IAPR Workshop on CV - Special Hardware and Industrial Applications OCT.12-14, 1988, Tokyo

SOME CONSIDERATIONS ON THE LIMITATIONS OF IMAGE PROCESSING COMPUTER ARCHITECTURES

Michael J. B. Duff

Department of Physics and Astronomy
University College London
Gower Street, London WC1E 6BT

ABSTRACT

The problems encountered in low-level processing of images, in which the output is another image, have mostly been solved, in that a good selection of algorithms exist and processors have been optimised to implement them. The situation is far less well understood at the next level of processing, where neither algorithms nor architectures have been optimised. In this paper, the general nature of intermediate-level processing is explored and guidelines discussed to aid the design of efficient processors.

INTRODUCTION

Image processing is a loosely defined concept and can mean different things to different people. In this paper, image processing is taken to comprise all those operations on images and on data derived from images that are necessary in order to interpret, enhance or restore images. Again, for the purposes of this paper, images are digitised optical images, usually on a square grid, such that all the relevant information in the image can be represented by an N by N array of binary numbers, each with B bits, where N ranges typically between 32 and 1024 and B between 1 and 8 (more for colour images). Image processing operations divide conveniently into three classes: low-level, intermediate-level and high-level [1]. This partitioning is convenient since it is both conceptually meaningful and technically significant. The low-level processing is concerned with taking in image data as an N by N array of pixels, abstracting information from the data (usually local information in the sense that each part of the abstracted data is derived from a group of pixels which are all close to each other) and outputting the result in the form of another N by N array of pixels. In the output array, each pixel has a value which is a measure of a local image property, such as edge strength, edge direction, texture type, and so on. The data can be described as still being in an *iconic* form.

It therefore follows that intermediate-level processing must take in iconic data but the output data is definitely non-iconic, i.e. it is symbolic. The data may take the form of lists, graphs, histograms or even two-dimensional arrays which are not N by N (that is to say, not the same size as the original image). Data elements can no longer have the same geometrical or topological relationships to the image pixels and, in fact, individual data points may represent information derived from pixels which are widely separated in the iconic representations.

The last stage, high-level processing, involves semantic analysis. The symbolic data is interpreted and *meaning* extracted from the data. This stage must be prepared to deal efficiently with input data in a variety of formats and will usually also be expected to interact with a knowledge database. Its output may require a good human interface or, perhaps, may involve controlling a mechanical device. It is the level on which the demands are least well defined and it must therefore be the most flexible.

This classification scheme can obviously be criticised for being over-simplified but it does provide a useful framework for discussing the desirable characteristics of image processor architectures optimised for certain types of processing. A more complete scheme would address the question of classifying systems which pass data and control in both directions through the three levels so that, for example, partial interpretation of the image can be used to control the course of the low-level feature extraction. Again, the scheme does not deal well with systems which use knowledge to aid image transformation. Nevertheless, dividing the total processing into the three broad stages covers the main aspects of the majority of systems which have been proposed and is therefore a good starting point for further analysis and a suitable basis for later refinement of the arguments.

The purpose of this paper is to try to identify the factors limiting processor performance and to relate these factors to the characteristics of the three levels of processing. The following sections treat each level in turn.

Low-Level Processing

This level, characterised by the phrase 'image in, image out', is required to create a new image in which each pixel has a value derived from a neighbourhood of input pixels centred around the array position of the output pixel. At its simplest, the neighbourhood is *local* in that the pixels in the neighbourhood are all adjacent to the input pixel corresponding exactly to the output pixel.

We can write:

$$g^{*}_{x,y} = F\{g_{x+j,y+k} \mid j,k = +1,0,-1\}$$

where F represents any function of the grey-levels g of the set of pixels in the 3 by 3 neighbourhood centred on the selected pixel at (x,y). For more complex operations, the neighbourhood may have to be enlarged and, in the limit, may embrace the entire input image. Even in this extreme case, it is always possible to decompose every operation into a finite sequence of local neighbourhood operations. Additionally, somewhere in the system, it is necessary to incorporate a test which checks for empty arrays of (one-bit) data and a counter which counts the one-bits in a binary array. Generally speaking, the large majority of useful low-level operations are confined to small neighbourhoods, presumably because, at the pixel level, the relevance of the information in one pixel to that in another decreases as the pixels become further apart in an image. Thus nearby pixels contribute to edge information in that grey-level gradients are measured by differencing adjacent pixel values; furthermore, edge elements are formed by clustering neighbouring pixels in regions of high density gradient. Arguments of this type suggest that, for low-level processing, communication between processing elements (PEs) and pixels should be facilitated in multiprocessor systems and that good communication paths should be available between PEs assigned to neighbouring pixels.

This philosophy has been implemented to the full in most two-dimensional mesh array processors [2,3,4] in which a PE is assigned to each pixel and all PEs connect with adjacent PEs in a 3 by 3 local neighbourhood (either with 4- or 8- connectivity). Good pixel to PE communication is usually only achieved once the data has been locally stored in the PEs, mainly because neither optical sensors nor framestores have pixel-parallel outputs. Alternatively, many systems have been constructed since the earliest days of image processing [5] until the present day which use a pipeline system to give a single, powerful processor parallel access to 3 by 3 neighbourhoods of pixel data; as pixels stream through the pipeline, in image raster-scan order, the effect is as if the 3 by 3 neighbourhood raster-scans though the image, producing an output stream of transformed pixels in step with the incoming pixel stream. Systems such as these are particularly useful when only simple processing is required such that the data input and output times are comparable with the processing time.

The essential qualities of the low-level processor therefore depend on the requirement that the PEs should all efficiently process data from small regions of the image and should output into single pixels. The number of communication channels is an order of magnitude larger than the number of processors (or number of pixels in the image). The high degree of parallelism possible at this level is intrinsic in the problem at this level, i.e. the same operation usually can be performed simultaneously on every pixel. In one sense, there is no data reduction at this stage of the processing since the output data consists of the same number of pixels as the input; in fact, since there may be several similar outputs representing image properties, including the input data, which will usually be retained, it could be argued that there has been a data expansion rather than a data reduction.

Intermediate-Level Processing

This stage of processing will be required to take in one or more N by N arrays of pixel data and then severely to reduce the data to either a much smaller array, a list, a graph or perhaps to a mixture of different data structures which are all in some way descriptive of the image. Whilst the input data will still be positionally related to the image in that each data point will correspond to an image property in the region of that point, the same will not be true of the output data which, in general, will carry global information about the image.

It is not difficult to specify the required properties of this stage in relation to the input data structure. As in the lower level, there is a need to enter every data point into a PE in the intermediate level. Since the data will be stored in each low-level PE in a two-dimensional array of PEs, this communication could be highly parallel; the situation is obviously less satisfactory if a pipeline processor has been used for the low-level processing. This can be taken to imply that the output data is again stored in an image buffer (such as a framestore) which will only permit serial access. However, the problem which next arises is that the operations to be carried out are likely to involve data from much larger areas of the array, perhaps clustering edge pixels into whole object edges or exploring the relative positions of object features (e.g. corners or edge intersections). To do this, PEs must now be able to communicate freely with more distant PEs and also be able to output data in structures which are not necessarily matched to the relative positions of the PEs themselves.

So far, the usual approach to this stage of processing has been to assemble PEs along a high-speed bus, which then carries all the communication traffic. The bus provides a non-parallel 'all-to-all' linkage between PEs which is efficient only when the amount of computation in each PE is sufficiently large to require computation times long compared with inter-PE communication times. If this is not the case, then bus contention soon forces an increasing fraction of the PEs to stand idle, waiting for further data. Alternative schemes, such as a hypercube network, eliminate some of the difficulties but will, ultimately, still prove inadequate when a particular algorithm involves large data transfers.

Fundamentally, the major obstacle to optimisation at this level of processing is that the science of image analysis has not yet advanced to the point where the relative frequency of occurrence of all the various algorithms which have been developed is known; indeed, it can be hoped that there are many valuable algorithms yet to be discovered. This is in contrast to low-level processing which is both more highly developed and theoretically better founded and is consequently susceptible to efficient matching with processor architectures.

High-Level Processing

The high-level processor in this threefold system must almost certainly be a general purpose computer, although future studies may suggest the use of architectures specialised for reasoning and running AI languages such as Prolog or perhaps processors for interrogating databases. At the present state-of-the-art, it is far from clear what is required and no worthwhile specialisation has been justified. Whilst researchers are still in this state of uncertainty, it is clear that the best policy is to aim at maximum flexibility and programmability, placing the need for a good user interface above that of obtaining high performance.

THE MIDDLE LAYER

Since the characteristics of the low-level architecture are reasonably well defined (offering a choice between at least two acceptable alternatives) and the high-level processor is almost certainly required to be a general purpose computer, attention can be focussed on the intermediate level. Ideally, the best approach would be to consider the most commonly used algorithms, analysing their computational and data communication requirements so that an appropriate configuration of PEs might be proposed. This configuration could then be benchmarked against the algorithms used in the analysis and against a weighted mix of these together with the less common algorithms. Unfortunately, as was stated above, the information needed to carry out this exercise is not available, except in very broad terms. In the absence of precise information, there is some justification for using methods which are non-rigorous in an attempt to find a workable solution.

The starting points for the discussion are the assumptions that the low-level layer will comprise N by N PEs and the high-level layer one PE only. With these assumptions, the first question to be answered is: how many PEs should be used to form the intermediate-level layer? The connections to the layer and within the layer are a separate issue which will be considered later (although the optimum connection strategy would obviously depend on the number of PEs employed and could affect that number if cost constraints are to be taken into account). In the interest of efficiency, it will be assumed that the number of PEs will be the minimum necessary to give the highest available performance. It is worth noting at this point that adding further PEs can reduce overall performance in that communication to or from idle PEs in part of a system will, as it were, 'distract' the working PEs and thus reduce their capacity for useful work. A less anthropomorphic description of this effect would be to point to the communication bandwidth limitation present in all practicable systems and to note that even idle PEs (i.e. PEs into which no part of a decomposed problem can be directed) will usually be polling other PEs in an attempt to find tasks to perform. It is proposed to examine the hypothesis that the optimum number of PEs in the intermediate-level layer is the *geometric mean* of the numbers on either side of the layer, i.e. N.

SUPPORTING ARGUMENTS FOR THE GEOMETRIC MEAN HYPOTHESIS

None of the following arguments is to be seen as conclusive in itself but rather as a contribution to a body of evidence pointing to the same general conclusion. A critic of this approach might reasonably say that five spurious pieces of information cannot add up to one indisputable fact. In reply, it could be said that there would seem to be no reason to claim that the arguments presented are *wrong*; they are merely *weak*. They should, perhaps, be taken as a challenge to the reader to find more conclusive evidence, either one way or the other.

Mathematical Sequence

In the absence of a well defined role for the intermediate-level processor, insofar as the range of algorithms to be considered is large in number and diverse in type, and assuming that general considerations can be taken to imply that the optimum number of PEs will lie somewhere between N^2 and 1, then it seems likely that the number sequence: N^2, M, 1 (where M is the number of PEs we are trying to determine) will be one which occurs elsewhere in physical systems. Typical candidates would be $(N^2 + 1)/2$ (the average), N (the geometric mean), or, again, N (the value to which N^2 would decay exponentially in half the time it would take to decay to 1). Thus choosing M to be N is in accordance with two of these examples.

Impedance Matching

In a three-stage system in which the output impedance of the first stage is Z_1 and the input impedance of the third stage is Z_3, if it is required to link the two stages by means of a middle stage whose input and output impedances are both Z_2, then choosing Z_2 to be the geometric mean of Z_1 and Z_3 maximises the power delivered into the output. Can this concept of optimal matching be extended to the information flow in a three-layer information processing system?

Communication of Data

If for reasons of bandwidth limitation or because of the processing capability of a PE, each PE can deal with a maximum of K inputs from the layer below, then the single PE in the top layer will expect to receive data from K PEs in the intermediate level. Similarly, each of these will serve K PEs in the lowest level, from which it can be deduced that K is equal to N.

Reduction of Parallelism

As has been discussed before, the intrinsic parallelism in the problem is reduced by the processing from 'pixel-parallelism' at the low level, through 'symbol-parallelism' at the intermediate level, to 'unit-parallelism' (or a serial problem) at the high level. Load balancing suggests that the parallelism should be reduced equally at

each stage which, in turn, implies the number of PEs should be reduced by the same factor between each level and that the factor is N, once again supporting the hypothesis.

Row Processing

Although not strictly supporting the hypothesis but, nevertheless, adding weight to the conclusion that there should be N processors in the intermediate level, it is worth noting that the intermediate-level processor could very well be structured as a row processor, with N elements in the row. It can be argued that the low level processes two-dimensional data optimally and the high level processes zero-dimensional data optimally; there would seem to be some logic in constructing an intermediate-level processor optimised for one-dimensional data.

Thus five hints can be found to help the computer architect to make a 'best guess' as to how many PEs to include in the intermediate-level layer. The indication is that N PEs would provide about the right amount of computing power. The way in which they are connected together and how they communicate with the other layers needs further thought.

CONNECTION STRUCTURES

In one sense, all communication schemes in multiple PE processors are a compromise between what would be maximally efficient and what can be afforded. Ideally, every PE would have a direct communication link with every other PE in the system, although, it must be admitted, the programming and control of such a system would present enormous difficulties. An acceptable compromise usually amounts to providing short paths between PEs which often need to communicate and longer paths (i.e. via several other PEs *en route*) between all other pairs of PEs. In practice, this policy often implies direct paths between adjacent PEs in the low-level processor, the connection structure forming a two-dimensional mesh with toroidal connections linking the edges. Enhancements of this connection scheme, such as the pyramid [6] or a hypercube network, as in the Connection Machine [7], represent attempts to extend the scope of the low-level processor into the domain of intermediate-level processing. Their value in low-level processing is to be questioned since the additional paths have only a small bandwidth and are inefficient when, as is often the case in low-level processing, large quantities of pixel data are to be moved around in the array (e.g. in performing affine transformations of the image).

Although the intermediate layer contains fewer (N) PEs, the N^2 connections that would give *all-to-all* connectivity would still be prohibitively expensive, especially when the design of the PEs necessary to deal with the data flow is taken into account. Even when hard-wired connections are provided between pairs of PEs, the data entering a PE must be stored in separate registers to await processing and, perhaps, subsequent accessing via a multiplexer. This accessing process can easily be the bottleneck in an operation, rather than the use of shared data paths outside the PE itself.

The massive parallelism in the low-level algorithms is less likely to occur at the intermediate level. As an example, consider the analysis of a plane, straight-sided figure. At the low level, ends and vertices would be identified and pixels corresponding with them would bear an appropriate label in the output array. The role of the intermediate layer might then be to establish the relative orientations of these labelled features and to trace their connectivity. An output from that layer could be a list of the form:

$$\mathrm{END}_1(\mathrm{VERTEX}_2);\ \mathrm{BELOW}(\mathrm{END}_2, \mathrm{VERTEX}_{1,2,3})$$

$$\mathrm{END}_2(\mathrm{VERTEX}_3);\ \mathrm{ABOVE}(\mathrm{END}_1, \mathrm{VERTEX}_{1,2,3})$$

$$\mathrm{VERTEX}_2(\mathrm{END}_1, \mathrm{END}_3);\ \mathrm{ABOVE}(\mathrm{END}_{1,3}), \mathrm{BELOW}(\mathrm{VERTEX}_1)$$

etc.

where the features listed within the brackets are those connected by lines in the image to the feature named in front of the bracket and where the positional relationships are as stated. Although the actual tracing of the connections would probably be carried out by label propagation in the low-level processor, the list construction would be assigned to the intermediate level and would require PEs to visit regions of the iconic data to abstract the necessary information. An obvious need would be for independent control of the intermediate-level PEs, as it would be difficult or inefficient, or both, to force their operation into an SIMD mode; the lists are of differing lengths and the tasks of constructing the lists differ both in type and complexity. Thus whilst it would be a design aim that every PE would be fully occupied for most of the available time, this could only be achieved if all were allowed simultaneously to execute different programs.

This short example does illustrate an important point: as the computation proceeds and the parallelism decreases in the problem, the decrease is partly in the number of data items to be processed and partly in the diversity of the processing required on the reduced data. The first point leads to a smaller number of PEs but the second point indicates increasing independence in the control of the PEs. One possible configuration for the intermediate layer would be a row processor with semi-autonomy in each PE; semi-autonomy here implies that instructions are broadcast to all PEs, as in an SIMD machine, but that each PE can interpret the instruction according to data stored within the PE itself [8]. An alternative mode of operation is provided by the bus-orientated systems, such as POLYP [9] in which a large number of microprocessors are connected by means of a high speed bus and share tasks between themselves by a process known as 'spawning'. Each PE stores the entire code for all the tasks and the appropriate segment of the code is executed as the need arises. These two examples represent the extremes of local autonomy and the choice between them would clearly be task dependent. In both cases, care must be taken to avoid a communication block along the

line of processors. CLIP7 has direct links between immediate neighbours and operates in a shift register mode; POLYP has the capability of spreading communication between several parallel buses (the POLYBUS).

Since it is to be expected that the intermediate-level processing will not involve moving large quantities of data, a hypercube network offers some attractive advantages and, indeed, even if the PEs are conceptually in a linear array, the linear nearest neighbour connections form a subset of the complete hypercube set. It will be remembered that a hypercube network is arranged so that if all the N PEs are given unique binary labels (with log N binary bits in each label), then there are hard-wired connections between all pairs of PEs whose labels differ by only one bit. In order to be consistent with the linear array architecture, PEs in the array must be labelled such that the sequence of labels down the array follows a Gray code (i.e. a code in which consecutive values differ by only one bit at a time). It follows that no PE-to-PE path need exceed log N steps. Each PE in the row processor must have access to a column in the low-level array of PEs and is therefore required to deal, either in parallel or serially, with N inputs, in addition to the log N inputs from the other PEs in the same level as itself.

It is important to note that the proposed intermediate-level processor should not be confused in structure or purpose with the hypercube network which overlies the two-dimensional mesh in the Connection Machine. The hypercube connections there are intended mainly as a means of enhancing low-level communications and do not link with further PEs. Nevertheless, confusion of purpose is inevitable in that even the low-level processor is a general purpose machine, being able to carry out the entire image analysis task itself, albeit inefficiently. Low-level processors are usually hosted by conventional, single-processor computers and the combined system can therefore be regarded as a three-layer machine with its middle layer collapsed partly into the lower layer and partly into the host. It is equally a matter of choice as to how the workload is to be divided between the low-level processor and the host and enhancements of the low level will often contribute to its capability to handle intermediate-level processing.

CONCLUSIONS

Although the tasks to be performed by an intermediate-level image processor can be conveniently described as *iconic-to-symbolic* transformations, the wide variety of output data structures involved prevents a complete analysis of the relevant algorithms; furthermore, this level of processing is still not well understood so that it must be expected that there are many more useful algorithms yet to be discovered. It is this lack of precise information which prevents the systematic design optimisation of intermediate-level processors and which prompts a more heuristic approach to the problem. In this paper, guidelines have been identified which suggest that the intermediate layer should comprise N PEs (assuming the image has N^2 pixels), connected as a linear array and with additional connections to complete a hypercube network. The PEs themselves should be more powerful than the low-level PEs and, ideally, should be capable of running their own programs. This proposal will now be studied in emulation and benchmarked against existing intermediate-level algorithms.

REFERENCES

1. S. L. Tanimoto, Architectural issues for intermediate-level vision, *Intermediate-Level Image Processing*, ed. M. J. B. Duff, pp. 3-17. Academic Press, London (1986).

2. M. J. B. Duff, Review of the CLIP image processing system, *Proc. Nat. Comp. Conf.*, 1055-1060 (1978).

3. S. F. Reddaway, The DAP approach, Infotech State of the Art Report on Supercomputers, Infotech Ltd., Maidenhead (1979).

4. K. E. Batcher, Design of a massively parallel processor, *IEEE Trans. C-29*, 836-840 (1980).

5. K. Preston, Jr., Feature extraction by Golay hexagonal pattern transforms, *IEEE Trans. C-20*, 1007-1014 (1971).

6. V. Cantoni and S. Levialdi (eds.), *Pyramidal Systems for Computer Vision*. Springer-Verlag, Berlin (1986).

7. W. D. Hillis, *The Connection Machine*. MIT Press, Cambridge, Mass. (1985).

8. T. J. Fountain, K. N. Matthews, and M. J. B. Duff, The CLIP7A Image Processor, *IEEE Trans. PAMI-10*, 310-319 (1988).

9. W. G. Griswold, P. H. Bartels, R. L. Shoemaker, H. G. Bartels, R. Maenner, and D. Hillman, Multiprocessor computer system for medical image processing, *Intermediate-Level Image Processing*, ed. M. J. B. Duff, pp. 267-286. Academic Press, London (1986).