MVA'90 IAPR Workshop on Machine Vision Applications Nov. 28-30,1990, Tokyo

ASSOCIATIVE CONTOUR PROCESSING

Avidan J. Akerib and Smil Ruhman

Department of Applied Mathematics & Computer Science
Weizmann Institute of Science
Rehovot 76100 Israel

ABSTRACT

Associative algorithms are presented that can accomplish *curve propagation with thresholding* in **1.5** μs per iteration, *ideal thinning* in **6.4** μs per iteration and *contour tracing and labeling* in **66** μs per iteration, while operating on a 512×512 image. This performance is achieved in a novel architecture designated ARTVM (Associative Real Time Vision Machine) with a modest hardware complement of 64 associative memory chips (and a microprogrammed controller).

INTRODUCTION

The classical approach to associative processing, as described by *Foster [1]*, uses a few primitives to implement all arithmetic and logic functions. These functions operate on all bits of all words of associative memory at the same time. Hence such a system may be regarded as an array of simple processors, one for each word in memory. Ruhman & Scherson (R&S) [2,3,4] introduced shift operations in the responder (tag) register to provide communication between processors, defined the primitive operations by logic equations, and adapted associative memory for modular VLSI implementation. Extending this approach to image processing led to the architecture used in this paper, which is designated **ARTVM** - *Associative Real Time Vision Machine*. In order to perform complex algorithms on 2-d images in real time at video rate, a word or simple processor was assigned to each pixel in the image. Many of the algorithms operate on a neighborhood around each pixel, thereby imposing heavier demands on inter-process communication. The addition of a *long shift* that is a sub-multiple of the image dimensions, satisfies this requirement. Implementation is based on a 4K word $\times$ 152 bit associative memory chip which makes up the bulk of the hardware. To handle a 512×512 pixel image the machine requires 64 such VLSI chips. Application of ARTVM to edge-detection and stereo-vision will appear elsewhere - the contour processing algorithms treated here represent an attempt to extend the work to mid-level vision functions, starting from the primitive curve propagation involved in dual thresholding of edge points, continuing with thinning, and ending with curve tracing, labeling and counting.

INSTRUCTION SET

The machine model is shown on Fig. 1, where the number of memory words is $J = N^2$, and image size is taken to be $N \times N$. The primitive associative operations are given below.

$$\forall j = 0,1,2,\cdots,J-1 \;\; and \;\; \forall k = 0,1,2,\cdots,K-1$$

COMPARE: $T_j \leftarrow T_j \overline{\biguplus_k M_k(A_{jk} \oplus C_k)}$
rsp $\leftarrow \biguplus_j T_j$

WRITE: $A_{jk} \leftarrow \overline{T}_j A_{jk} \uplus T_j (M_k C_k \uplus \overline{M}_k A_{jk})$
READ: $O_j \leftarrow \biguplus_k A_{jk} T_j$
SETAG: $T_j \leftarrow 1$
SHIFTAG(± 1): $T_{j\pm 1} \leftarrow T_j$
SHIFTAG($\pm b$): $T_{j\pm b} \leftarrow T_j$ (long shift)
COUNTAG: $S \leftarrow \sum_j T_j$

FIRSEL: $T_j \leftarrow \begin{cases} 1 & \text{if j is the first ONE in } T \\ 0 & \text{otherwise} \end{cases}$

Let **X** denote any one of the following:
M *(Mask)*, C *(Comparand)*, MC *(Mask & Comparand)*
SET**X**: $X_k \leftarrow 1$
LET**X** *opt1 opt2 opt3*
Where: $opt1 = d(r_1), d(r_2), \cdots, d(r_s)$
$opt2 = dseq(u_1, u_2)$
$opt3 = dvar(v_1, v_2, p)$
and options can be intermingled in any order. Then:

$$X_k \leftarrow \begin{cases} 1 & \forall k = r_1, r_2, r_3, \cdots, r_s & \text{if } opt1 \\ 1 & \forall k = u_1, u_1+1, u_1+2, \cdots, u_2 & \text{if } opt2 \\ p_{k-v_1} & \forall k \in [v_1, v_2] & \text{if } opt3 \\ 0 & & \text{else} \end{cases}$$

where p_i denotes the i-th bit of integer p.

The symbols $\uplus, \oplus$ and $\biguplus$ stand for *or, exclusive-or* and *or-expansion,* respectively. Up to four operations may be done concurrently during a given memory cycle: SETAG or SHIFTAG; loading M (SETX, LETX); loading C (SETX, LETX); and COMPARE, READ or WRITE. An extra half-cycle is required for each additional SHIFTAG and for loading different data into registers M and C (other than all ZERO or all ONE). FIRSEL executes in 6 cycles and COUNTAG in 12. Control functions are given in the C language, and are carried out in parallel with the associative operations, hence do not contribute to the execution time.

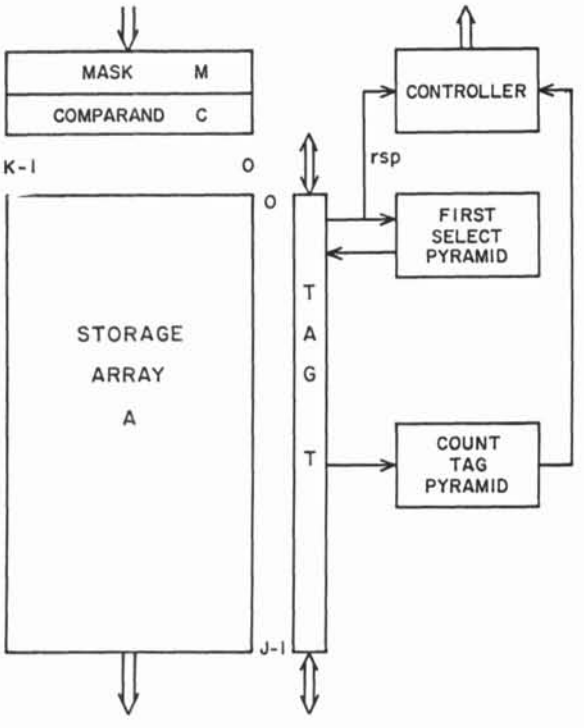

Fig. 1. ASSOCIATIVE PROCESSOR

TABLE 1

```
/*        WORD FORMAT     |---|---|
                          |---|---|
                         MARK  E0
 INPUTS : 1)  MARK  indicates all pixels above low threshold.
          2)  E0    indicates all pixels above high threshold.
 OUTPUT :     E0    labels all the edge points found.        */

main()
{
  /* ... declarations */

  letmc d(E0); setag; compare;  /* load Tag with E0 */

  while ( (growth = (new_condition - old_condition)) > threshold)
    {
     /* OR the three northern unambiguous edges into E0 */
          shiftag(-b); shiftag(-1); write;
          shiftag(1); write; shiftag(1); write;
          save_new_edges();
     /* OR the left and right unambiguous edges into E0 */
          shiftag(1); write; shiftag(-1); shiftag(-1); write;
          save_new_edges();
     /* OR the three southern unambiguous edges into E0 */
          shiftag(b); shiftag(1); write;
          shiftag(-1); write; shiftag(-1); write;
          save_new_edges();

      old_condition = new_condition;
      new_condition = countag;
    }
}
/* Find new edges by ANDing MARK into E0 */
save_new_edges()
{
      setag; compare;
      letc; write;
      letmc d(MARK); compare; letmc d(E0); write;
}
```

CURVE PROPAGATION WITH THRESHOLD HYSTERESIS

The process starts with every pixel labeled as to whether it exceeds each of the two thresholds. Pixels above *high* threshold are selected, those under *low* threshold are eliminated and the ones between *high* and *low* are returned as candidates. As in Canny edge detection [5], the propagation rule is to select every candidate that can be connected to a pixel above *high* through a chain of candidates. The propagation process is similar to *omnidirectional tracking* as described earlier by Rosenfeld and Kak [6], but does not produce any thinning.

The associative algorithm operates on a 3×3 neighborhood of all edge points in parallel, but does it in three stages: first on the northern neighbors, then on east-west, and finally on the southern neighbors. Selection can take place at any stage, thereby enhancing the propagation rate. The algorithm iterates until propagation stops. Execution time in machine cycles is given by,

$$T_p = I(32.5 + \frac{N}{b})$$

where I is the number of iterations, N is the row length and b is the extent of long shift. The N/b term accounts for communication with the two neighboring rows. While the upper bound of I is nearly $N^2/2$, for a representative value of $I = 100$ and our case of $N = 512$, $b = 32$, execution time becomes 4850 cycles, or 146 microseconds. The program is listed in Table 1.

THINNING

The propagation algorithm presented above produces a curve that may not be *thin*. A multi-pass thinning algorithm is proposed, which applies the following templates

```
X 0 X    0 0 X    X 0 0
1 P 1    X P 1    1 P X
X 1 X    X 1 X    X 1 X
```

first in the north direction, then in the south, east and west directions, in succession. This 4-pass sequence is iterated until there is no further change. Particularly worthy of note is the quality of the skeleton produced by this simple local process. The most precise definition of a *skeleton* is based on the *medial axis*. Davies and Plummer [7] proposed a very elaborate algorithm to produce such a skeleton, and chose 8 images for testing it. Our thinning algorithm was applied to these images and produced most interesting results: the skeletons agree virtually exactly with those of Davies and Plummer; any discrepancy, not at an end-point, occurs at a point of ambiguity and constitutes an equally valid result. Proof that the algorithm proposed here always yields the *ideal* skeleton will be given elsewhere. A *prethinning* phase is used to fill isolated ZEROs and to remove noise that may give rise to extraneous spurs in the skeleton. Time complexity of the algorithm is given by,

$$T_t = 70 + \frac{5N}{b} + I(150 + \frac{4N}{b}),$$

where the first two term account for the *prethinning* phase. Execution times are 150 cycles (4.5 μs) for *prethinning* and 214 cycles (6.4μs) per thinning iteration. For edge thinning 3 iterations will suffice, giving a completion time of 24 microseconds.

Single-pass thinning was considered and found to be rath-er critical. The algorithm proposed by Chin et alia [8] appears to be optimal, yet it does not yield an *ideal* skeleton, and application of their own preliminary phase of *noise-trimming* leads to amputation of some main branches during thinning.

CONTOUR TRACING AND LABELING

A preparation step labels each contour point with its x-y coordinates. The main process is iterative and operates on a 3×3 neighborhood of all contour points in parallel. Every contour point looks at each one of its 8 neighbors in turn and adopts the neighbor's label if smaller than its own. The circular sequence in which neighbors are handled appreciably enhances label propagation. Iteration stops when all labels remain unchanged, leaving each contour identified by its lowest coordinates. The point of lowest coordinates in each contour is the only one to retain its original label. These points were kept track of and are now counted to obtain the number of contours in the image. The time complexity of the algorithm (in machine cycles) is given by,

$$T_l = 16 + 4\log_2 N + I[65 + \frac{3N}{b} + 2\log_2 N(67 + \frac{3N}{b})].$$

Again, the upper bound of I is nearly $N^2/2$, but for a representative value of 100 iterations, execution time becomes 218 kilocycles or 6.6 milliseconds. A good approximation to the time complexity is,

$$T_l = I(67 + \frac{3N}{b})(1 + 2\log_2 N).$$

A list of contours, giving label and number of points, may be generated in relatively short order (22 cycles per contour). The program is listed in Table 2.

TABLE 2

```
*                    *** WORD FORMAT ***
  Image size is N X N pixels, i.e. coordinates fields are 2Log2(N)
  bits  in length.

      2Log2(N)       2Log2(N)    1  1  1  1  1    2Log2(N)    1
    |--------------|-------------|--|--|--|--|--|-------------|--|
    |---xy_coord---|---operand---|--|--|--|--|--|----label----|--|
                                 temp   sf lt gt               edge
                                   mark
INPUT FIELDS :
         1)  xy_coord  gives position of all pixels.
         2)  edge      indicates edge points.
INTERMEDIATE (WORKING) FIELDS :
         1)  label     gives x-y coordinates of all edge points.
         2)  operand   gives label of connected edge to be tested.
         3)  lt, gt    indicate if operand is less than or greater
                       than the label field.
         4)  sf        switch-flag indicates if label was changed
                       in current iteration.
         5)  mark      indicates if label was ever exchanged.
         6)  temp      holds edge flag of neighbor under test.
OUTPUT FIELDS :
         1) label      gives label of contours.
         2) mark       marks contour starting points.          */
main()
{
  /* ... declarations */

  /** Clear working fields **/
  letm dseq(label,mark); letc; setag; write;

  /*** Mark and label all edge points ***/
  letmc d(edge); setag; compare;
  letmc d(mark); write;
  for(bit_count=0; bit_count<label_size; bit_count++)
    {
      letmc d(xy_coord+bit_count) d(edge); setag; compare;
      letmc d(label+bit_count); write;
    }
  while( new_condition   >  growth_threshold)
    {
      letm d(sf) letc; setag; write; /*clear switch flag */

      /****** CONNECTIVITY TESTING ******/
      for(window_index=0; window_index<8; window_index++)
        {
          /* Shift "edge" and "label" into  "temp" and "operand" */
          letm dseq(temp,operand+label_size-1); letc; setag; write;
          for(bit_count=0; bit_count<label_size+1; bit_count++)
            {
              letmc d(edge+bit_count); setag; compare;
              letmc d(temp+bit_count);
              general_shift(window_index); write;
            }


          /** Test if "operand" < "label" **/
          letm d(gt) d(lt); letc; setag; write;
                              /* clear greater & less than flags */
          for(bit_count=label_size-1; bit_count>=0; bit_count--)
           {
            letm d(edge) d(temp) d(gt) d(lt)
              d(operand+bit_count) d(label+bit_count);
            letc d(edge) d(temp) d(operand+bit_count);
              setag; compare;
            letc d(edge) d(temp) d(gt) d(operand+bit_count); write;
            letc d(edge) d(temp) d(label+bit_count); setag;compare;
            letc d(edge) d(temp) d(lt) d(label+bit_count); write;
           }
          letmc d(lt); setag; compare;
          /* clear "label" and "mark", set switch flag */
          letm dseq(label,label+label_size-1) d(mark) d(sf);
          letc d(sf); write;
          /* copy ``operand'' into ``label'' */
          for(bit_count=0; bit_count<label_size; bit_count++)
            {
              letmc d(operand+bit_count) d(lt); setag; compare;
              letmc d(label+bit_count); write;
            }
        }
      /** Test for termination **/
      letmc d(sf); setag; compare;
      new_condition = countag;
    }
  /** Find number of contours **/
  letmc d(mark); setag; compare; countag;
}

/*        GENERAL SHIFT IN 3 X 3 NEIGHBORHOOD
                    | 2 | 1 | 0 |
                    |---|---|---| index 0-7 denotes
                    | 3 |   | 7 | neighbor position
                    |---|---|---|
                    | 4 | 5 | 6 |                    */
general_shift(index)
int index;
{
  if(index<=2) shiftag(b);
  if(index>=4 && index<=6) shiftag(-b);
  if(index==0 || index==6 || index==7) shiftag(-1);
  if(index>=2 && index<=4) shiftag(1);
}
```

CONCLUSIONS

Associative algorithms for contour processing were presented that achieve remarkable performance with a modest hardware complement. They were developed and verified using a simulator of our own design. When tested on some real as well as synthetic images, the *curve propagation* and *thinning* algorithms, supplemented by *edge gap filling* and *short edge removal* steps, produced clean, continuous edge lines.

References

[1] C. C. Foster, *Content Addressable Parallel Processors,* Van Nostrand Reinhold Co., 1976, chs. 2 & 5.

[2] S. Ruhman, I. Scherson, *Associative Processor for Tomographic Image Reconstruction,* Proc. Medcomp 82 (1982 IEEE Comp. Soc. Int. Conf. on Medical Comp Sc./Computational Medicine): 353-358.

[3] S. Ruhman, I. Scherson, *Feasibility Study of Associative Radar Signal Processing,* Internal Report, Weizmann Institute of Science, October 1984.

[4] S. Ruhman, I. Scherson, *Associative Processor Particularly Useful for Tomographic Image Reconstruction,* U.S. Patent 4,491,932, Jan. 1, 1985.

[5] John Canny, *Computational Approach to Edge Detection* , IEEE Trans. on Pattern Analysis & Machine Intel. 8: 679-698, 1986.

[6] A. Rosenfeld and A.C. Kak, *Digital Picture Processing,* Vol. 2, Academic Press 1982.

[7] E.R. Davis and A.P.N. Plummer, *Thinning Algorithms: A Critique and New Methology,* Pattern Recognition 14: 53-63, 1981.

[8] R.T. Chin, H.K. War, D.L. Stover, and R.D. Iverson, *A One-Pass Thinning Algorithm and its Parallel Implementation,* Comp. Vision, Graph. & Image Proc. 40: 30-40, 1987.