Associative Processor Arrays: Simulation and performance estimates for image processing

Andrew Duller
Andrew Morgan
Richard Storer

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

Abstract

The SCAPE chip, as an example of an associative processor array, is described; performance and algorithm details are given. The difficulties which were encountered when designing a SCAPE test-bed are stated and SCAPE is compared with a number of other image processing chips and system designs. The knowledge gained from this work is then used to suggest improvements in the design of VLSI associative processor arrays for image processing tasks.

To enable optimisation of associative processor array design a functional simulator has been written and results are given for a number of image processing algorithms. The design of a language which will complement these architectures is discussed.

1. Introduction

The Single Chip Array Processing Element (SCAPE) is a VLSI associative processor array developed at Brunel University (Jalowiecki I.P. and Lea R.M. [1986]). This research was carried out as part of Alvey project MMI 043 entitled "SCAPE based image processing and pattern recognition". The purpose of which was to design and build a test-bed system that would allow the concept of SCAPE to be proved in an image processing application.

In designing the test-bed and programming environment for a complete SCAPE system several difficulties were encountered with the SCAPE design.

These difficulties suggested a number of improvements in the design of the chip which will enhance both its performance and the ease with which it can be incorporated into a complete system. In addition this work pointed to a need to explore more thoroughly the class of processors, of which SCAPE is an example.

It is assumed at the outset that the principal application for this class of SIMD structures will be image processing. This assumption still leaves a wide range of application since it need not be restricted to low level, pixel based operation, but can include higher level processing such as image understanding and recognition. It is seen as vital that the chip design allows efficient higher level processing since low level operations can be performed relatively cheaply and efficiently using dedicated DSP chips.

We have compared SCAPE with other SIMD image processing architectures and are drawing conclusions as to the optimum architecture for a class of image related tasks. To this end a new general purpose simulation and programming environment has been designed to allow investigation into the effects of certain parameters in the design of such architectures. These parameters are identified and a number of results are then presented for a variant of the SCAPE architecture.

The need for a programming language that mirrors the hardware environment becomes clear when working with large, SIMD, associative processor arrays. The requirements for such a programming language are discussed.

2. SCAPE

A single SCAPE chip consists of 256 processors connected by a single shift register. Each processor containing 37 bits of content addressable store and a single bit arithmetic and logic unit.

A full description of the proposed SCAPE test-bed system can be found in Duller A.W.G et al [1987c] and only pertinent details will be given here.

The test bed comprises a chain of sixteen linearly connected SCAPE chips with associated frame stores, data stores and microcode sequencer (see figure 1). This forms a linear array of 4096 processing elements (PE's). For image convolution and other low level image processing algorithms a patch of 2-D image data is mapped row-wise onto the linear array, typically one pixel per PE. A result of this mapping is that nearest neighbour communication between PE's in different rows is inefficient. In order to overcome this, a narrower patch shape can be used although this increases the number of patches required to cover an image. Thus, there is a trade-off between the communication overhead involved with wide patches and the number of patches required to cover the image. These factors have been accounted for when estimating the execution time for a convolution algorithm.

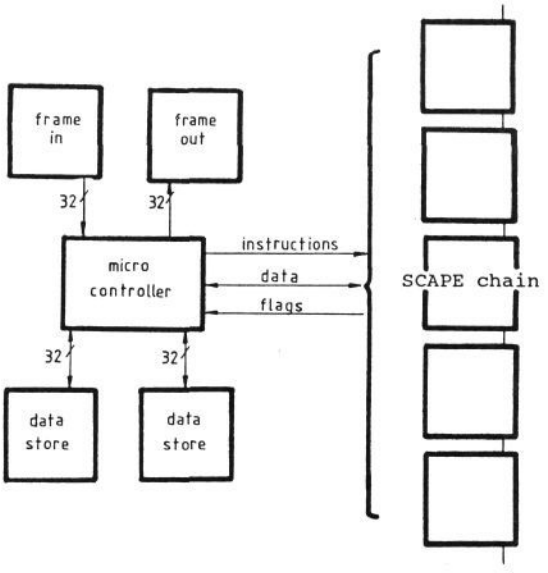

Figure 1.

2.1 Convolution with a chain of SCAPE chips

The algorithm begins by multiplying every pixel in the chain by a particular point in the mask. This scalar-vector operation is repeated for every point in the mask with the partial sums being shift/accumulated at the correct location in the chain at each step. The choice of scheme to shift/accumulate the partial sums is crucial because of the inter-row communication overhead. The following are a number of schemes that have been suggested. Figure 2 shows the case of a 3x3 mask.

In (i), as each partial sum is formed it is shift/accumulated in the PE that corresponds to the centre of the mask. In (ii), the partial sums are shift/accumulated at the PE that corresponds to the centre of the first row of the mask before being shifted to the PE that corresponds to the centre of the second row. This procedure is repeated until the last row of the mask. In (iii), the partial results are shift/accumulated at the PE that corresponds to the end of the first row of the mask before being shifted to the PE that corresponds to the beginning of the second row. This is repeated until the accumulated results appear in the PE that

AVC 1987 doi:10.5244/C.1.18

corresponds to the last element of the last row. These three schemes are compared in the next section.

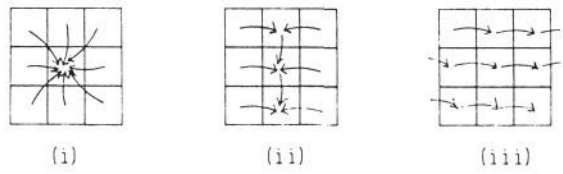

Figure 2.

2.2 Performance estimation

The execution time of a SCAPE chain performing a convolution or correlation depends on a large number of variables. The most important of these being :-

image size (M*N)

mask size (m*n)

patch size (p*q)

no. of chips in chain

I/O exchange time

precision of scalar, vector and result

In order to derive an expression for execution times that includes these variables, observe that

$$T_{conv} = \text{No. of patches}*[\, T_{I/O} + T_{process}\,]$$

where 'process time' can be split into the multiply and accumulate portions, as

$$T_{process} = \Sigma\, T_{shift} + \Sigma\, T_{arithmetic}$$

The I/O exchange time is constant for a given number of chips and the arithmetic time for a given precision can be estimated. The variables 'No. of patches' and '$\Sigma\ T_{shift}$' are more complicated and are defined below.

No. of patches:- for a given image size (M*N) and mask size (m*n) the number of patches (p*q) required to completely cover the image is given by the expression,

$$[(M-m+1)/(p-m+1)] * [(N-n+1)/(q-n+1)]$$

where p,q = 8,16,32,64,128,256,

and p*q = no. of chips * 256,

and [ ] means next highest integer.

This expression has been evaluated for sensible values of patch width p and mask size and the results are shown in the following table.

Image Size = 256x256;

Mask	Patch size 16	32	64
3x3	19	27	25
5x5	21	27	25
7x7	25	30	25
9x9	31	33	25
11x11	41	36	25
13x13	61	39	25
15x15	121	42	25
17x17	---	45	25
31x31	---	339	49

$\Sigma\ T_{shift}$:- this depends on the choice of accumulation scheme. For comparison of the schemes that were explained in the first section assume that both the patch (p) and mask (m) are square. Then the following expressions for no. of shifts may be derived;

For odd m, and

N_s = no. of scalar bits,

N_v = no. of vector bits,

N_{acc} = no. of extra bits to store accumulating result

$= (\log_2 m^2)$

method (i) $N_s N_v[(m^2-1)(1+mp)/4]$:term in m^3p

method (ii) $N_s N_v[m(m^2-1)/4 + p(m-1)]$:term in m^3

method (iii) $(N_s+N_v+N_{acc})[(m-1)(p+1)]$:term in mp

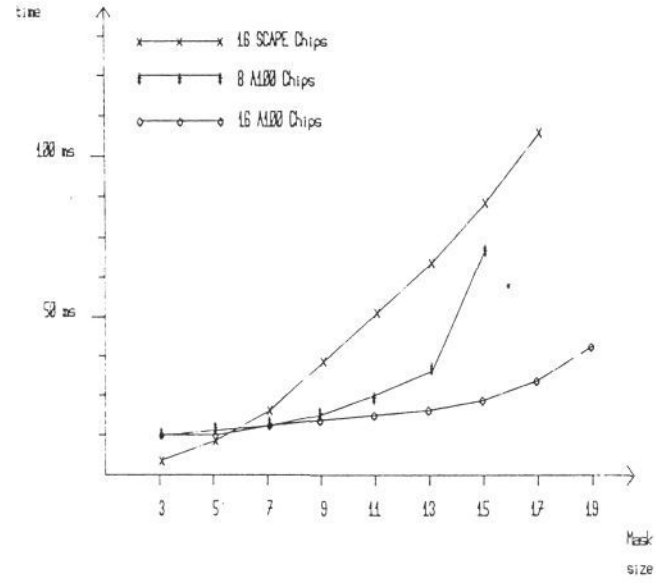

Figure 3

Obviously method (iii) is the most efficient way of accumulating the partial results and was used to estimate execution times. Figure 3 shows the execution times for a 16 SCAPE chip chain performing convolution/correlation with a variety of mask sizes over 256x256 image.

2.3 Observations on SCAPE system design

As a result of our experiences with SCAPE programming and system design we have compiled the following list of the difficulties encountered in the design of the test-bed and the associated software environment. A number of observations about the operating characteristics of such a system are included.

The microcode instruction for the SCAPE array has to be assembled external to the chip within 100ns, possibly from several storage locations.

Time must be allowed by the controller for extended instructions to execute, the time may well be data dependent and necessitate holding the current chip configuration in some form.

For greatest efficiency in loading image data, the data must be presented four pixels at a time. There is also a need to carefully rearrange the incoming data to avoid the bottlenecks which arise in shifting data large distances within a SCAPE chain.

An additional problem is created by the need to allow arbitrary image patch overlap for such algorithms as convolution.

I/O between the SCAPE array and the frame stores could account for a large percentage of execution time in many low level algorithms.

Linear connectivity of the array requires 'clever' algorithms to map 2-D image structure efficiently into 1-D array.

Lack of memory per processing element.

Associative nature of the array is of very little use in low level applications although it may be extremely useful for image recognition and understanding.

No hardware provision is made for counting the number of responding processing elements, for use in histogramming for example.

In paper studies which have been carried out on convolution and correlation algorithms (Morgan A.D et al [1987] and summarised in the previous section) the amount of time needed for I/O together with the organisational time (the time required to route data to the correct locations within the array) can be excessive when compared to the computational time. It seems clear that for most pixel based image processing tasks this is going to be the case with the current architecture.

The lack of memory associated with each processing element will often necessitate the use of more than one processing element per pixel. While this presents few problems in itself, it further exacerbates the problem of data routing because of the increased distances between pixels. In addition the efficiency of the array is greatly reduced and also the I/O problem is aggravated since greater numbers of image patches will be required.

Some of the difficulties which were associated with programming an array of chips such as SCAPE would certainly be alleviated through the implementation of a suitable high level language and this is discussed later in the section dealing with simulation.

2.4 Convolution with a cascade of IMS A100's

The IMS A100 digital signal processing chip incorporates 32 multiply-accumulators on a single chip organised as a pipeline. The architecture is a modification of a 32-stage transversal filter where the input signal is applied in parallel to all 32 multipliers followed by a delay and summation. A double-buffered bank of 32 coefficient registers provide the other input to the multipliers. There is also a 32-stage shift-register that carries forward results from previous points in the pipeline but this is only required when more than one A100 is used (Yassaie H. [1986]).

For convolution/correlation of 2-D images the elements of the 2-D mask are mapped into one of the coefficient registers. The image data is then presented serially for the parallel multiplication with the elements of the coefficient register. The results are parallel shift/accumulated into the next stage in the pipeline. After a pipeline delay valid results appear at the output. This method, however, produces a low percentage of valid results due to the way in which the input data is presented, for a 3x3 mask only 33% of the results are valid.

The recommended method for increasing the percentage of valid results appearing at the output is to insert 0's in between the rows of mask elements stored in the coefficient register. This has the effect of requiring the image data to be presented from a deeper patch resulting in a higher percentage of valid results appearing at the output. Again for a 3x3 mask it is possible to insert eleven 0's resulting in 86% valid results.

A single A100 is capable of convolution/correlation with maximum mask size of 5x5. Cascading these chips allows larger masks to be used and also makes the use of smaller masks more efficient (by inserting 0's to increase efficiency). With large images it is necessary to 'patch' the data where the patches are the width of the image and a depth determined by the allocation of the coefficient register.

2.5 Performance estimation

The maximum number of 0's that could be inserted for a given number of chips and mask size is given by the following expression

$$N_0 = \text{int} \left| \frac{32n - m^2}{(m-1)} \right|$$

where n = no. of chips,

and m = mask size,

and the number of patches required to cover the image is given by the expression

$$N_p = \left| \frac{M-m+1}{p-m+1} \right|$$

where M = image size,

and $p = N_0 + m$.

In some cases, the choice of the maximum number of 0's to insert results in the last patch containing useless data. In order to avoid this, choose the patch depth (p) such that the last patch has the maximum percentage of valid data.

Using the above expressions the performance estimates given in figure 3. were produced.

As can be seen from figure 3., a cascade of 8 or 16 IMS A100's can execute convolution/correlation algorithms faster than a chain of 16 SCAPE chips across a range of mask sizes. If real-time processing is required then a cascade of A100's provides a wider choice of mask sizes with a 256x256 image. These results show that the performance of a linearly connected associative processor array executing low level image processing algorithms is comparable to that of dedicated DSP hardware.

3. Comparison of Image Processing architectures

In order to grasp the many ways of designing an image processing architecture we have compared SCAPE to a number of other similar projects.

The comparison was made with the following four processor arrays:

LUCAS (Lund University Content Addressable System)

CAAPP (Content Addressable Array Parallel Processor)

The Connection Machine

AMT DAP 3,

the information being taken from the follwoing references, Fernstrom C. et al [1983], Weems C.C. [1985], Weems C.C. et al [1985], Weems C.C. et al [1987], Flanders P.M. et al [1977] and Hillis W.D. [1985].

Obviously there are many other computer architectures that could have been included in this type of comparison, however, this study has concentrated on massively parallel structures which can be constructed using the SIMD paradigm. The designs of the CLIP, MPP, GRID etc. were considered similar to DAP and they were therefore not included explicitly in this comparison. All except the LUCAS project are based on the design of a custom VLSI processor.

Perhaps the only feature which is common to all of the above computers is that they are designed around the concept of 1-bit processing elements. We have similarly taken this as a basis. In most other hardware respects the four machines mentioned above take differing routes to enable the exploitation of parallelism. From our study the following design parameters have been identified as crucial to performance.

4. Design parameters

4.1 Processors per chip

The number of processors (inapplicable to LUCAS since it is not a VLSI chip) on a chip varies greatly, with SCAPE possessing 256, CAAPP and DAP with 64 and Connection Machine with 16. The compromises that are represented by these numbers are more clearly seen later in the discussions on connectivity and memory size since they will have a direct bearing on this. However, it is clear that the more complex each processing element the fewer that can be placed on a chip. Equally the complexity of the connectivity that is required between processors on different chips plays a large part in determining the magnitude that this parameter can take. SCAPE represents the philosophy that packing as many processors as possible on a chip, at the expense of simple communication, is the way to enhance performance. At the opposite end of the spectrum is the Connection Machine

which has a complex routing network with only a few processors per chip.

4.2 Type of memory

The choice is associative, non-associative or a mixture of the two. The trade-offs will be between complexity of each memory cell and the additional speed which is possible with a parallel association mechanism. In order to answer this question it is necessary to evaluate the speed-up possible with associative memory and also the frequency with which it will be used. In all of the architectures considered arithmetic is performed in a bit-serial manner and this is assumed to be the case in all associative processor array designs.

4.3 Memory size

This relates to the amount of memory which is directly accessible to each processing element. This memory has to be split into on-chip and off-chip memory since it will drastically alter the speed of access and the I/O bandwidth required. Currently there are 32 bits of data memory available on SCAPE, the latest version of CAAPP has 3 x 128 bits which can be switched with external memory and the Connection Machine and DAP simply have blocks of external memory which have to be accessed for all operands. All of the work done so far indicates that the amount of memory has to be as large as possible, consistent with there being a 'large' number of processors per chip. It is important that all of the parameters should be chosen so that the amount of processing possible between frame store memory accesses is maximised.

The approach of DAP and the Connection Machine has the advantage of easy extensibility at the expense of access time.

4.4 I/O bandwidth and pinout

These two seem to be connected since more pins on the package will allow more data to be transferred to external memory and neighbouring chips. The pinout also affects the connectivity that is possible since N^2 PE's on a chip require 4N pins to allow full 2-D connectivity over the complete array.

These two parameters both need to be as large as possible, consistent with the physical constraints of the package. The I/O bandwidth is particularly important if low level image processing applications are considered.

4.5 Connectivity

The way in which processing elements communicate is vital to the efficiency with which any processor array can perform since the organisational time of many algorithms will dominate if the communications highways are not of sufficient bandwidth. We examine both the on-chip and off-chip connectivity strategies since they possess different constraints and are likely to be different.

Since LUCAS is not designed around a VLSI chip we deal with it first. The LUCAS array is arranged with cyclic linear nearest neighbour connections, but in addition there is a perfect shuffle network between the 128 processing elements. To implement a perfect shuffle on chip would be extremely costly in terms of silicon and is therefore ignored in the ensuing discussion.

4.5.1 On-chip connectivity

In the SCAPE design, on-chip connectivity is linear nearest neighbour, with longer communications being performed by means of a shift register to which all of the processing elements have access. The connectivity in the Connection Machine and the DAP is 2-D nearest neighbour. The connectivity of the CAAPP has undergone a number of changes since its initial design, and is now fully 2-D nearest neighbour.

From our experience with SCAPE the linear nature of the array the mapping of image data into the processors is vitally important. In some cases it seems impossible to efficiently map the data structures into the array and long execution times result. One of the questions which still has to be answered is whether or not it is necessary to have full 2-D connectivity. In the simulation section a small increase in connectivity is proposed which appears to be a good compromise between silicon area and the need for better communications.

4.5.2 Off-chip connectivity

Again SCAPE has linear connections off-chip, requiring only two 1-bit links with neighouring chips. In CAAPP and in the DAP the 2-D connectivity on-chip is extended off-chip, so that for the purposes of programming there is no difference between processors communicating on and off-chip. The Connection Machine is completely different in that the chips are conceptually placed at the vertices of a hypercube. Each chip possesses a router which transfers messages between distant processing elements. In this way, for an N-dimensional hypercube, the longest time to send a message between two processors is the time taken by N routing operations to take place.

4.6 Software considerations

Our experience with writing image processing programs for SCAPE has shown that associative processor arrays require unusual programming style. In the case of SCAPE a knowledge of some of the quirks of the VLSI design was needed to write efficient algorithms.

We feel that some sacrifices in VLSI economy will be justified if they allow a more consistent and useable instruction set to be implemented.

5. The need for simulation

The large number of variables in the design of an associative processor array dictates the need for an effective simulation of the machine to study their effects. The design variants need to be evaluated in a number of areas;

- execution time of image processing algorithms,
- ease of programming these algorithms,
- implications for supporting hardware,
- feasibility of implementation on silicon.

Our existing simulator for the SCAPE chip (Duller A.W.G. et al [1987a]) allows the programmer to use only one chip and takes no account of the supporting hardware functions needed in a real system. We have developed a simulation system which allows all the design parameters to be varied and example programs then executed on the processor array so described. This system allows for the evaluation of the simulated machine in all respects except its implementation on silicon.

A number of requirements were laid down for the new simulator to meet, as follows:

> As well as allowing variations in the function of the processing elements, it should allow us to study the effects of changes in interconnection networks between processors and between chips containing arrays of processors.
>
> It should be possible to simulate a complete, multi-chip image processing machine working on real image data and to do this reasonably fast, ie: a few minutes for low level operations such as convolution or threshold. The system should be able to simulate at least one video frame period of 'real time' machine operation without difficulty or long delays.
>
> It should simulate the supporting hardware, such as programmable frame stores, and allow for variations in their design.
>
> Wherever possible, chip design changes should be transparent to the user programs. A program for spatial filtering, for example, should run on the simulated system without alteration regardless of the machine simulated. This obviously precludes algorithms where different programming techniques have to be used in different variants of the machine.

A good user interface to the simulation should provide diagnostics for verifying results of the algorithm under test as well as the performance of the simulated machine. this should include the ability to set breakpoints in programs, single step execution, a display of the current contents of the processor array and of the images being processed. This is especially important when using the unfamilliar programming techniques needed for associative processors. A performance profile should be produced, showing the amount of simulated machine time spent on executing the various parts the algorithm, to highlight performance bottlenecks.

6. Description of the simulation

The simulator is written in C, for its speed and for its bitwise operators which are useful for modelling the action of bit-serial processors. A number of modules describe the functions of the various parts of the simulated system (see figure 4.). They are modelled as operations on global data structures which represent the physical data and register contents.

The lowest level module describes the memory size, registers and functions of a single processing element. Another module then defines the number of processors and their connectivity on a single chip and a third the connectivity between chips in a multi-chip system. Other modules model the supporting frame and data stores. Digitised photographs and some synthetic scenes provide the image data.

A user program, written in C, calls functions within these modules to move data around the simulated machine and to perform the image processing operations. As an example, this simple section of code reduces sixteen bit data to eight bits, in all processors:

```
clip_result ()
/* clip 16 bit result in bytes 1 and 2
   to 8 bits in byte 1 */
     {
     /* tag processors with no overflow */
     search_byte ("00000000", "", 2);
     /* set all others to 255 */
     activate (notself, 1);
     write_byte ("11111111", "", activated, 1);
     /* find all those with sign bit set */
     reset (left, "2", "2");
     search_bit ("1x", "");
     if (global.tr)
          /* if there are some, zero them */
          {
          activate (self, 1);
          write_byte ("00000000", "", activated, 1);
          }
     }
```

A user interface module is able to interrogate the data structures and display the contents of the processor arrays and frame stores at any point during execution. The frame stores can be displayed in either 16 or 64 grey levels using dithering on a monochrome display. This is the only part of the simulation which is machine dependent and is written for Apollo workstations. All results are shown in independent screen windows which the user can move, scale and overlap. This allows for simple comparison of results from a number of processing stages.

During the simulation, a trace is produced of the instructions executed by the simulated machine. This is then analysed by separate software to produce a timing profile of the algorithm executed. Full account is taken of any parallel activity within the hardware, which causes instructions to be overlapped.

There are two ways in which variations of associative processor array machines can now be studied. A user program can contain a number of parameters in a C header file which alter the configuration of the simulated sytem. Factors such as the size of memory per processor, the number of processors per chip and the width of various data paths can be altered in this way. Secondly, the functionality of a part of the machine can be altered by selecting another C module to model that part. There may be several Controller Model modules, for example, for the programmer to choose from.

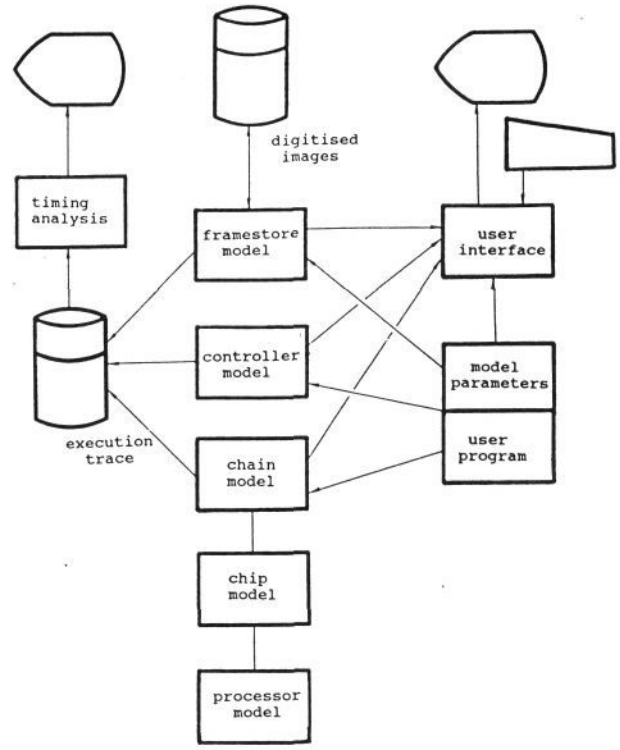

Figure 4.

7. Simulation Results

The first task applied to the simulator was to model the SCAPE test bed designed by Alvey MMI 043 (Duller A.W.G. et al [1987c]). This consists of a chain of 16 SCAPE chips (4096 processors) receiving instructions and data broadcast from a microcode sequencer and data and image frame stores (see figure 1.).

A convolution of a signed, four bit, 3x3 mask over a 256x256, eight bit image was performed on this machine. Using predicted execution times for SCAPE instructions (Duller A.W.G et al [1987b]), the simulated execution time was, typically, 3ms for a Laplacian operator, including time to load the image, which took two minutes of simulator time on an Apollo DN330 workstation.

Other algorithms looked at so far have been largely low level tasks. These have included convolution, thresholding, thinning, contrast stretching and histogramming. The table below gives results for 16 SCAPE chips and a 256x256 eight bit image. In the near future we will have evaluated higher level algorithms such as texture segmentation and object classification based on the Hough transform.

Operation	Execution time	Time for Simulation
Exchange Image	2.0 ms	17 sec
Laplacian	0.9 ms	104 sec
Edge Enhance	1.9 ms	243 sec
Contrast Stretch	0.8 ms	100 sec
Threshold	0.03 ms	7 sec
One pixel thin	1.1 ms	38 sec
8 bin histogram	26.0 ms	558 sec

7.1 An Improved Loading Scheme

A number of conclusions have, however, already been drawn. The existing design for SCAPE requires the image data to be sent along the same 32 bit path as program data and thus requires a complex frame store design and various multiplexers in the micro controller. It also means that most of the processors in the chain are lying idle while patches of image data are transferred from the frame store.

We propose the addition of a load channel to each chip allowing eight bit image data to be shifted into the chain concurrently with processing on the previous set of data. When loading and processing are complete, the data in the load channel is exchanged for data in the processors. The processed data will then be shifted out as the third set of image data is shifted in (see figure 5.).

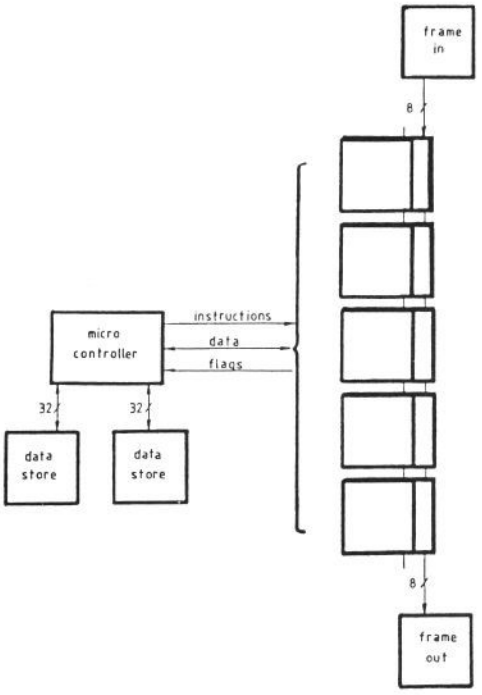

Figure 5.

Simulation of this variant gives a loading time of 205μs for a 4096 pixel patch. Although this is twice as long as the predicted time for SCAPE, it's concurrency with processing gives an overall saving for any algorithm which uses more than 100μs of processing time per image patch. Furthermore, this modification does not preclude the use of the existing method.

7.2 Improved Processor Connectivity

The high density of SCAPE processors on silicon is achieved at the expense of inter-processor communication. We are examining possible compromises which will improve communication while maintaining a high processor density.

One such scheme introduces a shift register which transmits single bits between every 64th processing element. The time to perform algorithms such as histogramming can be reduced significantly and in fact any algorithm that requires summation of data over the entire array will benefit from this.

The result of these modifications are encouraging especially when the amount of processing per patch of data is large. The reduction in shifting time is dramatically reduced in an application such as histogramming. Using the classical SCAPE design the shift time accounts for 26ms out of a total of 30ms, whereas with the extra connectivity this is reduced to 6.8ms out of a total of 10.7ms.

8. Future Work

Further work on the simulation and design of associative processor arrays is the subject of a current SERC funding application.

Improvements to the simulation system are being made continually. In particular, there is a need for a more sophisticated method for estimating execution times so that complex changes in chip design automatically alter the timing model.

8.1 An Array Programming Language

Programming the simulated machine in C is not really satisfactory as it allows the programmer to use control constructs and data structures which have no physical equivalent in the machine. Equally, the syntax of C is often awkward in expressing those functions that do exist, especially where operations carried out in parallel have to be described. Alvey MMI 043 proposes a programming language for SCAPE (Storer R. et al [1987]) which would be compiled to symbolic object code for execution on either a simulator or the SCAPE test-bed. We envisage that a general form of this language will be used for programming simulated variants of associative processor arrays.

The language contains instructions to both the processor array and supporting hardware, embedded in the simple control structures that could be executed by a microcode sequencer. For the current work, programs will be compiled to an equivalent C program containing calls to the C functions that make up the simulator. In addition to program instructions, the language will allow specification of the parameters for the modules modelling the simulated system.

The syntax of the language is similar to C in its control constructs, use of functions and the use of a preprocessor for including other source files and defining macros and constants. Instructions to the processor array borrow conventions from other array processing languages (Hockney R.W. and Jesshope C.R. [1981], Perrott R.H. [1979], ICL [1979]).

Usually, in these languages, arithmetic operations can be written which refer to whole arrays, or parts of them, all the elements of which are then processed in parallel. In the expression, the name of the array is followed by a description of those rows and column addresses that are to be processed, in square brackets. For example:

PICTURE [1, 5:17]

refers to the thirteen elements in row 1 with addresses 1,5 to 1,17 in the array PICTURE.

In an associative array, the individual elements do not have row and column addresses, they are identified instead by matching all or part of their contents to a given ternary pattern (0, 1 or x for "don't care"). Only those processors with matching patterns will perform the operation specified. The instruction broadcast to the array is followed by a description of the pattern which identifies the required processors, in the usual square brackets. For example, the instruction:

TAG [ ][ ]

would set the tag register in all processors. Alternatively, the whole data field (32 bits in SCAPE) of the processor may be used for matching:

TAG [word FFFF1100] [ ]

or one ternary byte:

TAG [byte0 11xx0011] [ ]

or one ternary digit from each of a number of predefined fields:

TAG [p134 101] [ ]

In addition, a processor can be further identified as belonging to a given subset. The subset bits can also be used in matching:

TAG [byte2 xxxx1111] [11xxx]

This format is used throughout the syntax. In general, each array instruction has the form:

instruction [processor selection] data

Processors which satisfy the selection criteria will perform the instruction with the data supplied. The example "clip_result" is repeated here in this new syntax.

```
clip_result
   ! clip 16 bit result in bytes 1 and 2
   ! to 8 bits in byte 1
     {
     ! tag processors with no overflow
     tag [byte2 0] []
     ! set all others to 255
     write [~tag] byte1 = 255
     ! find all those with sign bit set
     reset left p2 m2
     tag [p2 1] []
     if tr
          ! if there are some, zero them
          write [tag] byte1 = 0
     }
```

8.2 Faster simulation

In order to implement more complex algorithms, such as object classification, the possibility of implementing the simulation on an existing array processor is currently being explored and some trial software being written in DAP Fortran for the ICL DAP at Queen Mary College.

The mapping of such a simulation onto an array processor like the DAP is extremely straightforward and has the added advantage of being very fast. The main drawback with using a remote machine in this way is getting data to and from the machine for display and post-processing. When the input and output from the simulation are both 256 x 256 images this is not a trivial matter. Machines such as the AMT DAP 3, which can be attached to a workstation, will alleviate this problem.

8.3 Trial Chip Fabrication

As the design parameters for associative processor arrays become better understood, it is intended to fabricate some trial examples to evaluate their real performance and suitability for manufacture.

9. Conclusions

So far we have considered only minor changes to the original SCAPE architecture. However, it is clear that tools such as we have described ease the process of evaluating changes in chip design. We aim to define a set of image processing operations which can be used to evaluate any new chip configuration. Clearly the coding and strategy of the algorithms will change from architecture to architecture but the framework can remain the same, thus reducing development time.

10. Acknowledgements

We gratefully acknowledge the financial support of the Alvey directorate and the guidance of Professor Erik Dagless. In addition, thanks to; Quantel Ltd., Brunel University and in particular Professor Mike Lea for his unerring cooperation throughout this project.

Thanks also to Stuart Green for supplying synthetic images and to Mike Pout for algorithm development on the new simulator.

11. References

Duller A.W.G., Morgan A.D. and Storer R. (1987a)
The Apollo SCAPE simulator
University of Bristol ITRC report no. 100

Duller A.W.G., Morgan A.D. and Storer R. (1987b)
A description of the SCAPE instruction set
University of Bristol ITRC report no. 101

Duller A.W.G., Morgan A.D. and Storer R. (1987c)
An evaluation system for the SCAPE array processor
University of Bristol ITRC report no. 102

Flanders P.M., Hunt D.J., Reddaway S.J. and Parkinson D. (1979)
Efficient high speed computing with the Distributed Array Processor
High Speed Computer and Algorithm organisation Ed. Kuck D.J., Lawrie D.H. and Sameh A.H. (Academic Press)

Fernstrom C., Kruzela I., Ohlsson L. and Svensson B. (1983)
An Associative Parallel Processor used in Real Time Signal Processing
Signal Processing II, EURASIP, 1983.

Hockney R.W. and Jesshope C.R. (1981)
Parallel computers
Adam Hilger Ltd., Bristol

Hillis W.D. (1985)
The Connection Machine.
The MIT Press, Cambridge, Massachusetts 1985.

ICL (1979)
DAP Fortran language reference manual
ICL tech. pub. TP6918

Jalowiecki I.P. and Lea R.M. (1986)
SCAPE - A programmable VLSI chip for Signal and Image Processing
Brunel University Report

Lea R.M. (1985)
A VLSI array processor for image processing
Algorithmically Specialised Parallel Computers, (Academic Press) pp 159-168

Morgan A.D., Duller A.W.G. and Storer R. (1987a)
A description of Convolution and Correlation algorithms for a chain of SCAPE chips and a comparison with the IMS A100
University of Bristol ITRC report no. 103

Perrott R.H. (1979)
A language for array and vector processors
ACM Transactions on Programming Languages and Systems, Vol. 1, No. 2 pp 177-195

Storer R., Duller A.W.G. and Morgan A.D. (1987)
A programming Language for the SCAPE array processor
University of Bristol ITRC report no. 99

Weems C.C. (1985)
The Content Addressable Array Parallel Processor: Architectural Evaluation and Enhancement.
ICCD 85.

Weems C.C., Lawton D. and Levitan S. (1985)
Iconic and Symbolic Processing Using a Content Addressable Array Parallel Processor.
Proc. SPIE Vol. 504 1985 pp 92-111.

Weems C., Levitan S., Riseman E. and Hanson A. (1987)
The Image Understanding Architecture
Proceedings of the DARPA Image Understanding Workshop, Los Angeles, CA, Jan 1987.

Yassaie H. (1986)
Correlation and Convolution with the IMS A100
INMOS application note 3.