Def Sci J, Vol 35, No. 3, July 1985, pp 353-359

Implementation of a Prototype Cellular Logic Array Processor

A. MUKHERJEE & Y. V. VENKATESH

Indian Institute of Science, Bangalore-560012

Abstract. A prototype cellular logic array processor (CLAP-4), which has been indigenously constructed using TTL integrated circuits, can process 8 by 4 arrays of 4-bit image data in parallel, i.e. simultaneously, as a consequence of the parallel architecture which enables acceptance of contextual information from the neighbourhood of a pixel. Further CLAP-4 provides 48 arithmetic/logical operations on (up to) three operand images. In this paper, a description of the structure of CLAP-4 is presented.

1. Motivation

Analysis of images by the standard sequential execution of processing programme for feature extraction and classification is computationally demanding due to the large amount of data that has to be processed. When viewed in the light of the fact that the general purpose (sequential) digital computer has an architecture not particularly suited for image processing, and that the different points of the image undergo the same type of processing operations, one is led to an analysis of different architectures to facilitate faster and more efficient processing of image data.

In order to carry out image analysis on a computer, the source image is usually divided into a square array of picture elements (pels) by horizontal and vertical divisions. A binary code is used to represent an attribute of the pel that is of interest. For example, the intensity of a pel in a monochrome picture could be digitized to 256 gray levels. An algorithm is then applied to this data for feature extraction and classification.

A common characteristic of all image data (photographs, MSS tapes, TV camera output etc.) is the enormity of the information that has to be handled. As an example, a high resolution 512 $\times$ 512 television frame digitized with 8-bit resolution constitutes 512 $\times$ 512 $\times$ 8 bits = 2 M bits. A LANDSAT image with 2340 $\times$ 3240 pels of 8 bits each amounts to 60 M bits.

It is now recognised that interactive processing of remotely sensed data is an effective method of developing suitable software for resources estimates. Evidently, in order to achieve interactive image processing capability, the image data has to be processed and the results displayed fast enough to facilitate fatigue-free operator

performance. The required response time is estimated to be 2–5 seconds. In contrast, noise removal from a 200 × 256 × 8 bit picture takes 324 seconds on a time shared DEC 1090.

This clearly demonstrates that there is a need to explore alternative computer architectures that are more suited to image processing. Conceptually, processing an image in such a manner that as many pels as possible are subjected to the desired operations at the same time; i.e. parallel processing, offers an apparent solution to the problem. Since most of the image processing operations are dependent on contextual information, the architecture should allow use of data from the neighbourhood pels in the computation. An architecture of this type, the cellular logic array processor, was initially proposed by Unger[1] and later investigated by Golay[2], Duff[3], Preston[4] and Reeves[5], among others. What follows is a description of a prototype parallel processor that has been designed and is under construction. A comparison of its features with those found in the literature, especially Reeves[5], is also briefly given.

2. Cellular Logic Array Processor Architecture

Cellular logic array processors usually utilize a separate processing element (PE) for each pel of the image, operating in a fixed interconnection single instruction multiple data (SIMD) mode. The processing element is usually a simple bit-serial processing element which accepts data from one or more bit planes and the four or eight pels surrounding the pel it is attached to. Although the operations possible by the processing element are elementary, judicious use of the interconnection properties can give complex feature extraction with simple programmes. Alternative structures proposed include hexagonal tesselation[2] which gives six neighbours, or pipelining of cellular processors[6].

In addition to the processing element, such a processor would also require memory elements which are accessible in parallel to store data and intermediate results. Since it may not be possible to have as many processors as there are pels, extra storage elements for edge data would be required to allow successive applications of a smaller number of processors (a sub-array processor) to a larger number of pels. Provisions would also have to be made to write the data to be processed and to read the results.

The PE architecture used in this prototype CLAP-4 is adapted from that described by Reeves[5]. It offers the following types of instructions for array processing :

(*i*) *Boolean instructions :* A boolean function of the input data A and B is generated and stored in Y (Fig. 1).

(*ii*) *Simple near-neighbour instructions :* One of the operands A or B is output to neighbouring PE's. The near-neighbour function (NNF) generator receives these outputs and generates a function of the inputs from the neighbours enabled by control signals $g^1 \ldots g^8$. A boolean function of the operands A, B and the input from the NNF is generated and stored in Y (Fig. 2).

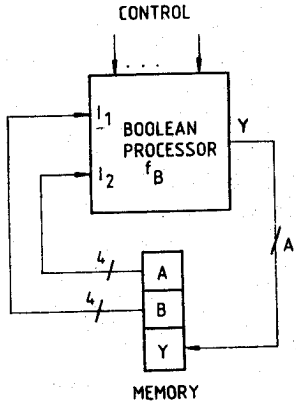

Figure 1. Local Boolean operations.

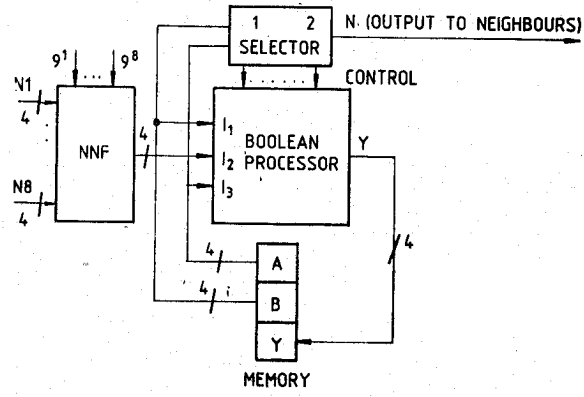

Figure 2. Simple near neighbour operations.

(*iii*) *Recursive near-neighbour instructions :* The output of the boolean function unit is sent to the neighbours. As the NNF uses the data received from the neighbours, which use the propagation output themselves, a recursive computation results (Fig. 3).

The complete PE architecture incorporating dynamic switching between the above operations is shown in Fig. 4.

CLAP-4 differs from Reeves' BASE architecture in two respects. Reeves suggests a boolean function generator that provides all or a subset of all, possible boolean functions of three input data bits. In contrast, CLAP-4 operates on 4-bit data and uses an arithmetic-logic-unit (ALU) that provides only the most useful boolean logic functions, as well as arithmetic capabilities not directly available in BASE. This allows more complicated algorithms to be processed faster.

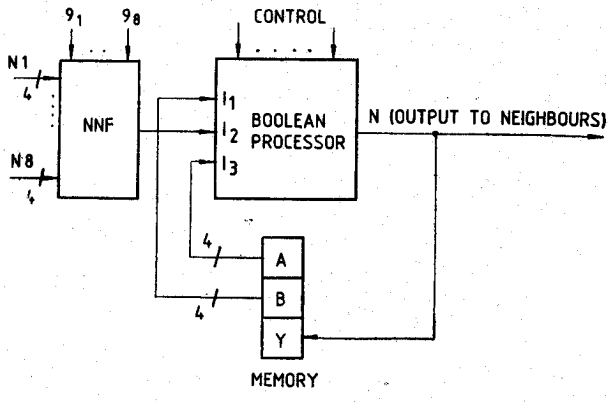

Figure 3. Recursive near neighbour operations.

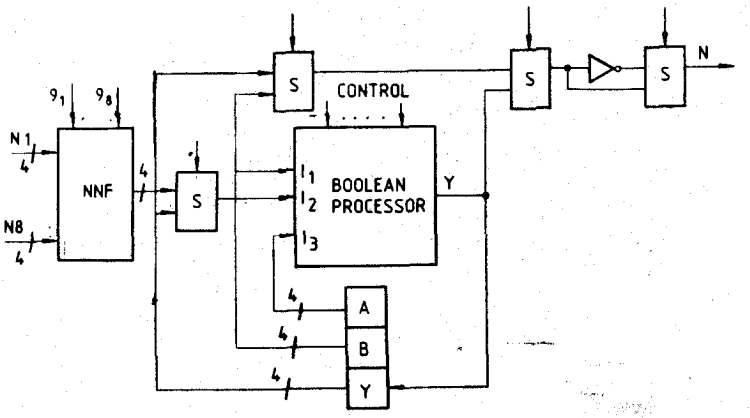

Figure 4. Processing element for all operations.

The near-neighbour function generator in CLAP-4 allows selection of an arbitrary number of propagation directions from the 8-neighbour connection (Fig. 5), thereby making the operation more flexible. To reduce the complexity of the circuit, the propagation output P is generated as the result of

$$P = \bigcup_{i=1}^{8} [N_i \cap \bar{G}_i]$$

where N_i is the i^{th} propagation input and G_i is the i^{th} control bit.

Operands for all the PE's have to be valid simultaneously to allow parallel operation. Ordinary random access memory cannot be used; rather a parallel data register is required. This register should also be able to accept data in a serial fashion to allow initial setting up of the problem. There should also be at least one register that accepts data in parallel from the ALU's and allows sequential access for result read out. This register should allow parallel access for chained computation. These two types of registers—serial-in-parallel-out and parallel-in-parallel-out-serial-out are designated X and Y registers respectively (Figs. 6 and 7). A centralized implementation was chosen in CLAP 4 to maintain simplicity. However, a distributed implementation may be necessary in a larger processor to reduce interconnections between printed circuit boards.

Use of a sub-array processor on a large picture will require successive applications of the sub-array to segments of the image (Fig. 8). To allow propagation of data from PE's at the edge of the PE array to a successive application, a storage element—the edge register—is provided. It may be noted that recursive operations may be impaired in power in this case, since dynamic propagation between sub-array applications is not allowed.

Finally, a control unit has to provide signals to coordinate the operation of the various components that make up CLAP-4. This should be capable of :

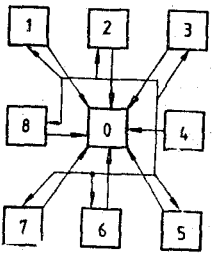

Figure 5. Near neighbour interconnections.

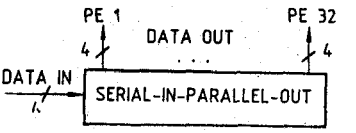

Figure 6. Type *X* register.

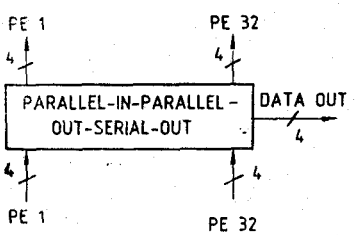

Figure 7. Type *Y* register.

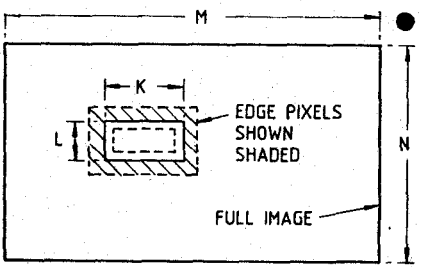

Figure 8. Processing a larger *MXN* picture with a *KXL* array processor.

(i) Data transfer between the host and the array processor,

(ii) Command word decoding to specify mode of operation, ALU OP Codes, NNF control bits etc.,

(iii) Detection of error conditions in the array processor and communication of status to host.

A block diagram of the architectural requirements is given in Fig. 9.

3. CLAP-4 Implementation

A prototype cellular logic array processor for 4-bit data (CLAP-4) has been designed and is under construction. Implementing the architecture described above, it has been designed as an attached processor to an HP 1000 mini-computer. Data and instructions are sent from the mini-computer, and results read back.

An 8 × 4 array of PE's has been designed, using a single 200 × 160 mm printed circuit board (PCB) for each processing element. Arithmetic and logic operations are performed by 74181 ALU's configured for three-operand operations. Neighbourhood data is input to the NNF implemented using random logic on the same PCB.

Two registers of type *X* have been used, designated *A* and *B* registers. They provide 8 × 4 × 4 bits of storage each. Shift registers (74164) are used to implement

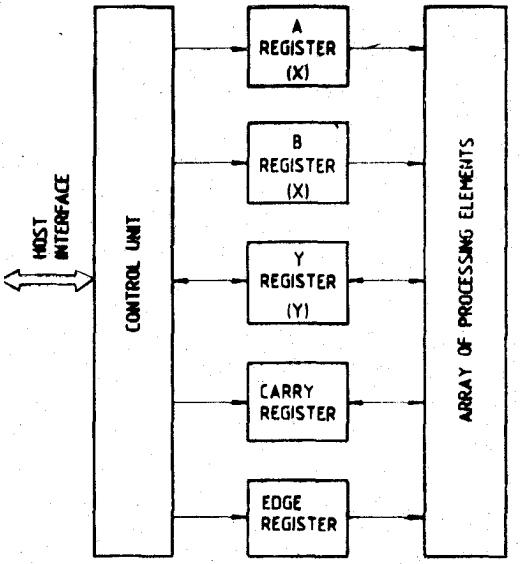

Figure 9. Array processor block diagram.

these serial-in-parallel-out operand registers. Data from the computer is clocked into these registers to set up the operand images.

The result is stored in a type *Y* register which is a parallel-in-parallel-out-serial-out shift register (74198). Loading of this register from the PE outputs is under computer control allowing choice of arbitrary settling time for recursive near-neighbour operations. By means of the serial out capability, data can also be read into the computer. Multi-word arithmetic is supported using a carry register which stores the carry output from the PE's. This is implemented using a type *Y* register with only one bit-plane, i.e., $8 \times 4 \times 1$ bits of storage. Overflow detection logic has also been implemented.

Repeated application of the sub-array to a larger array requires an edge register that can store the data from the 28 pels outside the border of the 8×4 array. This has been implemented using a type *X* register designated as EDGE register. The host computer can access data from the border pels and write them into this register.

A command interface accepts host commands and data, and generates the control signals for the various sub-systems described above. It decodes a 4-bit opcode to derive synchronous control signals using bipolar logic. As it can operate at 10 M bytes/sec, it is proposed to use a microprogrammed interface to the HP 1000 mini-computer. This will allow burst transfer of 256 bits at a time, at a maximum rate of 12 M bytes/sec.

The array processor has been implemented using MSI and SSI TTL logic and occupies three double-height 19″ racks. Individual sub-systems, i.e. registers,

processing elements have been tested and debugged. A wiring list, generated by a computer program, will be used to interconnect the various sub-systems.

4. Further Work

After completing the prototype array processor and interfacing it to the computer, testing is to be undertaken. Under control of the host computer, the prototype will be exercised and the results compared to that of a simulator of the architecture. Once the correctness of the implementation is verified, it is hoped to evaluate the design by implementing image processing algorithms and comparing execution speeds with those obtained on a sequential computer.

Extension of this architecture to larger arrays can be attempted only when it has been fully evaluated and modified as necessary to fulfil its purpose. Apart from the design of the architecture, implementation of a larger array processor will require that the following questions be resolved :

(i) The device technology to be used has to be chosen, depending on power dissipation, propagation speed and cost from among TTL, CMOS etc.

(ii) The level of device integration has to be chosen from among custom, semi-custom (gate-array) or standard MSI/LSI integrated circuits depending on the complexity and size of the architecture to be implemented and the availability of indigenous capability.

(iii) The interconnection scheme to be used, depending on level of device integration, edge connector and PCB connection density limitations and architectural requirements.

(iv) Input-output capabilities to be provided.

(v) How to provide for ease of maintenance and reliability by suitable design.

(vi) Choice of enclosures and mechanical structure to suit power and thermal requirements.

(vii) Software requirements to be fulfilled, including development tools, compilers etc.

Acknowledgement

Financial support from the Indian Space Research Organisation under its RESPOND program is gratefully acknowledged.

References

1. Unger, S. H., *Proc. IRE*, **46** (1958), 1744–1750.
2. Golay, M. J. E., *IEEE Trans. Computers*, **C-18** (1969), 733–740.
3. Duff, M. J. B., *et al.*, *Pattern Recognition*, **5** (1973), 229–247.
4. Preston, K. Jr., *et al.*, *Proc. IEEE*, **67** (1979), 826–856.
5. Reeves, A. P., *IEEE Trans. on Computers*, **C-29** (1980), 278–287.
6. Sternberg, S. R., *IEEE Computer*, **16** (1983), 22–34.