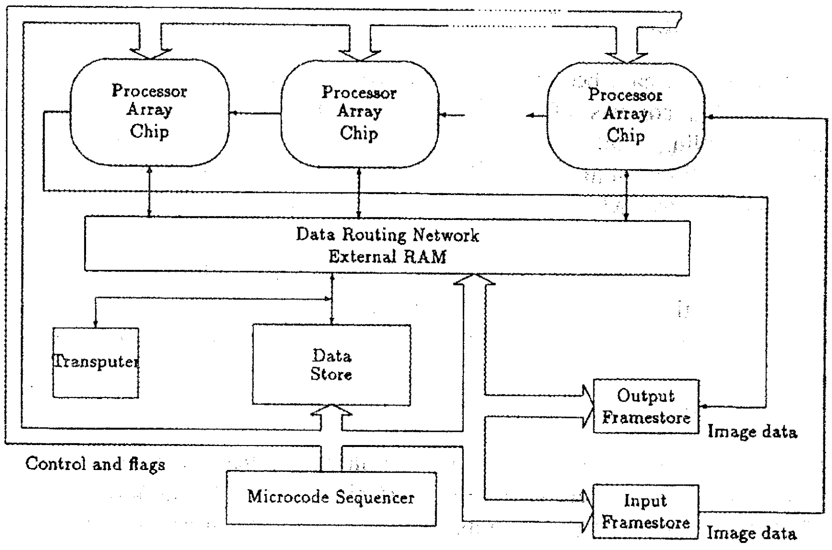

Fig. 2. A Multi-chip GLiTCH system.

GLiTCH chips. The RAM shown attached to each GLiTCH chip can be used to transfer data to/from a bitslice of the processors internal memory.

Each GLiTCH PE consists of a 64 bit data word, a 4 bit subset word, a single bit ALU and a few single bit flags. The data and subset words are stored in content addressable memory (CAM), which provides a bit parallel match with a search argument supplied on the data bus. The processor data CAM can also be read and written via the data bus; the subset bus allows only writing and matching. The matching of the subset data with the subset bus will set the match subset (MS) register. This may be used to locally control subsequent global operations. Each PE can also locally control the writing of global data, allowing the disabling or inversion of data and subset patterns. Two bits of data can be fetched from the CAM simultaneously for arithmetic and logic operations, the results of which are stored in the carry and tag registers. The tag and carry registers are wire-ORed to reply lines to the controller for conditional branching. A tag shift operation allows the tag registers to be shifted one PE up or down the array per clock cycle. Long distance tag shifts are effected using the data routing network.

As mentioned earlier the GLiTCH chip is initially envisaged as part of a multi-chip system which has a configuration such as that shown in Figure 2. A transputer is shown as part of the configuration both for its scalar processing capability and for compatibility with other parts of a hierarchical vision system that is being designed (Pout et al, 1991).

4. Image Understanding Benchmark

The benchmarks that have been used are the first set of DARPA Image Understanding Benchmarks. This consists of the following tasks: Edge detection (incorporating convolution, finding and outputting sequences of zero-crossings), Connected Component Labelling, Hough Transform, Geometrical Constructions (Convex Hull, Voronoi Diagram and Minimal Spanning Tree), Visibility, Graph Matching and Minimum Cost Path.

In order to give an idea of the types of methods required for such a processor we consider only the first three task in the benchmark. Results for all of the tasks are presented at the end of the paper but without algorithmic details for the other tasks.

5. Edge Detection

The first of the Benchmark tasks is a convolution of the image with a high-frequency filter, and the subsequent identification of edge pixels. This task is representative of a wide range of simple filtering algorithms. The benchmark requires the convolution of a 512 by 512 8-bit image with a sampled 11 by 11 Laplacian operator as described in (Haralick, 1984).

Convolution is a task in which each output pixel depends on a number of input pixels. This dependency is both regular and local: each output pixel is calculated as the weighted sum of all the pixels within a small radius around the corresponding input pixel. Normally the number of PEs available is less than the number of pixels within the image, and so the image is processed in several patches. The dependency of each pixel on a local neighbourhood means that the correct value cannot be calculated for pixels on the border of a patch; some of their input pixels are in a different patch.

The most efficient method depends on the type of mask required. Three different cases have been investigated:

- Arbitrary mask
- Repeated weights
- Symmetric mask

By using accumulation schemes specific to the mask type, fast algorithms are possible. Examples of these schemes are given in Figure 3. The most efficient of these for the benchmark convolution is a row accumulation method which makes use of the two degrees of symmetry in the operator; this is over four times faster than the first method shown.

Another convolution method exists, whereby the patches are 1 pixel high by the image width and data is kept from one patch to the next. This ``scan line" method requires as many registers as there are rows in the mask, and the product of the current scanline with each row of the operator is added into one of the registers. For 3 by 3 pixel operators, these registers can be stored in the CAM of a single PE. For larger

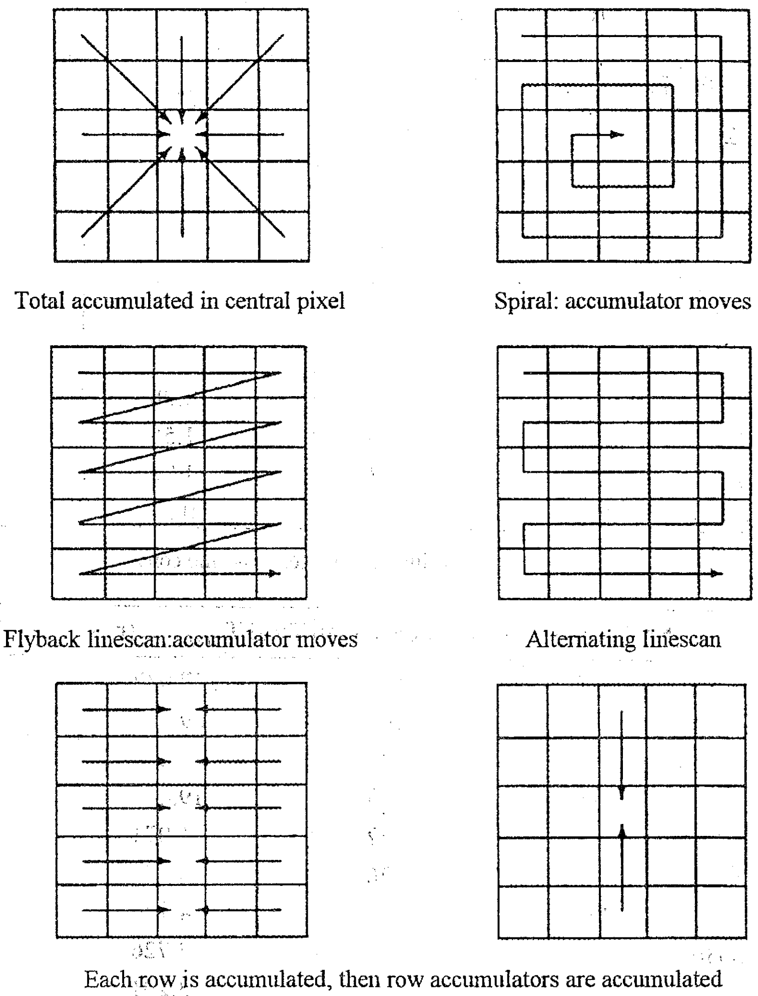
Total accumulated in central pixel
Spiral: accumulator moves
Flyback linescan:accumulator moves
Alternating linescan
Each row is accumulated, then row accumulators are accumulated

Fig. 3. Different Convolution Methods.

operators, registers must be stored in RAM. This method can make use of the symmetry in the operator.

Where there are more PEs than the number of pixels in a scanline, the extra PEs can either be used (instead of RAM) to store some registers, or to simulate other scan-line array processors working on other parts of the image. The former approach will increase communication time by separating the pixels, but not add significantly to the parallelism. The latter method will not affect the time per scanline, but will reduce the number of repetitions required. Results show that the scan line method is the more efficient for small array sizes, and the patch processing for large array sizes.

Table 1. Total convolution times for different system sizes

No. PEs	Patch Width	No. Patches	Total $T_{process}$ (ms)	Total T_{load} (ms)
256	16	7056	1629.23	90.317
512	22	1638	404.422	41.933
1024	31	528	143.035	27.034
2048	44	210	58.569	21.504
4096	56	88	24.543	18.022
8192	73	40	12.116	16.384
16384	111	20	6.378	16.384
32768	178	9	3.086	14.746
65536	128	5	1.555	16.384
131072	256	3	1.221	19.661
262144	512	1	0.599	13.107

Table 2 . Convolution times for scan line convolution.

No. PEs	No. Scanlines	Scanlines/Sections	Total $T_{process}$ (ms)	Total T_{load} (ms)
512	512	512	137.037	13.107
1024	522	261	69.857	13.363
2048	542	136	36.267	13.875
4096	582	73	19.472	14.899
8192	662	42	11.074	16.947
16384	822	26	6.875	21.043
32768	1142	18	4.776	29.235
65536	1782	14	3.726	45.619
131072	3062	12	3.201	78.387
262144	5622	11	2.939	143.923

6. Zero Crossing Detection and Extraction

The sequence of zero crossings have to be produced as a list. The zero crossings of the Laplacian of Gaussian operator are defined as those pixels at which the output is positive, but which have neighbours where the output is negative. This can be performed by fetching the sign bits from the neighbours and comparing them with the sign bit of the central pixel.

Having identified the zero-crossings (ZCs), it is now required that ordered sequences specifying the border of each region with positive LoG output are produced.

With one PE per pixel, this can be performed using either a sequential or a parallel method. If the array has fewer PEs, this introduces a problem where a sequence crosses a patch boundary, and this is dealt with separately.

The sequential method is as follows. An initial "seed" ZC can be selected using a "first" operation, and from this one pixel, the border must be followed pixel by pixel. The address of the initial pixel is read out as the first ZC of as sequence. The next ZC can be selected either by shifting a tag, or by broadcasting the next address. When the sequence returns to the original ZC address, the border of the first object is complete, and a seed point on the border of another region must be selected using a first operation. When there are no more regions is the process complete. If each PE stores a bit indicating which of its neighbours are ZCs, the controller can calculate the next address by examining the pattern of neighbours.

The parallel method is best performed using a connected component labelling algorithm, since the operations are effectively the same. The position within the sequence can be produced by marking the first pixel in each sequence, and passing a tag around the sequence, writing its position.

7. Connected Component Labelling

This task takes as its input a 512 x 512 binary image, and requires as output an image which has '0's where the input image has '0's, but replaces the '1's with an integer, such that all pixels which are part of the same contiguous region have the same unique number.

The labelling can be either component sequential or component parallel.

Component sequential. An initial "seed' pixel in one component is identified, and adjacent pixels are identified using either a breadth- or depth-first method. A breadth-first method, as described by Storer (Duller, Storer et al 1989) attempts to grow one pixel in each direction at each step. After each step, the new pixels are identified, and if none have been found, the component is labelled. A new seed pixel in another component is then selected. A depth-first method extends the labelled region as far as possible in each of the directions. Although this avoids continually trying to extend the region beyond a boundary, a test is required to decide when a boundary has been reached.

The only parallelism in these methods is that a component can be extended from all the pixels along one edge simultaneously.

Component parallel. In this method, each set pixel is given a unique label, e.g it's address (Rosenfeld and Kak, 1982), and a minimum operation is repeatedly performed between adjacent pixels until all the pixels in each component have the minimum label in that component.

7.1. Comparison of Methods

The time of all the above methods is data-dependent; a rectangle will be labelled more quickly than a snake-like object with the same area. The component parallel methods depend on the time to label the largest component only. The sequential methods have the advantage of not being too inefficient with large components, having simpler code, using less memory and producing shorter labels.

7.2. Patch Consistency

When an image is labelled in patches, each component within a patch is given a unique label, but those components crossing patch boundaries will be assigned different labels in different patches. It is therefore necessary to also perform a patch consistency phase, which re-labels such components correctly. The amount of work required depends on the data, and also on the choice of patch shape. Although a square patch gives the minimum total length of boundary (Duller, Morgan and Storer, 1987), image height patches make the consistency problem one- rather than two-dimensional.

The patch consistency problem can be approached using a global or a patchwise approach. In a global approach to patch consistency, the patch boundaries are retained as each patch is labelled. Pixels on the boundaries of adjacent patches can be used to produce a list of component label equivalence classes. Each image patch is then reloaded, and all components with inconsistent labels are changed. If the boundaries are extracted as each patch is labelled, the computation of the equivalence classes can be performed by the host in parallel with the labelling of subsequent patches.

A patchwise approach to this problem was taken by Brandao (Brandaô, Storer and Dagless, 1990) for the component labelling part of a Number Plate Recognition program on GLiTCH. All the labelled patches are placed in RAM, and subsequently pairs are fetched and re-labelled. This will deal with most components, but some components which appear unconnected prove to be joined in a later patch; thus the process may need to be repeated in the reverse direction. This method can be improved upon by keeping the previous patch in memory, and performing the consistency check before the first patch is stored. This will correctly label most components, and those patches which do require any relabelling can be selectively processed.

8. Hough Transform

The Hough Transform is a technique for detecting straight lines in noisy images. It performs a transformation on the image space to produce an output parameter space in which collinearities will be easily identified. Each edge pixel in the input image casts a vote in the parameter space for those lines on which it lies, and hence straight lines appear as peaks in the parameter space.

The most popular parameters for straight line detection are the polar coordinates (ρ,θ). ρ is the length of the perpendicular from the origin to the line, and θ is the angle which this makes with the x-axis.

$$\rho = x \cos\theta + y \sin\theta \qquad (1)$$

In the Image Understanding Benchmark definition, the input is a 512 x 512 binary image (which has already been processed using an edge operator), and the output array covers the ranges $\theta = [0\ldots180]$, and $\rho = [0\ldots511]$. Each pixel calculates a ρ value for each value of θ, and the total number of votes for each (ρ,θ) pair is stored in the output array. The following sections describe three approaches to the problem - θ sequential, pixel sequential and scan line sequential.

8.1. θ Sequential

If the array has sufficient PEs to provide one per pixel, each pixel can store an input pixel, and can also store an output accumulator. This approach to the Hough Transform is described in (Pout, 1988). For each value of θ, the values *sin* θ and *cos* θ are broadcast, and each set pixel calculates its corresponding ρ. These ρ values indicate which accumulators should be incremented in the current θ column. The process of collating these votes into accumulator bins is similar to a histogram operation, and can be performed using either a ρ-sequential method of counting responders, or a more parallel approach in which each set of 512 PEs forms a local histogram of its own ρ votes, and a cascaded addition performed to sum the local votes.

8.2. Pixel Sequential

The pixel sequential method provides one PE per value of θ (i.e. 180 PEs) and the accumulator bins associated with that θ are stored in the RAM associated with that PE. The (x, y) coordinates of each set pixel are broadcast by the controller, and each PE calculates a ρ value from Equation (1). The ρ values produced are used to calculate the RAM addresses of the bins to be incremented. These bins are loaded into GLiTCH, incremented, then stored back in RAM.

The host transputer passes the coordinates of set pixels to the controller. This should not amount to a significant overhead, since the bulk of the testing can take place in parallel with the processing of earlier pixels. Initially, each PE is loaded with the values of *sin* θ and *cos* θ for its θ. A match operation on the MSBs is used to discard ρ values outside the range [0...512].

The process of incrementing the accumulator bins stored in RAM is slow, since slices of RAM cannot be locally addressed. Thus, each different value of ρ requires a separate load-and-save cycle. Also, since the altered values need to be merged with the unaltered ones, it is more efficient to load, conditionally increment and save all the bins for each produced ρ value sequentially. For an average pixel on the image, only about

120 unique ρ values will be produced due to out of range values and duplication. Thus the required values are fetched selectively, using a first operation to select PE with valid ρ values, which are read and used to address the RAM to fetch the correct accumulator.

8.2.1 Multiple PEs per θ

The most direct way of using more than 180 PEs is to provide many PEs for each θ, each of which can calculate a different ρ values. There will be a slight increase in the time to calculate the ρ values, since Vector-Vector rather than Scalar-Vector arithmetic must be used.

8.2.2 Using Less RAM

The above method requires a large amount of RAM; this can be reduced if the 512 ρ bins belonging to each θ are divided equally amongst a group of M PEs to be assigned to each θ. Now each PE calculates the same ρ value, but instead of checking against the maximum and minimum values, ρ is compared with the range appropriate to each PE.

8.2.3 Multiple Buckets per PE

The method above uses a group of PEs for each θ, but the memory map of the CAM for the above algorithm is largely empty. This space can be used to store the ρ values from other few pixels, so that when an increment is performed, several votes are accumulated at once. The term "bucket" is used to denote a register in which a ρ value can be stored.

The above method has the drawback that the same calculation is carried out in the M PEs of the group, only one of which has a valid vote. If only the first PE in the group is used to perform the calculation, the remainder have more space for buckets. There will be a slight overhead in passing each ρ value to the other PEs in the group, but this is more than outweighed by the greater number of votes included in each accumulator update. The update now requires all the buckets to be compared with the broadcast ρ, and the number of matches accumulated in a register which is added to the accumulator bin. The value of Q (the number of bins needing updating) increases up to the maximum possible (i.e. 512/M) as the votes from several pixels are combined.

8.2.4 Bins in CAM

If the array has 32768 PEs, the RAM can be dispensed with completely, since all the bins can be stored in CAM. If the array is more than 32768 PEs, then the process can be replicated, with different sections of the array being given different pixel values. A small overhead is introduced in the calculation time, since Scalar-Vector multiplication can no longer be used, and also to sum the values of the bins across the array, but apart from this, the algorithm will show a linear speed-up.

Times for all the pixel sequential methods are shown in Figure 4.

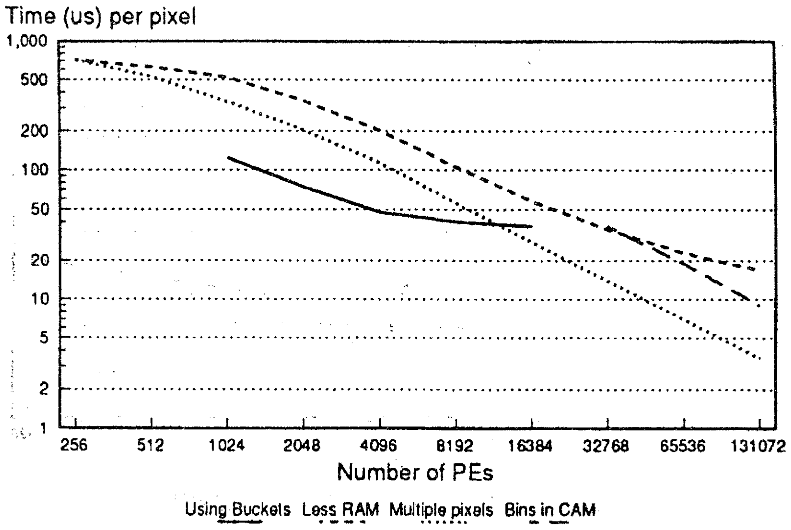

Fig. 4. Times for pixel-sequential Hough transforms.

8.3. Scan Line HT

In the projection-based algorithm (Sanz and Dinstein, 1987) described by (Fisher and Highnam, 1989) for their Scan Line Array Processor (SLAP) (Fisher, 1986), (Fisher Highnam and Rockoff, 1988), the Hough space accumulators (bins) travel along the path of pixellated lines, accumulating the votes from each set pixel as it is passed through.

A table is created which indicates those accumulator bins which are to be shifted at each scan line. Lines which intersect with the bottom of the image will have their bins left in the array when the process is complete. Lines which pass off the edge of the image are extracted from the array before they are shifted off the edge of the array. In order to minimise the shifting time, only lines which pass through one pixel per scan line (i.e. within 45° of vertical) are processed, the others being handled by a second pass on a diagonal reflection of the image.

The Scan Line method can be run on a GLiTCH system as described in (Duller et al, 1989), but the limited amount of CAM per PE means that many PEs must be used for each pixel, or several passes are required for different sets of θ values. The time taken by this algorithm is shown in Figure 5.

8.3.1 Double Scan Line

The above algorithm requires that the data store is sufficiently large to store the Hough space array, or alternatively that values be stored in the transputer. The alternative

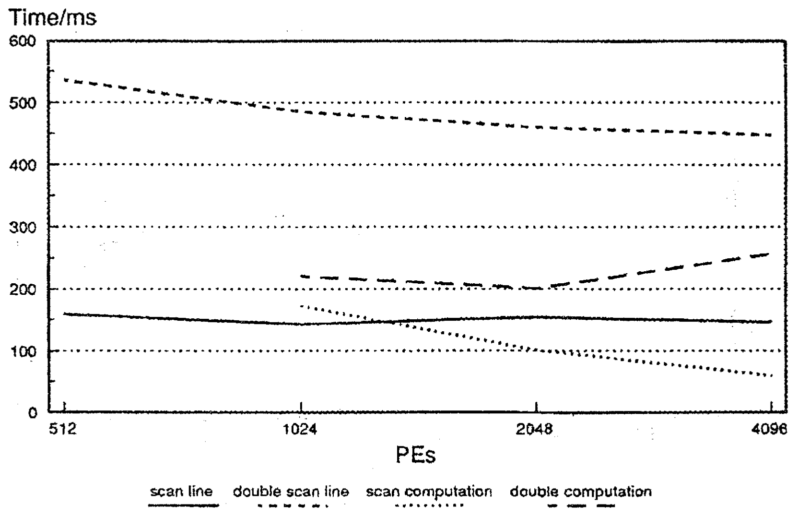

Fig. 5. Scan line transform times.

place for storing data, in RAM is not practical since all the bins which intersect the edge of the image are produced only at the end PEs of the array.

This inefficiency can be avoided if two scan lines' worth of processors are used: the scan line is loaded, filling only the first half of the array. Lines which intersect the edge of the image do not now "fall off" the edge of the array, but instead are projected down through the empty half of the array to reach the final scan line. The array is now configured as a ring rather than a chain, so that data wraps around. All the results for each θ value are thus available, 1 per PE, in order at the end of each pass.

The times for the double scan line method are also shown in Figure 5. The vast improvement in DSR loading time due to the use of RAM is clear, and other algorithmic improvements lead to a more nearly linear computation time. As the system size doubles however, the length of the DSR also doubles, and thus the idle time at the start of the first patch increases. This is unfortunate, since the amount of data loaded is constant, it is only the number of blank spaces interleaved with the data which increases. The alternative (and often faster) method to loading data interspersed with spaces is to load the data into the first few PEs, and subsequently use low data manipulation routines to organise the data as required.

8.3.2 Resampling Hough Space

The results produced by the scan line Hough Transform is in the form of a rectangular array of accumulators indexed by ρ and θ. The θ values can be chosen to be integers, but the ρ index goes in steps of *cos* θ rather than integers (which is required by the benchmark specification). Each column of the array must therefore be resampled using the method described by Storer (Duller et al, 1989).

9. Overall Results

A full set of results are not yet available for the IUB. The intention is to define two types of system, a small and large version. All of the benchmarks will then be run on these two systems which will allow sensible comparison with other such architectures. In addition the comparison of the two systems will give an indication of the efficiency of the algorithms used. It has already been found that for many of the benchmarks the optimum algorithm depends strongly on the number of PE's available. Early results are given below with others for comparison.

Architecture	Task/ time (ms)								
	1a-1b	1c	2	3	4a	4b	4c	5	7
Butterfly (16 nodes)	18300	-	-	45200	190	-	-	67000	-
Butterfly (100+ nodes)	2900	8200	7200	7400	-	-	-	4150	-
CM-1	3	100	400	700	200	2000	2200	1000	50
NON-VON	2	-	1000	400	40	-	40	100	40
Cube (256 Nodes)	100	-	140	1800	2.4	-	-	-	10
Mosaic (16K nodes)	2.5	1	6	10	3.6	-	-	-	1
MULTIMAX (20 nodes)	46000	6900	22700	244000	1800	30000	8700	91400	180
IUA	0.02	0.2	0.005	27	7	50	11	20	0.7
Warp	16	690	750	600	3	11	43	40	25
HBA (16 nodes)	3200	100	170	270	550	-	-	-	-
HBA (100+ nodes)	600	-	370	30	940	-	-	-	-
GLiTCH (4K PE's)	27	2.49	15?	205	2.52	1201	14	457	373
GLiTCH (256K PE's)	0.60	2.11	3.36	14.3	2.52	30	14	13	17?

10. Conclusions

As can be seen from the overall results, the GLiTCH architecture performs well on most of the benchmark tasks, especially when similar "size" systems are compared. It has been shown that a realistic system, the 4k PE configuration, is capable of performing all of the tasks in the benchmark.

The increasing integration possible with VLSI will make larger systems than that proposed readily accessible and algorithms such as those presented here will allow such architectures to be used efficiently.

References

Brandaõ Â., Storer R., and Dagless E.L. (1990): *Vehicle license plate recognition with an associative processor array.*- Internal Report Dept. of Electrical Engineering, University of Bristol.

Duller A.W.G., Morgan A.D. and Storer R. (1987): *Associative processor arrays: Simulation and performance estimates for image processing.*- Proc. of Alvey Vision Conf. 87, September, pp.139-145.

Duller A.W.G., Storer R.H., Thomson A.R. and Dagless E.L. (1989): *An associative processor array for image processing.*- Image and Vision Computing, v.7, No.2, May.

Duller A.W.G., Storer R.H., Thomson A.R., Pout M.R. and Dagless E.L. (1989): *Image processing applications using an associative processor array.*- Proc. of the 5th Alvey Vision Conf., September.

Fisher **A.L. and Highnam P.T.** (1989): *Computing the Hough transform on a scan line array processor.*- IEEE Trans. on Pattern Analysis and Machine Intelligence, v.11, No.3, pp.262-265.

Fisher A.L. (1986): *Scan line array processors for image computation.*- Proc. 13th Annual Int. Symp. on Computer Architecture, pp.338-345.

Fisher A.L., Highnam P.T. and Rockoff T. (1988): *Scan line array processors.*- Ambler T., Agrawal P. and Moore W. (Eds.), Hardware Accelerators for Electrical CAD, Adam Hilger, pp.312-324.

Haralick R.M. (1984): *Digital step edges from zero crossings of second directional derivatives.*- IEEE Trans. on Pattern Analysis and Machine Intelligence, v.6, No.1, January, pp.58-68.

Pout M.R. (1988): *Comparison of the content addressable parallel processors, SCAPE and GLiTCH based on the Hough transform.*- Undergraduate Thesis, Dept. of Electrical Engineering, University of Bristol.

Pout M.R., Storer R.H., Thomson A.R., Dagless E.L. and Duller A.W.G. (1991): *An associative processor array as part of a heterogeneous vision architecture.*- V. Milutinovic and B.D. Shriver (Eds.), Proc. of the 24th Annual Int. Conf. on System Sciences, IEEE Computer Society Press, v.1, pp. 260-268.

Preston K. (1986): *Benchmark results: The Abingdon cross.*- UhrL. et al (Ed.), Evaluation of Multicomputers for Image Processing, pp. 23-54, Academic Press.

Rosenfeld A. (1987): *A report on the DARPA image understanding architectures workshop.*- Proc. of the DARPA Image Understanding Workshop, pp.1:298-302.

Rosenfeld A. and Kak A.C. (1982): *Digital Picture Processing (2nd Edition).*- Academic Press.

Sanz J.L.C. and Dinstein I. (1987): *Projection-based geometrical feature extraction for computer vision: Algorithms in pipeline architectures.*- IEEE Trans. on Pattern Analysis and Machine Intelligence, v.9, No.1, pp.160-168.

Weems C., Riseman E. and Hanson R. (1988): *An integrated image understanding benchmark: Recognition of a 2 1/2d mobile.*-Proc. Image Understanding Workshop, v.1, pp.111-126.