Programming a parallel computer for robot vision

J. L. Armstrong*

Machine Intelligence Research Unit, University of Edinburgh, Hope Park Square, Meadow Lane, Edinburgh EH8 9NW

Work at Edinburgh has directed itself towards the automatic recognition and inspection of objects in an industrial environment using a television camera. A particular need for such systems arises in the context of numerically controlled machine tools. FORTRAN emulators of the CLIP array processor have enabled preliminary tests to be made of a parallel approach to removing noise and extracting primitive features from digitised pictures.

(Received July 1977)

1. Introduction

The National Engineering Laboratory at East Kilbride has expressed interest in the possibility of automatic transfer of partly maintained parts from machine to machine. In small batch production, where the cost of designing and building a special purpose transfer machine for each process is prohibitive, the indicated solution is a programmable 'hand-eye' system capable of being easily re-instructed for each new task of recognition, inspection and manipulation.

Previous work (Ambler *et al.*, 1975; Tsuboi and Inoue, 1976; Rosen *et al.*, 1974) had demonstrated that robot vision systems are feasible, but no complex system has yet been successfully transported from the laboratory to the factory floor. Unsuitability of existing systems for industry is due mainly to the relatively long time taken to preprocess the raw data, i.e. to remove noise and to extract low level information (features such as straight and curved lines, corners and edges, connected regions, etc.) from the two dimensional digitised picture. The serial architecture of the conventional computer is poorly suited to this task. We have begun to look at the use of a parallel array processor.

2. Hardware for picture processing

CLIP (Cellular Logic Image Processor, Duff and Watson, 1974) is a cellular logic array processor which has been developed at University College London especially for image processing. The processor is based upon an array of computer words, having the property that each cell in the array can communicate only with its immediate neighbours. The contents of each cell can be modified according to its neighbour's contents. Modifications take place at all points of the array simultaneously, allowing for extremely fast processing. The array is programmable in that the interconnection pattern between neighbouring cells can be specified by the programmer, as can the relation between the initial and final states of any given cell. For example, suppose we have a simple binary picture (0's and 1's) stored in the array and we wish to remove noise from the picture; this noise we will suppose to consist of isolated 1 cells in the picture. Apply the following process simultaneously at all points in the array:

```
Result: = 1 iff a 1 cell has at least one 1 cell as its im-
                mediate neighbour
            else 0
```

This process can easily be seen to achieve the desired result.

On such a machine the processing time for local processes is independent of the size of the array (the example given would take ≈ 3 μs. on a typical CLIP machine regardless of the number of isolated 1-cells in the picture array). This is in contrast to the conventional machine where processing times increase at least as fast as the square of the array size, often a lot faster. In picture processing, arrays of typically 100×100 cells or larger are used. The array size is dictated by the requirement of having sufficient resolution over the field of view. Simple preprocessing of such a picture such as thresholding or differentiating it, to clean up the data before more complex operations can take place, can therefore be expected to be performed on the CLIP machine at least some ten thousand times quicker than on a conventional machine.

3. Programs for robot vision

Two distinct problems arise in the interpretation of televised pictures. Firstly, simple low level operations must be applied to the raw data at every picture point. These operations are designed to remove spurious or irrelevant information from the picture and to extract other information. Secondly, the picture resulting from the application of such low level processes has to be manipulated and stored in computer memory in such a manner that, in the recognition phase, some description of the current scene being analysed can be compared with stored representations of real world objects in computer memory.

Suppose as an example, we were to digitise on to a 128×128 matrix a perspective view of a cube. We might represent the light intensity at each point on the matrix by an eight-bit number, so the initial data would correspond to some 130,000 bits of information. Now all the relevant information about the cube is contained in the positions of the vertices and in knowing which vertices are connected by edges plus the information that the object belongs to the class 'cube'. All this information could easily be stored using less than 200 bits. Ability to perform such data reduction in a very short time is in our view a precondition for good solutions to the higher level problems of object representation and recognition.

Approaches stemming from early work by Roberts (1965) have attempted to recognise objects by decomposing them into more primitive geometric solids such as cubes and cylinders. This approach, while it can handle some objects with ease, is at a disadvantage with irregular objects. Recent work by Baker (1975) represented objects in the form of 'wire-exoskeletons'. The 'wires' are lines on, or close to the surface of, the object being modelled, and the 'nodes' are points on the surface, usually corresponding to points where the surface curvature changes abruptly. This representation is constructed from the given object by stereoscopic analysis of several views of the scheme from different angles. Such a representation forms a convenient basis for the description of many real world objects. For example, discrimination among a small number of objects may be achieved by looking at the various moments about their centres of gravity.

This simple approach suffices to distinguish one out of a small

*Now at EISCAT Scientific Association, S 98101 Kiruna 1, Sweden

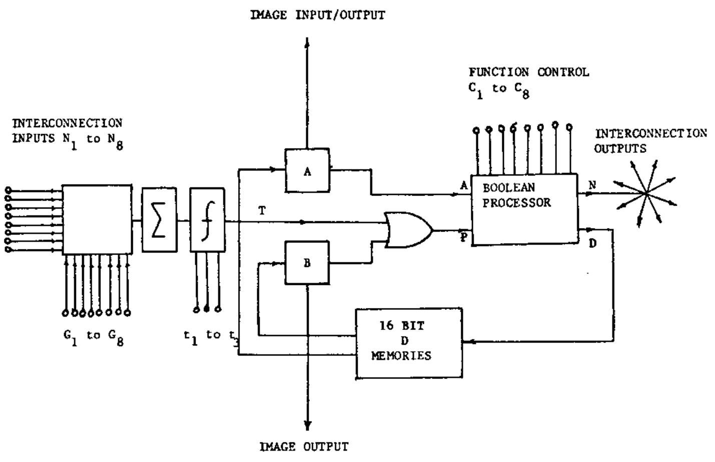

Fig. 1 The basic CLIP cell logic diagram. (reproduced from Duff and Watson, 1974)

class of objects as in the typical industrial environment. For a large universe of possible objects more complex methods would be needed, such as the relational description and graph matching techniques already investigated at Edinburgh (Barrow, Ambler and Burstall, 1972).

4. An integrated approach to picture processing

While the CLIP parallel processor is suited to tasks of an essentially parallel nature, it is not suited to the kind of serial operations required of many of the description building, recognition and high level guidance procedures. We envisage an approach in which the CLIP processor extracts primitive features from a scene, and passes its output to a general purpose machine in which the description building and matching routines reside. Formally, these embody graph manipulation algorithms, which would seem to lend themselves to parallel treatment. But until we know how to do this, we propose in our system to keep them on the sequential machine.

A CLIP operation such as smoothing or differentiating a picture takes some tens of milliseconds; finding a connected object within a picture or removing spurious noise from a picture takes some tens of microseconds. The serial machine in such an integrated system should be responsible for directing operations on the parallel machine. In order to conduct experimentation with the latter we have written two emulators for the DEC System 10 computer. Brief descriptions are given below.

5. Emulators

Firstly, we must briefly describe the CLIP hardware that our programs emulate. **Fig. 1** is a block diagram of the basic CLIP cell. Each cell has 16 single bit storage registers (referred to as the D registers), and two storage registers A and B which can be loaded directly from any one of the D registers and which are used as inputs to a programmable logic gate (the boolean processor). The B register is OR'ed together with some function of the cell's immediate neighbours. This together with the A register forms the input to a two-input, two-output boolean processor. Any logical function of the input variables can be selected using the boolean processor. One of its outputs, the D output, is stored in one of the D memories. The other, the N output, is propagated to the cell's immediate neighbours. In addition, the interconnection lines between adjacent cells in the array can be gated so that a given cell will only communicate with a particular set of its neighbour cells.

Both emulators are written in FORTRAN in the form of a package of FORTRAN subroutines which can be called as required to emulate the various CLIP hardware functions. The majority of CLIP programs are constructed from two basic instructions.

1. *Load instruction*
This has the form:

CALL LOAD (X, Y, Z)

This causes bit A to be loaded from store D_X, bit B from store D_Y and the result of the next process instruction to be stored in D_Z. If either of X or Y is zero, the respective register is cleared.

2. *Process instruction*
This has the form:

CALL PROCE (I1, I2, I3, I4, I5, I6, I7, I8, ITH, BN1, BN2, BN3, BD1, BD2, BD3)

where the variables I1-I8 specify the enabled directions for the array interconnections, ITH is the threshold value at the variable threshold gate, and BN1-BD3 specify the boolean functions of the active logic gate.

LOAD and PROCE are thus the two key FORTRAN routines; they emulate the most important CLIP operations.

Emulator 1 was written to be used with a variable matrix size (1 × 1 to *n* × *n*) and is intended for testing CLIP programs on a small matrix.

Emulator 2 uses a fixed matrix size 36 × 36, it is designed to be fast and efficient. It relies on the DEC System 10's logic functions which operate bit-wise on full words, so that each instruction actually performs 36 logical operations simultaneously.

6. Programming examples

Both the following examples were checked out on emulator 1, and the first example on both emulators.

```
      SUBROUTINE GO
C--REMOVES NOISE FROM D2, SHRUNK PATTERN IN D4
C--FINAL EXPANDED PATTERN IN D5
      DATA AND,OR,XOR,A,P/4,5,6,2,3/
      CALL SETEND(Ø)
      CALL SCAN
      CALL LOAD(2,Ø,4)
      CALL PROCE(1,1,1,1,1,1,1,1,Ø,Ø,-A,Ø,-P,AND,A)
      CALL LOAD(4,Ø,5)
      CALL PROCE(1,1,1,1,1,1,1,1,Ø,Ø,A,Ø,P,OR,A)
      CALL PRINTP(2)
      RETURN
      END
```

(a)

```
................      ................      ................
................      ................      ................
...1111111111...      ................      ...1111111111...
...1111111111...      ....11111111....      ...1111111111...
...1111111111...      ....1......1....      ...1111111111...
...111....111...      ....1......1....      ...111....111...
..1111....11111.      ....1......1....      ...111....111...
...111....111...      ....1......1....      ...111....111...
...1111...111...      ....1......1....      ...111....111...
...1111...111...      ....1......1....      ...111....111...
...111..11111...      ....1......1....      ...111..11111...
...1111111111...      ....1....111....      ...1111111111...
...1111111111...      ....11111111....      ...1111111111...
...1111111111...      ................      ...1111111111...
................      ................      ................
................      ................      ................
```

(b) (c) (d)

Fig. 2 (*a*) **A FORTRAN subroutine to remove spurious noise from a binary pattern. The subroutines SETEND, SCAN and PRINTP are responsible for data input and display**
(*b*) **shows the initial binary pattern with added spurious 1's detail.**
(*c*) **shows the effect of applying the 'shrink' operator to** (*b*)
(*d*) **shows the effect of applying the 'expand' operator to** (*c*). **Note that the small 1's detail of** (*b*) **has now been removed**

```
      SUBROUTINE GO
C--IF AN ELEMENT IN BIT PLANE 3 IS ONE RESULTING PATTERN IN BIT PLANE 4
C--IS COPIED FROM BIT PLANE 1, ELSE IF ZERO FROM BIT PLANE 2
C--USES BIT PLANES 12 AND 13 FOR TEMPORARY WORK SPACE
      DATA AND,OR,XOR,A,P/4,5,6,2,3/
      CALL SCAN
      CALL LOAD(3,1,12)
      CALL PROCE(1,1,1,1,1,1,1,1,Ø,Ø,Ø,Ø,P,AND,A)
      CALL LOAD(3,2,13)
      CALL PROCE(1,1,1,1,1,1,1,1,Ø,Ø,Ø,Ø,P,AND,-A)
      CALL LOAD(12,13,4)
      CALL PROCE(1,1,1,1,1,1,1,1,Ø,Ø,Ø,Ø,P,OR,A)
      CALL PRINTP(99)
      RETURN
      END
```

(a)

```
................
................
......1111......
......1111......
......1111......
......1111......
................
................
................
................
................
..111111111111..
..111111111111..
..111111111111..
................
................
```

(b)

```
................
.......1........
......111.......
.....11111......
....1111111.....
...111111111....
................
................
................
.......1........
......111.......
.....11111......
....1111111.....
.....11111......
......111.......
.......1........
```

(c)

```
................
................
................
................
................
................
................
................
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
1111111111111111
```

(d)

```
................
.......1........
......111.......
.....11111......
....1111111.....
...111111111....
................
................
................
................
................
..111111111111..
..111111111111..
..111111111111..
................
................
```

(e)

Fig. 3 (*a*) **The FORTRAN subroutine of Example 2**
(*b*) **Pattern in bit plane A**
(*c*) **Pattern in bit plane B**
(*d*) **Pattern in bit plane T (Test)**
(*e*) **The resulting pattern (R) of applying the operator: R(I, J) := IF T(I, J) = 1 THEN A(I, J) ELSE B(I, J); to (*b*), (*c*) and (*d*)**

Example 1
In this example we show how simple local operators can be used to remove small 1's detail from a binary picture. The algorithm consists of two steps, 'shrink' and 'expand'. The initial pattern is stored in register D_2, the shrunk pattern in D_4 and the final 'cleaned-up' pattern in D_5.

Firstly, the initial pattern is shrunk. An operator is applied to the pattern that for each cell produces a 1 **iff** the cell itself is a 1 **and** totally surrounded by 1 cells. This is done as follows: each cell propagates to its neighbours the inverse of itself, i.e. a 1 cell propagates a 0 to its neighbours and vice versa ($N = \bar{A}$), thus only cells that are totally surrounded by 1 cells will receive a propagation signal of 0. If this condition is satisfied and if the cell is itself a 1 cell then the result is 1 ($D = \bar{P}$ & A). In this example the 'shrink' operator is emulated with the FORTRAN statement:
CALL PROCE (1, 1, 1, 1, 1, 1, 1, 1, 0, 0, −A, 0, −P, AND, A)

The 'expand' operator sets as a 1-cell any cell that is itself a 1-cell or is next to a 1-cell. **Fig. 2** shows the results of applying the 'shrink' and 'expand' operators to a picture containing spurious 1's detail.

Example 2
CLIP programs have to be written remembering that whatever operation is performed on any particular cell is performed simultaneously at all points of the array. In this example we show how an operation that appears to require doing different things at different points on the array can be reduced to doing the same thing at all points on the array. We wish to copy from bit plane A, or bit plane B into bit plane R (Result) depending upon the value of some flags set in bit plane T (Test). On a serial machine this could be accomplished using some construction of the form:

$$R(I, J) := \text{IF } T(I, J) = 1 \text{ THEN } A(I, J) \text{ ELSE } B(I, J)\ ;$$

where I, J index the array elements.

This could be implemented using some form of conditional branching. We can, however, rewrite this expression in a suitable form for parallel evaluation. For single bits A, B, R & T the above relation can be written as:

$$R := (A \,\&\, T) \wedge (B \,\&\, \overline{T})$$

This form of expression can easily be implemented on the parallel machine (**Fig. 3**).

7. A new programming language

In the previous sections, we have shown how the CLIP computer can be programmed in a simple 'assembly' type language. For complex image processing tasks this is inconvenient and we need a higher level language, whose basic operations correspond to easily understandable picture processing operations. We are designing CLIPTRAN (CLIP Translator) with this in mind. In CLIPTRAN the examples given in Section 6 would be coded.

$$\langle D5\rangle = \text{EXPAND}\ \langle \text{SHRINK}\ \langle D2\rangle\rangle$$

and:

$$\langle R\rangle = \langle\langle A\rangle \text{ AND } \langle T\rangle\rangle \text{ OR } \langle\langle B\rangle \text{ AND } \langle \text{NOT } \langle T\rangle\rangle\rangle$$

Note that in example 1, the intermediate bit plane used in the computation is not explicitly specified. We leave allocation of memory to the system, and in this way can be guaranteed optimal utilisation of the memory.

The early experience that we have gained in writing and using the CLIPTRAN language will prove a valuable aid in the design and implementation of further languages to be used for parallel picture processing. It now appears that the simplest way of specifying such a language is by using some form of macro-processor to translate the high level language directly into CLIP assembly code.

By analysing several programs written both in the CLIP assembly language and CLIPTRAN we have discovered that of the set of all possible language constructs, only a small subset is ever used in writing meaningful programs. This observation led to a theoretical investigation by Jelinek (1977), based on an extension of boolean algebra, of the general properties of CLIP-type machines. It appears possible to design processors that retain the useful instructions that are available on the CLIP machine, but which do so using a somewhat simpler form of processor.

Acknowledgements
I would like to express appreciation to Professor Donald Michie for help and encouragement, to my co-worker Jan Jelinek who wrote the second emulator program, and to the National Engineering Laboratory for financial support.

References

AMBLER, A. P., BARROW, H. G., BROWN, C. M., BURSTALL, R. M. and POPPLESTONE, R. J. (1975). A versatile system for computer controlled assembly, *Artificial Intelligence* 6, pp. 129-156.

BAKER, H. H. (1975). Building models of 3-D objects, *M.Phil. Thesis*, Department of Machine Intelligence, University of Edinburgh.

BARROW, H. G., AMBLER, A. P. and BURSTALL, R. M. (1972). Some techniques for recognising structure in pictures, *Frontiers of Pattern Recognition* (ed. S. Watanabe) Academic Press, New York, pp. 1-29.

DUFF, M. J. B. and WATSON, D. M. (1974). A parallel computer for array processing, *Information Processing 74*, North Holland.

JELINEK, J. (1977). An algebraic theory for parallel processor design, *Research Memorandum, MIP-R-119*, Edinburgh: Machine Intelligence Research Unit.

ROBERTS, L. G. (1965). Machine perception of 3-D objects, in *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, Massachusetts.

ROSEN, C., NITZAN, D., AGIN, G., ANDEEN, G., BERGER, J., ECKERLE, J., GLEASON, G. HILL, J., KREMERS, J., MEYER, B., PARK, W. and SWORD, A. (1974). Exploratory research in advanced automation, Stanford Research Institute.

TSUBOI, Y. and INOUE, T. (1976). Robot assembly system using TV camera, *3rd Conference on Industrial Robot Technology and 6th International Symposium on Industrial Robotics.*

Book reviews

Introduction to Logic and Switching Theory, by N. N. Biswas, 1975; 354 pages. (*Gordon and Breach*, £12·80)

The material in this textbook will be familiar to electrical and computer engineers, although the style of presentation is more formally mathematical than many other books on the subject. The mathematical approach has both merits and disadvantages, but its overall effect is to clarify the presentation to a very acceptable degree. Concepts are introduced and developed using clearly stated definitions, theorems and proofs, on the whole presented at just the right pace for the intended audience (i.e. students and practising engineers). There are just one or two points where it is easy to feel that the mathematics have little practical value and are merely being presented for their own sake. One or two words of explanation could have helped to dispel this atmosphere.

The book is organised as eleven chapters which broadly cover four areas, namely, (i) basic mathematical concepts of Boolean algebra; (ii) Boolean expressions and minimisation; (iii) combinational logic; (iv) sequential logic. Each chapter is terminated with a concise list of references and several exercises. The first three chapters constitute an introduction to the terminology of Boolean algebra and switching networks. The mathematical background (sets, operations, number systems and codes) is presented in Chapter 0. Boolean algebra is discussed in detail in Chapter 1 and in Chapter 2 the reader is shown how electronic gates and electromechanical relays may be represented as such an algebra.

Chapter 3 contains detailed descriptions of Boolean expressions and various notations, including a clear explanation of canonical orfms. Minimisation techniques are covered at some length in Chapter 4. These techniques are studied in the order (i) finding common subcubes on an *n*-cube; (ii) using Veitch-Karnaugh maps; and (iii) using the Quine-McCluskey tabular method. The inclusion of the adjacency method (a modification to the tabular method) in the text seems unwarranted: this could have been introduced in the examples. Chapter 5, on the topic of symmetric functions is rather long and out of place. A shortened version might have been added to Chapter 3.

In Chapter 6 it is shown that all expressions may be generated by NAND or NOR circuits, and several relevant theorems are explained Hellerman's tables of three variable NAND and NOR circuits are also introduced. Chapter 7 discusses threshold logic and is perhaps too advanced for the intended audience: this topic is of specialised research interest at the present. Its presentation at this point also breaks up the book unnecessarily.

The last three chapters deal with sequential circuits. Several types of flipflops are studied in Chapter 8. The design methods presented are sound, but several of the resulting circuits would be difficult to build from readily available components. A word about practical circuits was needed here. The final chapters present too much information in too short a space. Chapter 9 deals with synchronous machines, their description and minimisation, Chapter 10 covers asynchronous machines. Space is short here, and races and hazards in particular are discussed rather briefly.

The points mentioned above are minor criticisms. On the whole I am more than happy with the clarity of presentation of ideas and with the content of the text. There are few errors and a reasonable number of worked examples and exercises are provided.

J. R. GURD (Manchester)

Decomposability: Queueing and Computer System Applications, by P. J. Courtois, 1977; 201 pages. (*Academic Press for ACM*, £12·40)

In the last 20 years analytic modelling of computer systems has advanced considerably through the work of Baskett, Buzen, Chandy, Coffman, Denning, Kleinsock, Kobayashi, Muntz and others. This book adopts somewhat different methods although their work is discussed and related to the results obtained. Given a computer system with R resources and N jobs, the state of the system is determined by the length of the queue for each resource; the number of possible states, n, is the number of R partitions of N objects; it is very large even for small N and R. The equilibrium condition is given by $y = yQ$ where the elements of y represent the probabilities of the various states and Q is the stochastic transition probability matrix. From y we can compute the various performance indicators (response, throughput, average queue lengths, etc.) but the computation of y is difficult because n is large. Fortunately Q is often sparse, the non-zero elements form diagonal and non-diagonal blocks; when the elements of the latter are sufficiently small the system is said to be nearly completely decomposable and it is then possible to study separately interactions within subsystems (diagonal blocks) which determine their short term behaviour, and the interactions between subsystems which determine the long term system behaviour. The non-zero diagonal blocks are often in turn nearly completely decomposable leading to a hierarchical network and a form of aggregation. The first six chapters study in detail this theory and the next three chapters apply it to memory hierarchies, program behaviour and multiprogramming systems. A short final chapter relates this theory to the concept of a hierarchy of levels of abstraction in the design of operating and programming systems. This short book is packed with appendices and references; the reader has to follow some heavy matrix and stochastic theories; but he will find it rewarding and illuminating, e.g. the origins of the methods in Markov economic models.

I. M. KHABAZA (London)