E. PARAPIC user manual

MACHINE INTELLIGENCE RESEARCH UNIT

UNIVERSITY OF EDINBURGH

Subject: Parapic language definition

Version: 2.1

Author: A. Blake

Date: April, 1982

Contents

1 Introduction

PARAPIC is a language for image processing, with two important features. First, it is for processing pictures in parallel; each primitive of the language consists of a sequence of the basic operations that are available in parallel array processors like the CLIP (Duff 1978) and the DAP (Marks 1980). The basic operations of the CLIP have been thoroughly analysed by Jelinek (1979). Secondly, it is a high level language, an extension of POP-2 (Burstall et al. 1971). PARAPIC bears some resemblance to APL (Iverson), in that there are array data-types and that arithmetic operators, applied to array variables, perform array operations. It is, however, more specifically tailored to image processing, and in that sense has design principles similar to those of PIXAL (Levialdi (1981)) and those in (Bruha 1977). PARAPIC has been implemented, running under UNIX† on PDP-11 minicomputers. It drives a parallel array processor emulator, whose design has evolved from the work of Armstrong (1978), Zdrahal and Blake (1980) and Blake and Ruttledge (1980).

There are two components of the PARAPIC extension to POP-2. The image data type is introduced and a number of new operators and functions are added. An image variable is a square array which takes one of three forms:

boolean image - an array whose cells contain truth values

grey image - an array whose cells contain 8-bit signed integers

double image - a double precision grey image, whose cells contain 16 bit signed integers

Of the new operators and functions, some are entirely new (for instance array shift) and some are arithmetic operators or functions, extended to act over arrays (for instance + - * /).

As an example of an arithmetic operation over arrays, the + operator acts as follows:

In $C=A+B$, the array C has components each of which is the sum of corresponding components in A and B The expression above is equivalent, in component form to

$$for\ all\ i,j\ C_{ij}=A_{ij}+B_{ij}$$

†UNIX is a Trademark of Bell Laboratories.

and x - / and other arithmetic operations are similarly defined for grey and double arrays. For boolean arrays there is a set of boolean or logical operations, for instance & (AND) | (OR), and they are defined over array components, similarly to the arithmetic operations. An important operator for arrays, which is not an extended arithmetic operation, is the shift operator |>. In the expression $C=A|>d$, C denotes the array A shifted in direction d. For instance, when d is North,

$$C=A|>(North) \text{ means: for all } ij \; C_{ij} = \begin{matrix} A_{i-1,j} \text{ if } i \geq 1 \\ 0 \text{ otherwise} \end{matrix}$$

Other operations available in PARAPIC include

- parallel array conditional expression; this is simply an extension to arrays of the ? operator, in which $C=B?(X,Y)$ means IF (B) THEN C=X ELSE C=Y. B must be boolean valued and X,Y must, in the array operation, share the same type.

- global measurements over boolean and grey arrays

- image input/output

- start and stop functions for the underlying image processing machine

These operations are specified in more detail in chapter 4.

The PARAPIC language is specified without reference to a particular target parallel array processor. However the choice of operations is constrained by the notion of a parallel array machine. All the operations in the language extension can be succinctly expressed in terms of the basic operations of the CLIP4 and the DAP. PARAPIC provides, therefore, a natural programming environment for the development of image processing programs which make good use of the high processing capacity of parallel array machines.

The PARAPIC language, as specified here, has been implemented by the Author, under the UNIX operating system, driving a software emulator of a parallel array processor. A sample session is shown in appendix A Software based on PARAPIC (Reynolds 1981) has been written which uses the CLIP4 machine although it is not possible, using interpreted POP-2 on the PDP-11 (Clocksin 1979), to drive the CLIP4 at its full speed.

Summary

The PARAPIC image processing language consists of the POP-2 high level language with an extension for image processing. The extension is composed of a data-type - the image - which may be integer or boolean valued, and a substantial set of image operators and functions. PARAPIC embodies the constraints imposed by the structure of the parallel array processor and facilitates the development of image processing programs, within those constraints, and in a high level language environment.

2 Image types and type conversion

Three image types are available: boolean, grey and double precision, which are arrays of truth values, eight bit signed integers and sixteen bit signed integers, respectively. The arrays are square and of a size determined by the underlying parallel array machine or software emulator. In the Unix implementation of PARAPIC (described in the appendix), for instance, a parallel array emulator is started up by the PARAPIC command *

: **start** *(size)*;

where *size* is the length of the sides of the square arrays.

Initialised image are obtained by calling the functions **initb**, **initg** and **initd**. For instance:

initb(**true**) returns a boolean image array of cells set to **true** (1).

initg(0) returns a grey image array of cells set to 0.

initd (-4000) returns a double precision array of cells set to -4000.

Functions are provided for conversion between types. The function **extend** converts a grey image to a double image with each cell having the same value (including sign) as the corresponding cell in the grey image. The functions **upper** and **lower** yield, from a double image, a grey image consisting of the most or least significant bytes, respectively, from the cells of the double image. Conversion from grey or double to boolean is achieved by arithmetic comparisons (> = etc.) described later on. Conversion from boolean to grey or double is done by the **?** operator, also described below.

* The POP-2/PARAPIC prompt is '.'.

3 Image input and output

Several functions and operators are provided for image input and output, both for transfer to and from files on the host machine, and for communication with other image processing processes - these will often be processes for image capture or display. In all cases the result of input is a grey image, and any picture submitted to PARAPIC for input must be the same size as the image arrays. Any required windowing or magnification must be performed by external processes, invoked from PARAPIC, as part of the picture input/output command. Pictures for output are 8 bit grey and of the same dimensions as the PARAPIC image arrays. A grey image may therefore be submitted unchanged for output, whereas a boolean image is first converted to grey (with one grey level representing true and another representing false - system constants), and a double image is converted to grey by the function **upper** before display. †

The input/output operators are:

<< *'filename'* and >> *'filename'*

respectively, where the single quoted string is the name of a file on the host computer Also:

<<|' *command'* and >>|' *command'*

perform input and output respectively, where the single quoted string is a name for a process (this assumes a multi process operating environment on the host machine). Under UNIX for instance, *command* is a shell command which executes a process whose input/output is piped to/from PARAPIC. The view and display operators

<<* and >>*

perform piped input and output of pictures, as described above, but the process called in each case is chosen in advance by executing

: **setview** (' *command* ')

† It is intended, in the future, to adopt a picture file format which allows boolean, double precision and unsigned images, in which case these conversions will be unnecessary.

and

setdisp (' *command* ')

respectively.

4 Image operators and functions

This section describes the image processing operators and functions available in PARAPIC, and the image types to which they may be applied.

4.1 Boolean operators

The unary operator ~ inverts a boolean image - all true cells become false and vice versa. The binary operators **| & ~| |~ ~& &~ @ ==** produce a boolean image, from two boolean image arguments The logical **or and** are performed over all the array cells by the operators **| &** ; the operators **~| |~ ~& &~** also include an inversion, for instance:

$$a|{\sim}b \equiv a|(\sim b)$$

and

$$a{\sim}\&b \equiv (\sim a)\&b$$

The exclusive or operator, @, is defined by:

$$a@b \equiv (a\&(\sim b))|((\sim a)\&b).$$

The equivalence operator, ==, is defined by:

$$a{=}{=}b \equiv \sim(a@b).$$

4.2 Grey and double operators

Operations on grey and double images are either arithmetic, yielding grey or double images, or comparisions, which yield boolean images. For the arithmetic operations with one argument, allowed types are:

```
 grey      -> grey
double   -> double
```

and with two arguments:

(grey,integer)	-> grey
(grey,grey)	-> grey
(integer,grey)	-> grey
(integer,double)	-> double
(double,double)	-> double
(double,integer)	-> double

The unary operator ~, performs ones complementation throughout the grey or double array. The binary operators + * - / perform addition, multiplication, subtraction, signed integer division and remainder. The first four of these operators are already used in POP-2 for arithmetic, so that the allowed types specified above are in addition to the types already allowed in POP-2. The function **abs**(*a*) and its alias **mod**(*a*) - modulus and the functions **max**(*a*, *b*) and **min**(*a*, *b*) - maximum and minimum - have been incorporated. The POP-2 functions **logand**(*a*, *b*), **logor**(*a*, *b*) and **logshift**(*a*, *b*) which perform bitwise **and, or, shift** on integers, are extended for grey and double images Note that the right shift for pictures *does* include sign extension. The POP-2 function **sign**(*a*) has also been extended for grey, double images.

Comparisions

The operators > >= < =< = /= are extended from the POP-2 definition to yield boolean images from grey/double. The allowed types of arguments are the same as for the arithmetic operations, with two arguments, above.

4.3 Shifting and propagation

The array shift is a basic parallel array processor function, arising from the connections between neighbouring cells in the processor array. Thus, shifting an array from the North (Northwards shift means shift *from* the North as in "North wind") each cell takes on the value of the cell immediately to its North. Cells along the Northern border of the array, assume the value 0. The shift operator, |>, in PARAPIC, is applied to a boolean, grey or double image, *p*:

p |> *dir*

in direction *dir*, which must be in the range 1..4. Directions 1,2,3,4 are North, East, South, West. The resulting image has the same type as *p*.

Some operations which can be synthesised from the shift are available in PARAPIC. The expand and shrink operators, ^^and ~^, are defined by:

$$\text{^^}a \equiv a|(a|>1)|(a|>2)|(a|>3)|(a|>4)$$

and

$$/\text{^}a \equiv a\&(a|>1)\&(a|>2)\&(a|>3)\&(a|>4)$$

The propagation function

prop(*a*, *b*, *dirstring*)

propagates a boolean image b, inside a boolean image a, in the directions specified in *dirstring*. We define

dir ::= 1|2|3|4
dirstring ::= ' *dir* *',

The function **prop** may be defined recursively as:

prop(*a*, *b*, *dirstring*) =
    IF there exists dir in dirstring such that
        (b|>dir)&a
    is not everywhere (throughout the array) identical to *a* THEN
        prop(a,(b|>dir)&a,dirstring)
    ELSE
        b

In the case, for instance, that

dirstring = '1234'

the effect of **prop** is to produce a boolean image which contains **true** at any cell that is joined by an unbroken path of **true** cells in *a*, to some true cell in *b*, sometimes referred to as 'labelling' *a* with *b* or as the 'reconstruction' of *b* in *a*.

The function

frame(*dirstring*)

returns a boolean image which contains **true** cells along those array edge specified in dirstring and **false** elsewhere. Thus

frame('1234')

returns an array with **true** cells around all four edges and **false** everywhere else.

Finally the function

ramp(*dir*)

which can also be synthesised using shift, produces a grey image containing a grey level ramp. It has 0's along the edge specified in *dir* and the cell contents increase by one grey level per pixel, towards the opposite edge. For instance:

: ramp(1) -> a;

produces an array whose components are:

$$For\ all\ j\ a_{ij} = i - 1.$$

4.4 Parallel conditional expression

The ternary parallel conditional operator, **?**, combines a boolean image with grey/double arguments to produce a grey/double image. Executing:

: b?(d,e) -> c;

produces an image c which, in component form, is:

$$c_{ij} = \begin{matrix} d_{ij}\ if\ b_{ij}\ (=true) \\ e_{ij}\ otherwise \end{matrix}$$

The argument b must be a boolean image and allowed types for d,e are exactly as for the arguments of a binary arithmetic operator (see section 4.2) and with the same type for the returned value.

4.5 Global array measures

The image operations described so far all produce images as their result. The global array measures described here, however, operate on an image but return boolean (as opposed to boolean image), integer or real results. For instance, the predicates **everywhere**, **somewhere** and **nowhere**, applied to a boolean image, b, return **true** if:

everywhere(b): all cells in b are true

somewhere(b): there exists a true cell in b

nowhere(b): all cells in b are false

and **false** otherwise The function **area** returns the number of true cells in a boolean image. The function **greymax**, **greymin**and **greyav** return the maximum, minimum and average, respectively, of all the cells in a grey or double array.

Acknowledgements

The author wishes to acknowledge the considerable contribution made to the design and implementation of PARAPIC by Hugh Ruttledge, who suggested writing a parallel image processing language, and wrote much of the software for a prototype.

The author acknowledges the contribution of W.Clocksin, in the form of his POP-2 system for Unix.

This research was conducted with the aid of a grant from the SERC to Professor Donald Michie for work on a versatile programmable industrial image processor. The author is also indebted to the University of Edinburgh for the provision of facilities.

References

Armstrong,J.L. (1978). Programming a parallel computer for robot vision. *Computer Journal*, 21, 215-218.

Bruha,I. (1977). The CLIP language, *Research memorandum MIP-R-120*, MIRU, Edinburgh University, Edinburgh.

Blake,A. and Ruttledge,H. (1980). CAP 4 assembler and driver for CLIP4 emulator Machine Intelligence Research Unit, University of Edinburgh.

Burstall,R.M., Collins,J.S. and Popplestone,R.J. (1971). Programming in POP-2. *University Press, Edinburgh.*

Clocksin,W. (1979). The Unix POP-2 system. Documentation, Department of Artificial Intelligence, University of Edinburgh, Edinburgh.

Duff,M.J.B (1978). Review of the CLIP image processing system, *National Computer Conference 1978*, 1011 - 1060.

Iverson,K. (). A Programming Language.

Jelinek (1979). An algebraic theory for parallel processor design. *Computer Journal*, 22 (4), 363-375

Levialdi,S., Maggiolo-Schettini,A., Napoli,M. and Uccella,G. (1981). PIXAL: a high level language for image processing. *Real Time / Parallel Computing*, Onoe,M., Preston,K. and Rosenfeld,A., Plenum Press, New York

Marks,P. (1980). Low-level vision using an array processor *Computer Graphics and Image Processing*, *14*, 281 - 292.

Reynolds,D. (1981). POPX user manual, Image Processing Group, University College, London

Zdrahal,Z. and Blake,A (1980). A simple emulator of a parallel processor: user guide. Machine Intelligence Research Unit, Edinburgh

Appendices

A. PARAPIC under Unix, with an array processor emulator.

To enter PARAPIC, type

parapic <RETURN>

and the usual POP-2 prompt : appears. To start the emulator, working with 64 x 64 arrays.

```
: start(64);
```

The arraysize must be a multiple of 16, between 16 and 128 inclusive. Now try some commands:

```
: ramp(4) -> g;
```

puts a grey level ramp into g.

```
: g >>* ;
```

displays it.

```
: (g >= 16) & (g < 48) -> b;
```

makes a boolean image, b, with a central vertical stripe of **true**, 32 pixels wide.

```
: b >>* ;
```

displays b. Now use b to mask off the parts of g which lie outside the stripe, to produce h.

```
: b?(g,0) -> h;
```

The result can be saved on a file called masked_ramp, by executing:

```
: h >> 'masked_ramp' ;
```

Typing control-Z stops the emulator, and exits from the PARAPIC system, returning to the shell. On a future occasion,

```
parapic

: start(64) ;

: <<'masked_ramp' -> g ;
```

retrieves the masked ramp, for further processing.

B. Summary of operators and functions.

Operator	Precedence	Description
<< <<| <<*	1	Picture input
~	2	Invert / 1's complement
|> ^^ /^	3	shift, expand, shrink
* / %	4	arithmetic
+ -	5	arithmetic
> >= < =< = /=	6	comparisons
| & |~ ~| &~ ~& @ ==	7	boolean operations
?	8	parallel conditional
>> >>| >>*	9	output

Note that precedences follow POP-2 conventions. In an expression the operator with lowest precedence is evaluated first.

Function	description
setview setdisp dump	picture input/output
start stop	image processor control
initb initg initd	image initialisation
logor logand logshift sign abs(mod) max min	arithmetic
upper lower extend	double/grey conversion
prop frame	propagation, boolean frame
ramp	grey ramp
area somewhere nowhere everywhere	global boolean measures
greymax greymin greyav *	global grey measures

* not yet implemented.