The Disputer : A Dual Paradigm Parallel Processor for Graphics and Vision.

Ian Page
(Programming Research Group, Oxford University)

Published in "Parallel Processing and Vision", Ian Page (Ed.),
Oxford University Press, 1988.

Abstract

This paper reports the design and programming of a parallel processor which is a coupled arrangement of a 256-processor SIMD machine and a 42-processor MIMD machine. Many applications, particularly in the areas of computer graphics and vision, naturally exhibit extensive data-parallelism in some of the key algorithms and extensive control-parallelism in others. We have constructed a computing engine which encompasses both a data-parallel computing paradigm in a 16 x 16 SIMD array, and also a control-parallel paradigm in a MIMD network of transputers. The two parallel machines are closely coupled and the SIMD machine has a video output channel for the generation of real-time animations (with video input being added soon). The entire system is programmed in Occam 2 which will eventually be extended to include facilities specifically for certain array data types and operations handled in the SIMD machine. This system is now allowing us to investigate dual paradigm algorithms effectively and we report some of the early results from using this novel parallel processor for applications in both graphics and vision.

1 Introduction

1.1 S.I.M.D and M.I.M.D Processing for Graphics and Vision

Image generation in computer graphics and model formation in image understanding are two sides of the same coin, whose currency is computational geometry (Faux and Pratt, 1982). Algorithms at the 'back-end' of the graphics pipeline and at the 'front-end' of machine vision applications, work almost entirely in the domain of two-dimensional arrays of picture elements (pixels). These algorithms can generally be characterised as local-support operations on potentially very large arrays of pixels. For this type of algorithm, the SIMD (Single Instruction Stream, Multiple Data Stream, or data-parallel) model of computation on a two-dimensional array processor is often the most cost-effective parallel implementation strategy.

Locally-based computations are in fact more powerful than was once recognised, and a great deal of work has been put into algorithms that can compute global state from local interactions over the last few years. Some examples are structure from motion (Ullman, 1979), surface interpolation (Grimson, 1981), optic flow (Horn and Schunk, 1981), shape from shading (Ikeuchi and Horn, 1981), stereo (Marr and Poggio, 1982), and image restoration (Geman and Geman, 1984). The development of such algorithms is changing our ideas about the computational nature of intermediate-level vision processes and we can expect many more innovations in this area.

For the algorithms earlier in the graphics pipeline and for many intermediate-level and higher-level vision problems, the algorithms typically need much longer range and unpredictable support in the data, with computation being centred on a relatively small number of 'regions of interest'. Thus, the most useful type of parallelism will often be task-oriented and it is more reasonable to use the MIMD (Multiple Instruction Stream,

Multiple Data Stream, or control-parallel) model of computation. It should be noted however that there has been relatively little work to date on the parallelisation of higher-level vision algorithms, so we know relatively little about the architectures required. For our own work, we are proceeding by making use of available fixed arrays of transputers. In the future we hope to use either a dynamically-switched network of transputers using the recently available transputer crossbar switches, or an ALICE-like machine (Cripps, Field and Reeve, 1986).

1.2 SIMD Processors for Graphics and Vision.

The seed that germinated into today's bit-serial, SIMD processor arrays was a paper by Shooman (1960) which noted that arithmetic over two-dimensional arrays of numbers was best carried out bit-serially on all the numbers at once. The advantage of this method is that the design of the serial adder is very simple and that no carry delays are incurred, these being the bane of the parallel adder designer's life. Pipelining was at the time well recognised as a method for increasing the throughput (computational bandwidth) of hardware at the expense of increased latency. Shooman's bit serial arithmetic operations are effectively very heavily pipelined since there is a pipeline register between every bit slice in the adder. The same hardware circuitry is re-used for each bit in sequence, so that the pipelining actually occurs in the temporal rather than the spatial domain. The benefit remains, however; there is no carry propagation delay and there is consequently no need to design complex circuitry to overcome it, such as carry lookahead networks.

The first design of a bit-serial parallel computer based on Shooman's observation was the Solomon computer (Slotnick 1960). Solomon was never actually built but it was the direct inspiration for Illiac IV computer (Falk, 1976). This was eventually built in a somewhat reduced form and never achieved its design objectives. Illiac was however the first real parallel computer and has inspired many of today's SIMD processors such as the DAP (Reddaway, 1973), CLIP (Duff, 1976), MPP (Batcher, 1980), and the Connection Machine (Hillis, 1985) as well as the SIMD part of the machine described in this paper. Good general texts on parallel processors are Hwang and Briggs, 1984 and Hockney and Jesshope, 1981.

1.3 RasterOp.

RasterOp (Newman, 1979), also known as BitBlt (Bit Block Transfer), is a primitive operation in computer graphics that operates over two-dimensional arrays of bits, and has been extended more recently to arrays of scalar and colour-valued pixels. In its simplest form, RasterOp takes two similarly-sized rectangular image portions, source and destination, and performs :

```
destination := destination OP source
```

where OP is one of a pre-defined set of functions mapping pairs of pixels into single pixels. Hardware support for this operation has been a big factor in increasing the performance of a number of workstations, starting with the Xerox Alto (Thacker et. al, 1979). RasterOp is a key algorithm in bitmap-based graphics, and can even sensibly be employed as the only primitive which directly updates the frame buffer. Many non-trivial and non-obvious operations can be accomplished using this primitive (Goldberg, 1983) and it is very closely related to the bit-serial arithmetic operations of conventional SIMD processors.

Due to the great usefulness of RasterOp and the fact that many computers now support it in hardware, it has come to be used in a number of unexpected places. In particular, RasterOp is critically involved in the implementation of the following vision algorithms at the MIT Artificial Intelligence Laboratory:

- The optimal edge finder developed by Canny (1983).
- Image processing algorithms such as edge thinning and edge following (Rosenfield and Kak 1976, Ballard and Brown 1982).
- Hildreth's (1985) motion analysis algorithm that tracks edges isolated by Difference of Gaussians (DOGs) filters at successive time intervals.
- Marr-Poggio-Grimson (MPG) stereo (Grimson, 1985).
- Terzopoulos' (1982) multilevel surface reconstruction algorithm (and, by extension, a variety of finite element algorithms).

To understand the role played by RasterOp in the implementation of such algorithms, consider MPG stereo. Step one requires that zero-crossings of DOG filters (computed by convolution hardware or software) be isolated and marked in the left and right images. The presence or absence of an edge at a particular location in one of the images can be recorded in a bitmap, and the program maintains two bitmaps Bl and Br for the left and right images. The correspondence computation that is at the heart of stereo then amounts to matching bits in Bl and Br after allowing for a lateral offset (the cyclopean disparity), assuming prior image rectification (the epipolar constraint). This matching operation can be realised by several invocations of RasterOp applying simple Boolean operations between Bl and Br with a range of different offsets.

1.4 DisArray and the Disputer

Earlier work on a SIMD machine for graphics applications (Page, 1983) resulted in a hardware system which significantly reduced one of the major bottlenecks of real-time graphics, namely that of rendering images into a frame buffer. The initial motivation for building this system (DisArray) was to execute the RasterOp (Newman, 1979) primitive in parallel at high speed. DisArray uses an array of 16 x 16 special-purpose, single-bit processing elements and is in fact a reasonably general purpose SIMD machine, similar in some ways to the A.M.T. (previously I.C.L.) Distributed Array Processor (Reddaway, 1973). Due to the general purpose nature of DisArray, it has proved a very effective base for a much wider range of algorithms than that originally proposed. Parallel algorithms have been developed for other vision and graphics tasks, such as polygon rendering (Theoharis and Page, 1987) among others.

We have recently replaced the original control unit of DisArray with a purpose designed controller (Winder, 1986) based on a transputer (INMOS 1). This new controller is somewhat slower than the original micro-programmable (AMD 29116-based) controller but has the significant benefit of having a good software development environment. The transputer is a powerful component in building general-purpose MIMD computing systems. The built-in serial communications links, the clean, mathematically-specified model of communication, and the Occam language (Inmos 2), make it relatively easy to build MIMD systems and to program complex parallel algorithms.

There is a dedicated array of 42 transputers interfaced by one or more Inmos links to the DisArray controller providing the MIMD computational resource and we also have occasional access to a further 40, more powerful transputers. The combination of

DisArray with a transputer network inevitably had to be named the Disputer! Since all of the programmable processors are transputers, the whole system is programmed in Occam 2 and a complete SIMD/MIMD application can thus be a single Occam program. This arrangement allows us to investigate those computationally-intensive applications which can benefit from two parallel computation paradigms, both well-supported in hardware. It seems that the class of such applications may in fact be quite wide, possibly much wider than the vision and graphics areas with which we are currently concerned.

2 Architecture Of The Disputer

2.1 DisArray

The DisArray processor is basically an array of 256 single-bit processing elements (PEs) in a two-dimensional (16 x 16) arrangement. Fig. 1 shows the internal architecture of each PE (Processing Element) of the DisArray processor. Each PE has bi-directional data links to its four nearest neighbour PEs. Row-based and column-based broadcast lines transmit data, addressing and control information to the PEs from the controller. In addition, a video shift register is threaded through all of the PEs, and this supports real-time display of a 512x512, 16-colour bitmap from some part of the distributed array memory. A proposed re-design of the video board will upgrade the screen resolution and allow real-time video input as well.

Each PE has 256k x 1-bit memory, implemented by a single dynamic RAM, giving a total array memory size of 8 Mbyte. This is enough to hold a significant number of images and related data during execution of some of the more data-intensive algorithms and it can even hold a significant number of consecutive images for image sequence analysis. All processors in the array execute the same (globally broadcast) instruction at the same time. The array has a low-level scheduler which arbitrates between requests from the controller for such computational cycles and from the refresh controller for video refresh cycles.

There is a single, sequential controller which generates instructions for the SIMD array. The array instructions are generally of the form :

```
Mem_Plane [addr] := Fn ( Mem_Plane [addr],
                         Register_Plane
                         RowData AND ColumnData )
```

where Fn is an arbitrary three-operand Boolean function The operation takes place between two 256-bit square words, called planes and a further mask plane. The mask plane is generated by ANDing together at each PE a 16-bit value which is broadcast row-wise through the array and another 16-bit value broadcast column-wise. The addr field refers to one of the 256k planes in the distributed array memory. Fig. 2 shows the major modules in the Disputer System.

The micro-instruction to the entire array basically consists of the 8 ALU function bits, the RAM address, the 16-bit row and column data, the neighbour selection and the ram/register file control lines. All of these micro-instruction bits are copied to every processor in the array. Having given a single instruction to the array, the controller and array operations then proceed in parallel. The micro-instruction register for the array, together with various address and data registers for communication are mapped into the address space of the controller. The controller itself is a 20MHz, 32-bit, 1Mbyte RISC machine, based on the T414 transputer. It has 4 20MHz serial line interfaces

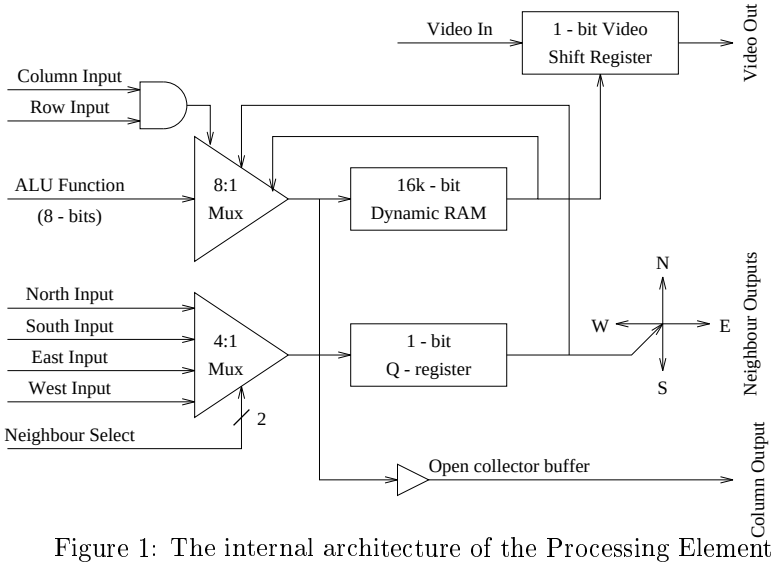

Figure 1: The internal architecture of the Processing Elements

which can be connected to any of the 26 uncommitted ports on the transputer array or the development system.

Due to the provision of a novel addressing scheme, the basic word of the SIMD array can be regarded as a 16 x 16 bit patch of some image, or alternatively, a 256 x 1 bit row (scan-line aligned word) or a 64 x 4-bit pixel row. These two additional linear addressing modes allow us to implement many existing scan-line based graphics and vision algorithms very effectively. The two-dimensional algorithms are typically harder to develop but offer the promise of equally good performance on short, fat objects and on tall, thin objects which is often a drawback with scan-line based algorithms.

2.2 The Transputer Network

The transputer network is a rectangular array of 6 x 7 transputers (Fig. 3, left-hand side), each of which is a 20MHz T414 with no external support chips. The memory for each transputer is limited therefore to the 2 kbytes of on-chip, 50nS static RAM. The memory on each transputer is obviously severely limited but it is surprising what can be crammed into only 2 kbytes when it is all you have! Fortunately, we also have access to a further 40 T414 transputers each with 256 kbytes of fast external RAM and these will eventually be upgraded to T800 (floating point) transputers. The internal array links are hard-wired and the 26 external links can be arbitrarily assigned by a patch panel to link between themselves, the DisArray controller and the development system as dictated by the algorithms being developed. Fig.3 shows a minimal interconnection for the system to operate.

2.3 The Host and Development Systems

There is a 16-bit DMA link between the DisArray controller and a 32-bit, user microprogrammable Unix host processor (an Orion), which runs BSD4.2. A running application on the Disputer can arbitrarily access the Orion's physical memory. This facility is often used to locate a display file, or other applications-oriented data structure, in the address space of a Unix program and then to continuously redraw a picture represented

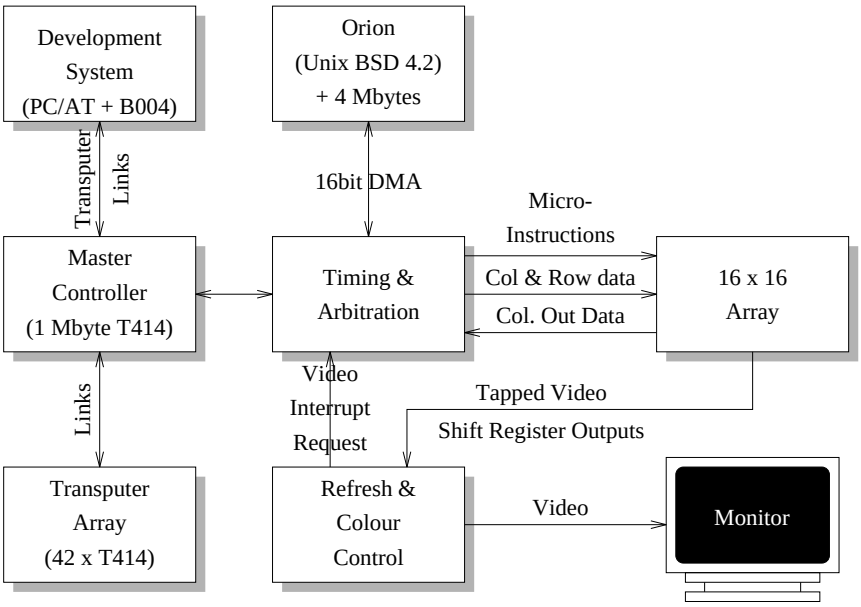

Figure 2: The major modules in the Disputer System

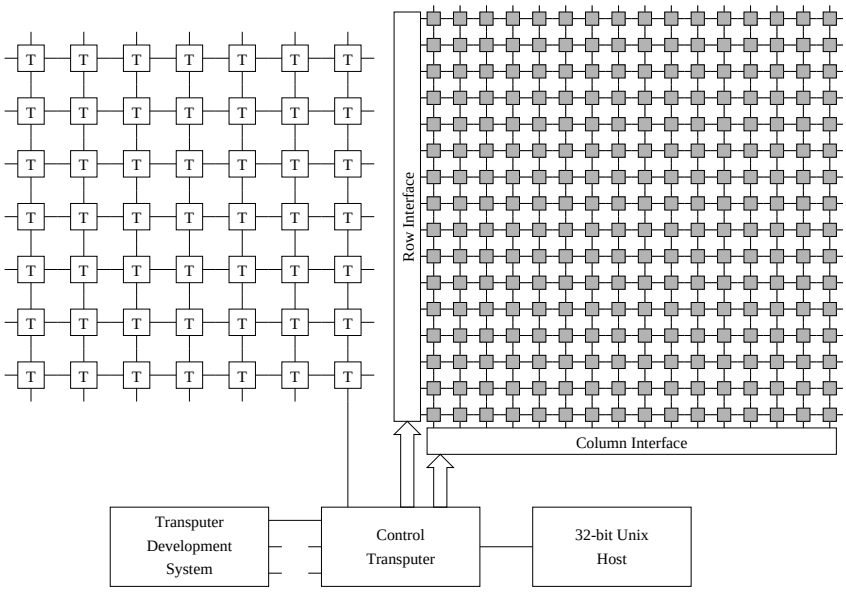

Figure 3: An overview of the Disputer architecture

in the display file by high-level display commands (line, polygon etc.). Note that this achieves real-time animation with no explicit graphics i/o commands being involved in the Unix program. We also use the DMA link to get image data into the Disputer from a Datacube frame grabber/processor which is acessible from the Unix host over an Ethernet.

At present, all of the software for the Disputer is written in Occam 2 and the software development is carried out on a 2Mbyte T414 transputer hosted in an IBM PC/AT. The PC acts as a terminal and filestore to the development transputer and we run the Inmos Transputer Development System (TDS2) software. The development system can also act as a host to the Disputer system, in which case communication is by up to 4 transputer links rather than the DMA link. At some point in the future, we would like to move the development environment from the PC onto a networked Sun workstation.

2.4 An Example of a Disputer Graphics Application

We have programmed a dual-paradigm implementation of a Mandelbrot set browser (Forschungsgruppe Komplexe Dynamik, 1985). This was chosen partly because it produces beautiful pictures from virtually no input of data and partly because it is the only program that we are aware of that seems to have been implemented on every other machine in the world, and 'thus provides a useful benchmark! This application uses a tiny program on the development transputer to provide the co-ordinates of the next window onto the Mandelbrot set which is to be computed. It generates these co-ordinates either from a dialogue with the user or at random.

The window co-ordinates are passed to the Disputer controller transputer, which splits the window space into 1024 work packets' each one corresponding to a half scan-line sub-window. The transputer network is used as a linear pipeline of 42 processors in this application, so the control transputer simply sends the work packets sequentially down the pipeline. Each transputer takes a work packet from the pipeline and evaluates the Mandelbrot set at each point along the half scan-line sub-window specified in the packet. When finished, it sends a run-coded representation of the pixels calculated for that sub-window back to the control transputer and picks up another work packet from the pipeline.

A buffer process which keeps one work packet 'in hand' is essential to ensure that the processors are always kept busy. A problem with this buffering is that towards the end of execution some processors are idle while others still have unevaluated work packets buffered. This problem can be overcome by a number of strategies; unfortunately, they all severely complicate the program. Fig. 4 shows the gross process architecture of the program, which is the now famous processor farm model of parallelism (May, 1986).

The control transputer receives the run-coded result packets and uses the DisArray SIMD processor to render the pixels. Using DisArray's linear addressing mode, two table look-ups and one write operation are all that is required to render up to 64 4-bit pixels in a row. This takes only 3 SIMD instructions (currently about 500ns each). In this application, the 42 transputers are the bottleneck and the fast rendering of DisArray is somewhat superfluous. This was our first dual-paradigm algorithm and it still provides one of our more impressive demonstrations. The performance of the system is such that with an iteration limit of 250 on the Mandelbrot equation, an 'average' 512 x 512 image is computed and displayed in around 3 seconds. The worst-case (completely 'black') image takes about 25 seconds.

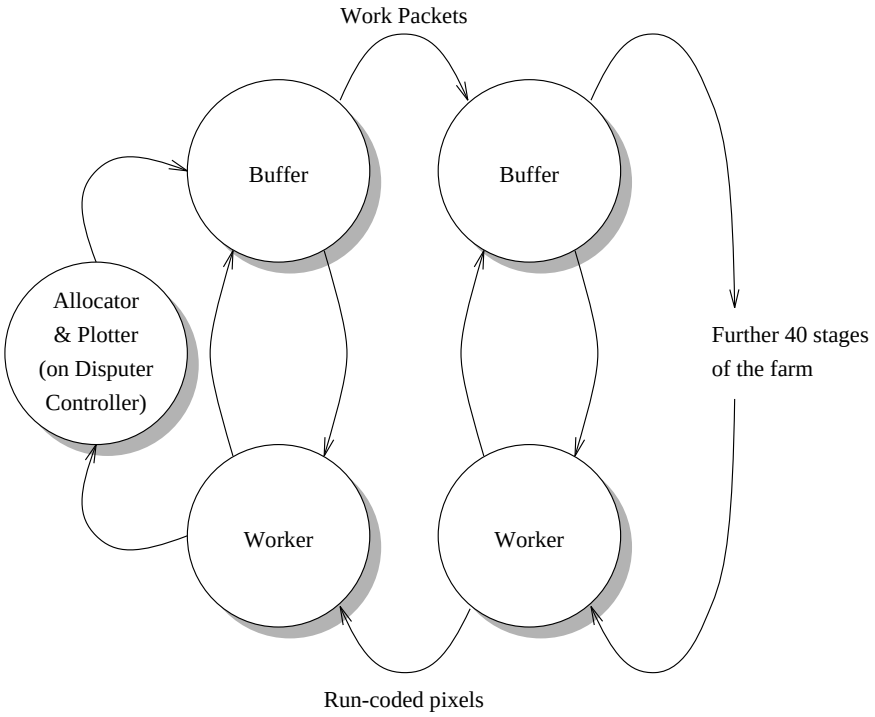

Figure 4: The Mandelbrot Processor Farm

3 Vision Algorithms On The Disputer

3.1 Introduction

At present we are only just starting to mount vision algorithms on the Disputer, which was originally built for computer graphics applications. It has been due to the recent arrival of Prof. Brady at Oxford and the formation of an active vision research group here that our work is now being directed towards vision applications as well. It is abundantly clear that there is a large amount of commonality between some of the key algorithms in graphics and vision and both of our groups have benefitted from the collaboration.

We are currently developing some more interesting software than described here, and this will be be reported later. In particular, we are working towards parallel implementations of the Canny edge finder (Canny, 1985) and the Fleck phantom edge finder (Fleck, 1987) in the SIMD engine and hope soon to have some simple intermediate-level vision demonstrations running in the MIMD engine.

3.2 Gaussian Convolution

We show here a part of a very simple early vision program that we have implemented on the Disputer. This part of the program runs mainly on the SIMD machine and performs a convolution with a 3x3 approximation to a Gaussian kernel. The kernel was chosen for ease of implementation, in that the post-scaling division is by a power of 2. The central limit theorem ensures that only a few applications of this kernel result in a very

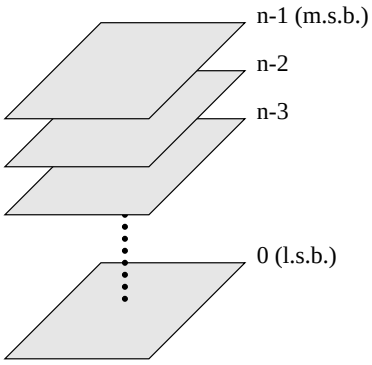

Figure 5: 16x16 columnar numbers

1	3	1
3	16	3
1	3	1

Figure 6: Gaussian kernel

good approximation to convolution with a Gaussian (Burt, 1981).

The convolution uses bit serial arithmetic, performed over sets of 256 n-bit integers. Each set of 256 n-bit numbers is stored as a set of n planes in array memory. Fig. 5 shows n memory planes standing, in a columnar form, on top of plane 0, which contains the 256 least significant bits.

Fig. 6 shows the kernel and Fig. 7 shows how the convolution operation is achieved. A plane-sized set of columnar numbers representing a 16x16 patch of pixels from the input image is multiplied by 16. This operation is made very simple by the choice of the small integers in the kernel, so that multiplication by a power of two involves only an address offset calculation. Eight other sets of 256 pixel values, each offset from the first by one pixel in the eight compass directions respectively, are similarly multiplied by the corresponding constants from the kernel. The multiplication by 3 is of course achieved by multiplying by 1 and 2 separately (both virtually cost-free) and adding the results. Thes products are added bit serially into a running total and placed in the output image, with a final division by 32 to rescale appropriately.

A convolution on a 256 x 256 image involves repeating this operation 256 times to cover the whole image and this takes about 120 ms, which is really quite a long time. The DisArray machine shows its origins here in that, being designed for the RasterOp function only, there is no direct support for arithmetic operations in it. This is clearly a limitation of the current (greatly outdated) hardware, as we have to perform seven

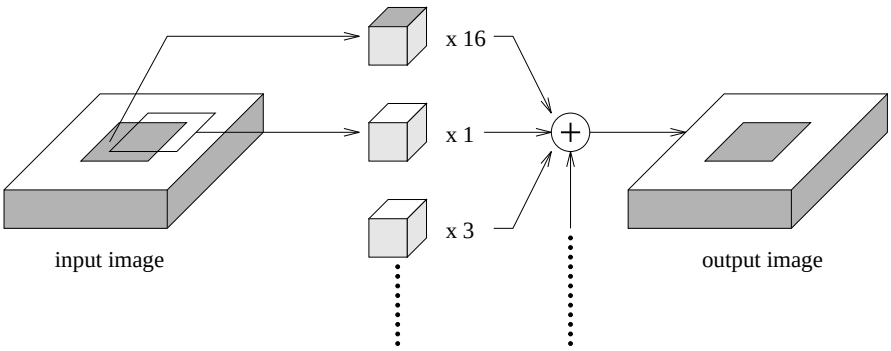

Figure 7: The Gaussian Convolution

SIMD instructions just to add two planes of bits together with carry!

3.3 The Vision/Graphics Virtual Machine

It is one of our aims to develop a single virtual machine for both graphics and vision, which should eventually simplify the writing of graphics and vision applications software somewhat. It will not have a single implementation model, but should be readily supported by parallel hardware of different types. We hope that an additional advantage might be that simply having a well-considered set of parallel kernel operations may itself lead to new vision algorithms, just as the RasterOp operation has led to a wide range of novel algorithms and applications in graphics, and beyond.

RasterOp, extended to the domain of coloured images, will certainly be included in the virtual machine. The MAP-2D function, originally devised by Canny (1985) for his edge-finder, is another interesting contender. MAP-2D was originally implemented in Lisp with micro-code support to provide a two-dimensional analogue of the Lisp map function.

As arguments, MAP-2D takes a list of two-dimensional, m x n arrays and an arbitrary Lisp function; the result it produces is also a list (often singleton) of m x n arrays. The given function is invoked m x n times and it can call for support from anywhere at all in the argument arrays (or anywhere else in the environment of the invokation!). Typically, however, support will be from the near neighbours of the array elements indexed by a particular call. A somewhat restricted form of MAP-2D where support can only be drawn from a fixed (small) neighbourhood will almost certainly be included in the virtual machine. A further-restricted form, where the user-provided function is a polynomial expression, will implement the usual linear convolution operations.

A further feature of Canny's MAP-2D is that some simple transformations on the arrays can be obtained by substituting a user-provided indexing function for any of the arrays in place of the normal one. For instance, interchanging the two indices effects a transpose operation. In our virtual machine, these types of transformations, together with a number of others, are provided as primitive operations. It will be the responsibility of the compiler to optimise the implementation of a program-specified set of transformations and other function applications. The use of a largely functional notation will considerably aid this process.

4 Summary

Early results from the Disputer have been extremely encouraging. We have found that there is a good match between the hardware a a number of interesting graphics and vision algorithms. Although small and potentially quite inexpensive, our system has a SIMD computational bandwidth of about 0.5 Gigabits per second and an aggregate MIMD execution (RISC) instruction rate of over 800 MIPs. There is great scope for increasing the SIMD speed considerably, using perhaps CMOS gate arrays and static RAM technology. A new version of the SIMD processor is currently being designed in collaboration with IBM which may then be prototyped. The system is relatively straightforward to program, since only a single Occam program is constructed to control all of the processors.

We are developing a superset of Occam 2 which will enable us to express algorithms in a single language which uses both computational paradigms effectively. At the moment, we manage the storage and manipulation of planar SIMD objects by the side-effects of Occam 2 assignments into the DisArray instruction and data registers. This is tedious and highly unaesthetic, but it is acceptable in the short term since there is only a relatively small amount of such code and it can usually be hidden away in library procedures. The extensions to Occam 2 will involve integrating planar objects, columnar numbers and suitable arithmetic and logical operators into the language.

Algorithms for computer graphics and those being developed by the machine vision community, have a great deal in common; this is most evident in the data-parallel algorithms. The degree of commonality is such that it warrants the development of a virtual machine for graphics and vision, and we are actively working on this. It should provide an interface between applications programs and the highly parallel hardware which implements the kernel algorithms. It is the intention that the virtual machine is not heavily oriented towards any particular hardware model; rather, it should be implementable in a number of different ways on different types of parallel hardware.

Both the graphics and vision research communities urgently need hands-on access to parallel computing workstations with image input and output facilities, and the Disputer is a prototype of such a machine. Experience shows that some otherwise promising lines of research simply are not followed up, simply because the computing resources available ensure that the algorithms run at an unacceptably low speed. The only foreseeable route ahead for real-time vision is to learn to exploit the massive parallelism that is now being offered by VLSI technology. Luckily, there appears to be a good match between what is becoming available in hardware and what may be needed by the software. As always, the real problem will lie in the understanding of the algorithms and languages.

A Acknowledgements

I particularly want to thank the Science and Engineering Research Council, Inmos Ltd., I.B.M. (UK) Ltd. and International Computers Ltd. for their financial support of this work and Phil Winder for the enormous effort he has recently put into building and commissioning the transputer controller and developing the software library for the Disputer.

B References

- Ballard D. and Brown C.M. Computer Vision, Prentice Hall, 1982.
- Batcher K., "The Design of a Massively Parallel Processor", IEEE Trans. on Computing, C-29, no. 9, 1980.
- Burt F.P. "Fast Filter Transforms for Image Processing", Computer Graphics and Image Processing, pp.20-51, 1981.
- Canny J.F., Finding Edges and Lines in Images, PhD thesis, MIT AI Laboratory, 1985.
- Cripps M.D, Field A.J., Reeve M.J., "The Design and Implementation of ALICE : a Parallel Graph Reduction Machine", in Functional Programming Languages : Tools and Architectures, ed. Esienbach S., Ellis Horwood 1986
- Duff M.J. "CLIP4. A Large Scale Integrated Circuit Array Parallel Processor", 3rd. Joint Int. Conf. on Pattern Recognition, 1976.
- Falk H. "Reaching for the Gigaflop", IEEE Spectrum 13(10), 1976.
- Faux I.D. and Pratt M.J. Computational Geometry for Design and Manufacture, Ellis Horwoood, 1982.
- Forschungsgruppe Komplexe Dynamik, Frontiers of Chaos, University of Bremen, 1985.
- Geman S. and Geman D. "Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images", IEEE Trans. PAMI, PAMI-6, no. 6, 1984.
- Goldberg and Robson. Smalltalk-80. The Language and its Implementation, Addison Wesley, 1983.
- Grimson W.E.L. "Experiments with an Edge-Based Stereo Algorithm", IEEE Trans. PAMI, 1985.
- Grimson W.E.L. From Images to Surfaces, MIT Press, 1981.
- Hildreth E.C. The Measurement of Visual Motion, MIT Press, 1985.
- Hillis, W.D. The Connection Machine, MIT Press, 1985.
- Hockney R.W. and Jesshope C.R. Parallel Computers, Adam Hilger, 1981.
- Horn B.K.P. and Schunk B.G. "Determining Optical Flow", in Computer Vision, ed. Brady J.M., North Holland, 1981.
- Hwang K. and Briggs F.A. Computer Architecture and Parallel Processing, Mc-Graw Hill, 1984
- Ikeuchi K and Horn B.K.P. "Numerical Shape from Shading and Occluding Boundaries", in Computer Vision, ed. Brady J.M., North Holland, 1981.
- Inmos 1, Transputer Product Definition (various documents), Inmos Ltd.
- Inmos 2, Occam2 Product Definition. Inmos Ltd.

- Marr and Poggio T. "Stereo", in Computer Vision, eds. Ballard D.H. and Brown C.M., Prentice-Hall, 1982.
- May D. Communicating Process Computers, Technical Note 27, Inmos Ltd., 1986.
- Newman W.M. and Sproull R.F. Principles of Interactive Computer Graphics, 2nd. ed., McGraw Hill, 1979.
- Page I. "DisArray : A Graphics-Oriented Fifth Generation Workstation", Proc. Nicograph '83, Tokyo, 1983.
- Reddaway S.F. "DAP - A Distributed Array Processor", 1st. Annual Symposium on Computer Architecture, Gainsville, Florida, 1973.
- Rosenfeld A. and Kak A.C. Digital Picture Processing, Academic Press, New York, 1976.
- Shooman W. "Parallel Computing with Vertical Data", Proc. AFIPS Conference, 1960.
- Slotnick D.L., Borck W.C., McReynolds R.C. "The SOLOMON Computer", Proc. AFIPS Conference, 1960.
- Terzopoulos D. Multilevel Reconstruction of Visual Surfaces : Variational Principles and Finite Element Representations, MIT Artificial Intelligence Laboratory, AIM-671, 1982
- Thacker C.P, McCreight E.M., Lampson B.W., Sproull R.F., and Boggs D.R. Alto: A Personal Computer, Xerox Palo Alto Research Center, Palo Alto, California.
- Theoharis T. and Page I. "Parallel Polygon Rendering with Pre-Computed Surface Patches", Proc. Eurographics '87, 1987.
- Theoharis T. and Page I. "Parallel Incremental Polygon Rendering on a SIMD Processor Array", Proc. Int. Conf. Parallel Processing for Computer Vision and Display, Leeds, 1988.
- Ullman S., The Interpretation of Visual Motion, MIT Press, 1979
- Winder P. Transputer Upgrade of the DisArray Graphics Processor, MSc thesis, Programming Research Group, Oxford, 1986.