Image Processing Applications using an Associative Processor Array

A.W.G. Duller R. Storer A.R. Thomson M.R. Pout E.L. Dagless

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

GLiTCH is a VLSI associative processor array chip designed at Bristol University. This paper summarises the results of three studies of real-time image processing applications: image resampling, fast Fourier Transforms and Hough transforms.

The design of the GLiTCH chip and its use in image processing systems is described by us in a companion paper [1]. Several applications of such a system in real-time image processing have been studied, which show that GLiTCH chips can be used effectively in a programmable image processing system.

FAST FOURIER TRANSFORM

It was decided to implement the FFT on GLiTCH because it is both a useful routine for image processing and also very computationally demanding. When implemented on an array processor, the FFT serves as a good test of the mathematic and data movement capabilites of the machine. Given that GLiTCH is restricted on memory size, and is also only a pseudo-2D array, it was felt that the 2D FFT would serve as a good all-round test.

FFT implementations vary considerably from machine to machine, and it is clear that the algorithm used must be adapted to closely match the target machine's architecture if maximum performance is to be achieved. For bit-serial machines in particular, the time taken to calculate a FFT will also be very dependent on the precision required.

A FFT program has been written and simulated for the GLiTCH system and even with only simple fixed-point mathematics and unoptimised code, it has been shown that a LARGE number of GLiTCH chips could achieve real-time 2D FFTs (and inverse FFTs) on 256×256 pixel 8-bit images. Comparisons between a simple and a more sophisticated architecture have shown that very significant speed increases can be obtained in large systems by increasing the memory available to each processing element. This is done by adding external storage (possibly in the routing chips) and swapping data with the high-speed on-chip memory [1].

Basic Algorithm

The algorithm used on GLiTCH is based on the one for the DAP, described by Davies in [2][3], which works as follows.

The DAP [4] is a mesh-connected 2D bit-serial array processor with a large amount of storage per processing element (PE). Davies' algorithm assumes one data point per PE, and only linear connectivity. Starting with a set of N points, $\log_2 N$ iterations of the algorithm will be required:

1. Split each set into two halves
2. Copy the data between these two halves
3. Multiply by a value dependent on the set number
4. Add the result to the data present at the beginning

This is remarkably simple to implement, having nothing more complicated than addition, subtraction and multiplication. The sine and cosine multipliers are precalculated and broadcast to the individual PEs requiring them. When all have been stored, the calculation takes place. The result is produced in bit-reversed order.

Data routing takes about 10% of the total time, or 20% if the bit-reversed ordering is corrected [5]. Roughly 18-20% [2] of the time is used setting up the sine / cosine values. The algorithm operates in $O(\log_2 N)$ time.

GLiTCH Algorithm

The basic GLiTCH architecture imposes the following constraints not encountered on the DAP:

- Pseudo 2D connectivity
- 64 bits of memory per PE

The first of these has only a small effect as the algorithm does not require (though can benefit from) 2D connectivity, but the second is rather more serious, especially if a very high precision for the FFT is required.

GLiTCH has been designed to allow 2 or even 3 PEs to work together efficiently to perform complicated calculations, make use of each other's memory etc, and this is clearly a possible approach to tackling the FFT mathematics required. However, combining PEs in this way effectively wastes $\frac{1}{2}$ or $\frac{2}{3}$ of the processing power of the system, as only one PE in the group can be active at a time. Hence it is very important, if at all possible, to fit all the required data into the memory space of a single PE. In fact, if it appears that 2 PEs will be needed, it will be worth tring to find an alternative algorithm which is up to twice as slow providing it requires sufficiently little memory to fit in just one PE.

A fixed point version has been implemented using 24/28 bits to store each number. This allows an almost-perfect 2D FFT/inverse-FFT to be calculated, but typicaly 1 or 2 pixels will end up with an intensity one grey-level different to the original. This is because the 28 bits are just insufficient to cope with the dynamic range of values found in a 2D FFT. Preliminary tests on floating point indicate that it will be both approximately 20 – 25% faster and also overcome problems of accuracy.

Results

The following results have been obtained for GLiTCH arrays of varying sizes, all performing a 2D FFT followed by an inverse-2D FFT on 256 by 256 pixel 8-bit images using fixed-point arithmetic.

System size	Total time (ms)	
(chips)	Simple system	Sophisticated system
8	630	-
16	323	305
32	169	154
64	93	78
128	55	40.6
256	38.6	22.1
512	34	14
1024	39	10.8

As can be seen, the times for the smaller systems are very similar, but as the systems become larger the more sophisticated architecture starts to show its benefits. This is primarily because of the large amount of time required to load data into the simpler machine which is unnecessary in the more advanced one. This clearly makes real-time 2D FFTs a possibility, though at some considerable cost! Further, the extra memory available in the more sophisticated architecture allows for greater precision to be achieved if necessary.

These results (in conjuction with the other work in this paper) provide a strong argument for building a reasonably large amount of external memory into the system, but keeping the high-speed on-chip memory as well.

SCAN LINE PROCESSING

The Scan Line Array Processor (SLAP[6]) is a SIMD linear array of processing elements. It provides one PE for each pixel in a scan line of the image, and this leads to somewhat unusual image-processing algorithms. Although GLiTCH has not been specifically designed to implement scan line algorithms, comparing the architecture of SLAP (see Figure 1) with a GLiTCH system, the only obvious addition is the 'End' registers. These are used to provide or retrieve data when a shift between PEs is performed. A GLiTCH system can simulate these by reading out and writing data before performing *cyclic* shifts.

The other major difference is in the complexity of PEs. SLAP has bit-parallel arithmetic, and considerably more memory per PE. Fisher and Highnam [7] suggest a VLSI implementation with 4 PEs per chip, and a cycle time of 125ns.

Hough Transform

The Hough transform [8] is a well known image processing algorithm which is used for extracting lines from images. Straight lines are characterised by two parameters (e.g. ρ, θ, where $\rho = x \cos \theta + y \sin \theta$)

Edges detected using a simple operator, and each edge votes for the parameters (ρ, θ) of the lines on which it lies. The transformation to parameter space, in which are votes passed to the correct parameter plane accumulator is expensive to implement on SIMD machines due to the chaotic mapping of data.

The projection-based algorithm [9] described by Fisher and Highnam [7] is radically different; rather than passing the votes to a static array of accumulators (bins), the bins are passed along the path of pixellated lines, accumulating votes as they progress.

The bin for a given near-vertical line will either stay in the same PE, or pass one PE to the left (or right), between one scan line and the next. All the lines at one angle are calculated simultaneously. The number of angles which can be calculated in a given pass is dependent on the number of bins which will fit in a PEs memory. If a line intersects the edge of the image, it cannot accumulate further votes, and hence it may be read out. The accumulator is then zeroed, to simulate an empty bin being shifted into the far side of the system. Once the entire image has been scanned, a set of bins corresponding to those lines which intersect with the bottom of the image will remain in the PEs, and these too are read out and placed into the accumulator array.

It is important to note that lines which are nearer to the horizontal than the vertical can pass through more than one pixel in a given scan line. A SIMD machine cannot efficiently implement shifts of varying distances, and so it is usually more efficient to transpose the image perform

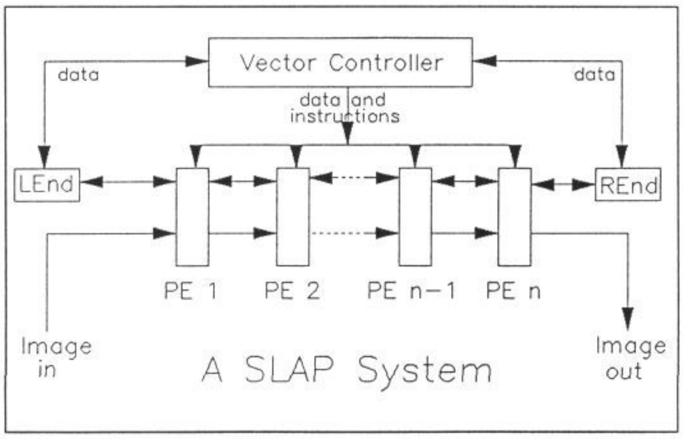

Figure 1: Scan Line Array Processor System Architecture

a second pass to detect near-horizontal lines. This transposition can implicitly performed by a GLiTCH framestore.

The timings below assume that the input is a binary, edge-detected 256 square image (GLiTCH can perform this calculation in under 4ms). The minimum system size is the width of a scan line (256 PEs = 4 chips), and bins must be 8 bits long, implying that only 8 may be stored per PE. The transform was of 3 degree resolution; more accurate transforms will take proportionately longer. In a larger system, several PEs can be assigned to each pixel, and more values of θ accumulated at once. Assigning many PEs per pixel has the side effect of increasing the shifing time, and the image loading time:

Chips	Total Time	Load Time	Process Time
4	45.45	26.21	41.50
8	37.08	26.21	22.77
16	32.94	26.21	13.82

Comments

The overall time shows a relatively low speed up, but the actual processing time is more nearly linear. The overhead due to increased DSR length when more PEs are added would be decreased either if some other processing were performed (e.g. the edge-detection) or a faster way of loading the data were available. Considering that the same data is loaded for each pass, if the system were enhanced with two hundred bits of RAM per PE [1], data could be loaded for subsequent passes in a fraction of the time. (GLiTCH can transpose a binary image in RAM in 0.3ms). Thus a 16 chip system can transform a raw image in *under 20ms*. This compares very favourably with Fisher [7] and Highnam's prediction of SLAP taking 14-20ms for a 3 degree transform on a 512 square image, particularly when it is remembered that the SLAP system would needs 128 chips. GLiTCH also has the advantage of flexibility: even a small system is capable of performing a Hough Transform to an arbitrary degree of accuracy if given enough time.

The scan line Hough Transform is an interesting and efficient algorithm. Although it has only been applied to straight lines, it can be applied to a wide range of non-parametric curves. One important consideration is the format of the output. The scan line algorithm produces the θ coordinate explicitly, but the resolution of the other (ρ) coordinate is not constant across the θ range.

IMAGE RESAMPLING

A digitised image consists of an array of discrete samples of the continuous function that is the real image. Image resampling alters the sampling rate of a digitised image without reference to the original scene which produced it. It is used to expand an image so that a part of it can be examined more easily, to replace erroneous or lost samples in a digitised image, to register two comparative images of the same scene or to correct geometric deformations caused by sensor attitude or motion.

If a new sample is required at a point between the existing samples one of three interploation methods is commonly used to obtain it: nearest neighbour, bi-linear, and bi-cubic [10]. All of these are approximations to convolution with the sinc function, $sinc(t) = \sin(\pi t)/\pi t$, which would provide a perfect, smoothed interpolation if it could be realised on a digital computer.

Making Use of Parallelism

A typical bi-cubic convolution requires about 15 multiplications and 45 additions for each pixel processed, about 10^6 multiplications and 3×10^6 additions for a 256×256 pixel image. As the operations on each pixel are identical and do not depend on the result from any other pixel, the algorithm is ideal for processing on a SIMD array.

Warpenburg [11] describes a technique for image resampling where the original image is divided equally between the processing elements (PEs) of a generalized SIMD array. Each PE in the array executes the resampling algorithm much as a uniprocessor would. Thus, a performance improvement of factor N is claimed for an array of N processing elements.

This result overlooks the fact that as N increases, the amount of the input image held by each PE decreases and there will be more inter-PE communication needed to process the pixels around the edge of each PE's region of the image. Inter-PE communication is much slower than access to data within a PEs memory so that, in a fine-grain SIMD array (like GLiTCH) which assigns one PE for each pixel in the input image, the inter-PE communication becomes the critical performance factor.

Resampling Using GLiTCH

The current work investigates the use of a GLiTCH array for resampling where each PE is used to calculate one pixel in the output image. If the number of PEs available is smaller than the output image size, several passes of the interpolation algorithm are used to generate successive patches of the output image. This scheme makes maximum use of the available parallelism since all the processors have identical work to do. It also ensures that the output image is available in a regular form, one pixel per PE, and so can be easily read from the array. However, the data reordering to align the existing samples to the grid of the required samples becomes the most costly part of the algorithm.

If the new sampling intervals are ΔX_r and ΔY_r in the x and y directions, an existing sample at x, y in an input image sampled at ΔX and ΔY is moved to location x_r, y_r in the output image where:

$$x_r = x\frac{\Delta X}{\Delta X_r} \qquad y_r = y\frac{\Delta Y}{\Delta Y_r}$$

Each PE holds an input pixel and an address x_r, y_r, each calculates the values $x_r - x$ and $y_r - y$ to find the x and y distance from the input pixel it should hold. Where the ratio between sample intervals is a power of two, Flanders' "musical bits" algorithms [12] provide an elegant means of reordering array data expressed as the exchange of two address bits. Otherwise all the input

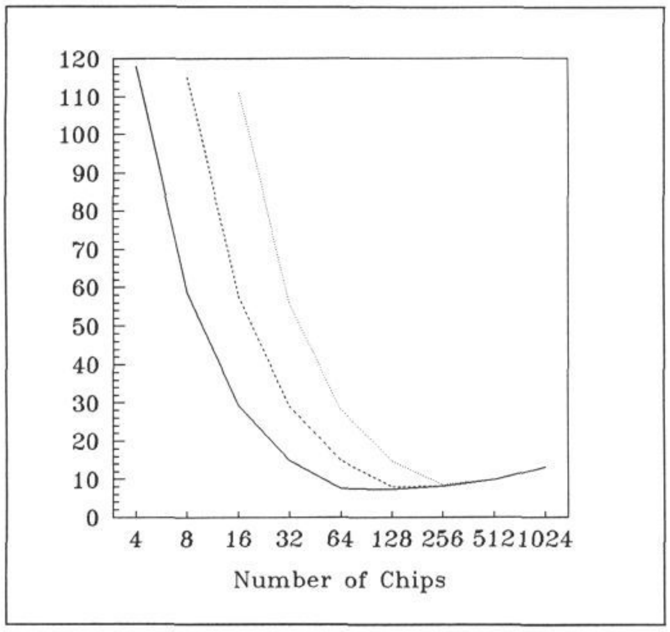

Figure 2: Maximum resample times (ms) for 256 × 256 image, solid line - nearest neighbour, dashed line - bilinear and dotted line - bicubic interpolation

pixels are passed from one PE to another until each has collected the ones it needs for the interpolation.

Results

The maximum resampling times required for the three interpolation techniques are shown in figure 2. The times are inversely proportional to the number of PEs except with large arrays of chips where the time taken to load and dump the image dominates the performance.

For nearest neighbour interpolation the required sample value at each point is taken to be that at the nearest integer values to x_r and y_r. The resulting image is acceptable where the resampling is intended to enlarge the original pixels, otherwise the magnified interference between the spatial frequencies of the image and the original sampling frequency makes it unusable.

Using two passes through the image, 16 GLiTCH chips (1024 PEs) are able to resample a 256 × 256, 8-bit image in less than 30 ms, depending on the ratios $\Delta X/\Delta X_r$ and $\Delta Y/\Delta Y_r$. If these ratios are near to one, the time falls to about 128 μs.

For bi-linear interpolation each PE must collect the four existing samples surrounding x_r and y_r. 16 chips require at most 58 ms for large interval ratios, falling to 3 ms with interval ratios near one.

For bi-cubic interpolation, cubic polynomials are used to define the weighting function for a convolution of at least 16 existing samples surrounding x_r and y_r. 16 chips require at most 110 ms, falling to 4 ms with interval ratios near one.

CONCLUSIONS

A number of image processing and image generation tasks have been coded for arrays of GLiTCH chips. The results show that fine-grain, associative, SIMD architectures hold promise as versatile, real-time image processing machines.

ACKNOWLEDGEMENTS

We wish to acknowledge the financial support of the Science and Engineering Reasearch Council, the Royal Signals and Radar Establishment and BP International Limited for this work.

References

[1] **Duller A.W.G., Storer R.H., Thomson A.R., Pout M.R. & Dagless E.L.** "An Associative Processor Array Designed for Computer Vision" *Elsewhere in these proceedings*

[2] **Davies S.T.** *The Implementation of the FFT on the DAP*, Centre for Parallel Computing, QMC London, DAP Report 6.39.

[3] **Cooley J.W. & Tukey J.W.** "An Algorithm for the Machine Calculation of Complex Fourier Series" *Mathematical Computing*, Vol. 19, 1965, pp 297-301.

[4] **Reddaway S.F.** "DAP - A Distributed Array Processor" *First Annual Symposium on Computer Architecture*, Florida, December 1973.

[5] **Flanders P.M., Hunt D.J., Reddaway S.F. & Parkinson D.** "Efficient High Speed Computing with the Distributed Array Processor" *High Speed Computer and Algorithm Organization* eds. D.J. Kuck et al. Academic Press 1977.

[6] **Fisher A.L., Highnam P.T. & Rockoff T.** "Scan Line Array Processors" *Hardware Accelerators for Electrical CAD*, eds Ambler T., Agrawal P.& Moore W., pp312-324, Adam Hilger 1988.

[7] **Fisher A.L.& Highnam P.T.** "Computing the Hough transform on a Scan Line Array Processor" *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 11 No. 3 pp262-265, 1989.

[8] **Duda R.O.& Hart P.E.** "Use of the Hough transform to detect lines and curves in pictures" *Comms. of the ACM* Vol. 15 No. 1 pp11-15, 1972.

[9] **Sanz J.L.C.& Dinstein I.** "Projection-based geometrical feature extraction for computer vision: Algorithms in pipeline architectures" *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 9 pp160-168, 1987.

[10] **Niblack W.** *An Introduction to Digital Image Processing* Prentice Hall International, London, 1986.

[11] **Warpenburg M.R. and Siegel L.J.** "SIMD Image Resampling" *IEEE Trans. on Computers*, Vol. c-31 no. 10, 1982.

[12] **Flanders P.M.** "A unified approach to a class of data movements on an array processor" *IEEE Trans. on Computers*, Vol. c-31 no. 9, 1982.