PETAL: A parallel processor for real-time primitive extraction

Kirk Martinez and Don E. Pearson

Department of Electronic Systems Engineering, University of Essex
Colchester, Essex, CO4 3SQ, England

Abstract

A real-time parallel processor named PETAL is described which has been developed to extract cartoon primitives from grey-level television images. It is based on a cascaded look-up table architecture and is controlled by a 68000 microcomputer. It can process 256x256 images at 50 frames/s.

Introduction

We have an interest in the extraction of cartoons or sketches from moving grey-level images. These cartoons convey a great deal of information about the shape, size, location and movement of objects in the scene, but require very low data rates. Studies have shown that 64x64 pel images of this kind contain sufficient information for the deaf to be able to communicate by signing at a data rate of less than 9.6 kbit/s [1]. More recently we have been experimenting with the use of such cartoons as primitives from which grey-level images can be reconstructed [2]; here the application is at slightly higher resolutions and data rates (up to 64 kbit/s) to areas such as videoconferencing.

To extract economical cartoon representations from television images, a non-linear convolution operation is required which detects luminance valleys in the image [1]. Our initial real-time implementation used 3 frame-store boards and a 68000 processor. By using various methods to gain speed, and some simplifications of the algorithms, we obtained an output of 6 frames/s with 64x64 pel images. But we could not use a convolution operator larger than 3x3, although non-real-time tests had shown that 5x5 operators were superior; these could take up to eight minutes for one 512x256 pel frame (figure 1). An investigation was then carried out into the use of a signal processor as a sub-processor and speed gains of a factor of two obtained [3].

These restrictions caused us to become involved in parallel processing architectures in order to gain increases in speed; we hoped thereby to be able to process larger pictures at higher frame rates, as well as to use more complex algorithms. Our initial studies [4] led to a proposal for a simple low-cost real time processor called PETAL. In this contribution we describe a working prototype of PETAL which is capable of processing 256x256 pel pictures at 50 frames/s in real-time using a 3x3 algorithm. We also mention certain extensions which will increase its versatility further.

Figure 1 An example of a cartoon extracted from a grey-level image

Algorithms for primitive extraction

Only a small neighbourhood around each pel is needed to determine whether there is a feature or not. We have experimented with 3x3 and 5x5 pel regions. The 5x5 algorithms are described elsewhere [1]; a 3x3 extraction algorithm is described below as this was the one actually implemented. The algorithms were originally designed to run on a 68000 system and thus make use of fast techniques for such a processor: non-linear comparison techniques and simple arithmetic.

The 3x3 feature extraction operator

This is a simplified description of the algorithm used to detect features in the original grey level image:

Take a 3x3 region around each pel P:

a	b	c
h	P	d
g	f	e

Then there are two basic stages:

1) for each neighbour (a....h) } referred to later as
   check if (neighbour - P) > edgethreshold } difference detections

2) using these results, see if there is a luminance valley at P } referred to as the final
   if valley then the result = black } decision
   else result = white

This second stage may involve comparing counters set by the first, for example an N counter for "northern" differences (a,b,c). The results are usually added to a simple threshold of the image to give a more realistic representation. A suitable choice of thresholds is needed to give good results.

PETAL

The requirements of a real-time architecture for these algorithms are low cost, small size, some flexibility and high speed. These point to a dedicated architecture using reasonably fast devices and the sufficient use of parallelism.

The prototype PETAL was devised by decomposing a 3x3 algorithm described above into its independent parts: a window-read, eight large difference detections, and a logical operation on the eight results. A common method of tapping the raster scan with line delays was chosen for the window-read. Thus a new window is produced every pel period of 90ns. It was noted that each of the difference detections could be implemented as a look-up table, as could the final decision. As these stages are independent they can be pipelined, which allows 90ns for each stage. This pipelining allows moderately fast devices to be used (which reduces the package count). Figure 2 shows a simplified diagram of PETAL.

The digital raster scan was obtained from memory-mapped image store-boards, and the output was fed back to one store-board. The line delays were most affected by the 90ns time restriction, as they were made from SRAM, using a read-modify-write cycle. In this technique the address to the SRAM is clocked for each new pel, the old pel read, and the new one written. The address, which is from a counter, is cycled around a limit which determined the delay. It was here that fast ASTTL devices were necessary, the rest were all standard LSTTL. This method requires few devices for each line delay compared to a shift register technique. Each one can obtain the address count from the same counter and high density SRAMs can be used. The prototype had 256 pel delays for convenience so it could only process 256x256 images.

The look-up tables were implemented with single 4kx1 55ns SRAM devices. Each one required a bus connection to the 68000 for memory mapping (not shown for clarity). This is switched to high impedance when the system is processing images, and the pipeline registers are enabled. An initialisation of the look-up tables is required, but afterwards the system is left to run at real-time rates.

As can be seen in Figure 2, the window scanner provides nine new pels in parallel for each incoming pel from the store. Neighbour-centre pairs then drive the address inputs of eight SRAMs. The eight outputs from these drive the last look-up, which produces the result to be sent to the store board. When a processed image is required, a frame grab command is sent to the store-board, and it receives processed data from the next frame. Only one board was needed for the PETAL hardware.

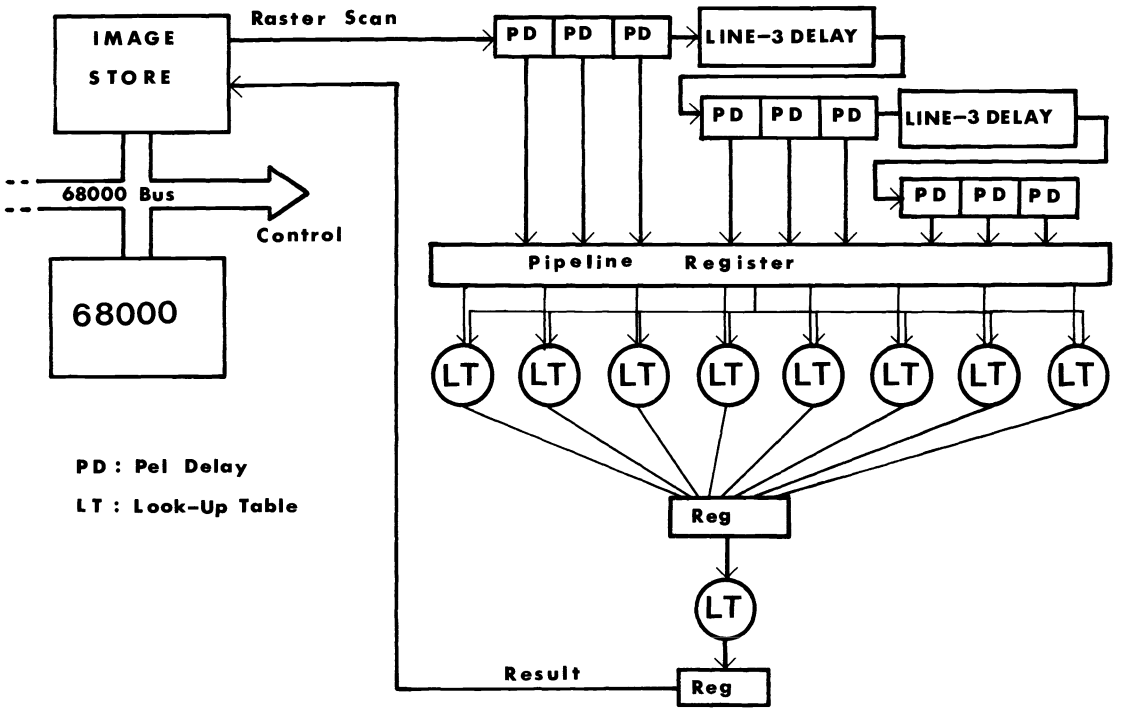

Figure 2 Simplified diagram of PETAL

Window scanners and look-up tables have been used in VAP [5] and the Cytocomputer [6]. VAP uses six-bit output tables, with each pair cascaded to another, to gradually reduce the number of bits involved. This was designed for more general neighbourhood operations. Cytocomputer uses a pipeline of 88 3x3 operators, each of which is a binary function implemented with 256 byte rams, also for general use. In contrast PETAL was designed for a specific type of algorithm and uses look-up tables roughly midway between those used in VAP (4kx6) and Cytocomputer. Some extensions to the PETAL system are presented later in this paper. These are still based around certain algorithms rather than an attempt to produce a general machine.

PETAL software

The description of a system would be incomplete without mentioning its software as this can affect its effective performance so drastically. The driving software, which runs on the 68000 (8 MHz), is all written in the C language [7] dowloaded from a VAX 11/750. As the look-up tables and control registers are memory-mapped into the 68000 address space they can be treated as an array. A register controls whether the look-up tables are memory-mapped (loading) or linked to the data flow (running). The programmer provides two functions, one for each set of tables. These process a general input to the corresponding table (e.g. two pels for the first ones) and give an appropriate result. For example these (simplified) functions would detect East West valleys:

```
neigh_func()
{
if ( centre - neigh > edge_threshold)
        return(1);
else
        return(0);
}
```

```
end_func()
{
if(neigh.bit.n && neigh.bit.s)
        return(BLACK);
else
        return(WHITE);
}
```

then a simple command loads PETAL with the appropriate data:

```
        petal(neigh_func, end_func);
```

This petal function steps through all possible inputs to the look-up tables and stores the result in the appropriate place. The first eight tables are mapped to look like a 4kx8 array, each bit of a byte corresponding to a particular SRAM. As these tables usually

hold the same data, the initialisation is thus simplified and faster.

The first initialisation takes one or two seconds for the 3x3 algorithm described (it is obviously algorithm dependent). Clearly other algorithms can be implemented, some easily, some with alterations (this is the old problem of matching an algorithm to an architecture). In general, the possible functions performed by PETAL can be expressed as:

result(f1(a,P) , f2(b,P) , f3(c,P), ... f8(h,P))

a....h being the neighbours of the centre pel P
f1...f8 being the neighbour pair functions
each function returning a one bit result

This is fairly restricted but various functions can be implemented, due to the use of look-up tables. In the case of feature extraction, non-linear thresholds can be investigated, whereas if the system had been implemented with adder and comparison chips this flexibility would be lost.

Performance

Once initialised, PETAL produces output at video rates: 50 frames/s (256x256 pel frames). If it were used to process only one frame, before a change in the algorithm, then the initialisation time would become dominant. During the initialisation a maximum of 4096 + 256 possible inputs to the tables must be considered; this is roughly equivalent to only 6% of the data in one frame. Thus an initialisation followed by processing one frame would be expected to be around fifteen times faster than the 68000. This is only an estimate, as only one of the eight pel-pairs normally processed needs consideration during initialisation. The 68000 takes around 20s to feature extract a 256x256 frame using a 3x3 algorithm written in C. A typical PETAL initialisation is 1.7s, giving a factor of twelve times faster (for one frame). Continuous processing is thus around 1000 times faster. If the same algorithm is to be re-used but with one parameter change, then often only a small part of the look-up tables needs changing: only 4% of the first tables need changing if the "edge_threshold" changes by one. Pre-stored data directly loaded is also a fast method. Thus by intelligent re-initialisation the look-up scheme loses some of the disadvantages of initialisation often quoted for such systems.

PETAL provides a case where speed ratings in terms of MOPS (million operations per second) etc. are not valid. For example, in terms of look-up operations (excluding the window scanner) it has a speed of 100 MOPS. In terms of the original ALU operations a microprocessor would have to carry out, one could say the effective speed was 500 MOPS.

Parallelism in PETAL

Conventional parallelism can be found in PETAL such as pipelining and MIMD [8] type processing. In order to clearly express how the system achieves speed apart from these one can say that look-up tables are a form of parallelism with respect to the original operations that they replace. Like other forms of parallelism they have drawbacks such as initialisation time and size limitations.

The window scanner provides neighbourhood parallelism [9], in that the neighbourhood data are presented in parallel. This replaces the serial read operations required on a von Neumann system [10]. The technique is known as a data-flow approach, as it is the data which controls the sequence of events, rather than a program.

PETAL extensions

As the basic PETAL design is based around a 3x3 window and the raster scan stream, it seems logical to make use of this for some related operations. Work is currently in progress on the implementation of some possible additions to PETAL to make it a more general processor.

The prototype can only process incoming video. It is also necessary to allow input from any of the three image stores for operations on stored images. If the line delays were made with four-bit devices, then the eight-bit width would accommodate data from a binary image from another store. This means that two 3x3 windows are produced, one binary, the other grey. Expansion of the final table output to six bits (now known as the grey neighbourhood operation GREYNOP) allows more information to be output to the store and grey level results to be obtained. Operations on binary input images such as shrink/expand can be carried out by another look-up table. This only needs nine inputs, so one bit from the GREYNOP result can be included to allow binary operations on two images (e.g. mask). If a 4kx1 SRAM is used, the other two address inputs can be used to select one of four look-up functions (useful for fast function swapping). A 4kx6 look-up on the centre pels

of the two selected images is useful for temporal pre-filtering and other two-image point operations. If this is located before the window scanner, the input can be pre-filtered before further processing.

Input and output selectors allow any two inputs and outputs, thus two processes can run simultaneously; in effect, there are parallel pipelines. A simple counter which could select one of the image streams could provide counts of a particular value (for histograms), the number of white/black pels (useful in pattern recognition) and so on. Registers would control the selectors in all the above.

For feature extraction algorithms these changes will allow an input image to be temporally pre-filtered, and a cartoon generated simultaneously. Isolated spots could then be removed by a pass through the binary operator. By collecting suitable information about the 3x3 neighbourhood and storing it in a frame store, a second pass can use information outside the normal 3x3 region. It is hoped that simple 5x5 operators can thus be decomposed into two passes.

The future

Dedicated systems offer very high performance/cost ratios. This suggests that a set of fundamental image processing operations, implemented as dedicated sub-processors, could give very high speed while not restricting the possible algorithms. This approach is used in general computing where, for example, floating-point processors are common. By using parallelism at a more local level, the problems of programming systems using a more general parallelism might be reduced. More standardised modular systems could be produced with this technique, which could allow greater interaction within the image processing field. This will be facilitated by the growing use of custom and VLSI devices.

Conclusions

We have been involved with the data-rate reduction of moving images to low levels. The complexity of the algorithms required a very fast processor. General-purpose systems were found to be too expensive, large and sometimes too slow. This led to the design of a system utilising parallelism matched to that found in the algorithms. Look-up tables were found to be flexible and fast enough for our purpose. A data-flow approach was taken and a video-rate real-time system produced (PETAL).

One type of processor has been considered in detail, for one set of algorithms. It was found that the strong interaction between the algorithms and hardware produced very high speed. This demonstrated the trade-off between speed and flexibility. Through slight relaxations in the hardware-algorithm matching, a useful range of other processes can be carried out.

References

1. Pearson, D.E. and Robinson, J.A., "Visual Communication at Very Low Data Rates", Proc. Institute of Electrical and Electronic Engineers, 73 (4) pp.795-812, April 1985
2. Hanna, E., Pearson, D.E. and Robinson, J.A., "Low Data-Rate Coding Using Image Primitives", Proceedings of the ANRT/SPIE Image Processing Symposium, Cannes, France, 2-6 December 1985
3. Mizuno, S., Pearson, D.E., and Robinson, J.A., "Real-time feature-extraction architecture for moving-picture transmission over telephone lines", Electronics Letters, 19 (22), pp.949-950, October 1983
4. Martinez, K. and Pearson, D.E., "Algorithmic Complexity, Speed and Architectural Parallelism in Low Data-Rate Visual Communication", Proc. Fourth International Conference on Digital Processing of Signals in Communications,(62), pp.185-191, University of Technology, Loughborough, England, 22-26 April 1985
5. Keller, H.J., Comazzi, A., and Favre, A., "Fast multi-image interaction and hierarchical processing, a new video array processor (VAP-80)", Proc. 6th International Conference on Pattern Recognition, Munich, Germany, 1, pp.246,249, 19-22 Oct. 1982
6. Sternberg, S.R., "Biomedical Image Processing", IEEE Computer, 6 (1) pp.22-34, 1983
7. Kernighan, B.W. and Ritchie, D.M., The C Programming Language, Prentice-Hall Inc., New Jersey, 1978
8. Flynn, M.J., "Some computer organisations and their effectiveness", IEEE transactions on Computers, C-21, pp.948-960, 1972
9. Danielsson, P.E. and Levialdi, S., "Computer Architectures for Pictorial Information Systems", IEEE Computer, 14 (11), pp.53-57, 1981
10. von Neumann, J., Theory of Self-Reproducing Automata, University of Illinois Press, Urbana, 1966