Associative parallel processing

by R. H. FULLER
General Precision Inc.
Glendale, California

INTRODUCTION

Several previous investigations\(^1\)\(^-\)\(^5\) have shown associative processors to have utility in solution of problems which allow the same operation to be performed simultaneously over many elements of a data set. Examples of such problems include picture processing, matrix manipulation, signal correlation, ELINT data processing and information retrieval. It is the purpose of this paper to contrast the efficiency of solution of such problems on an associative parallel processor to efficiencies obtained using other computer organizations described in this session. Efficiency is measured as solution rate per unit cost.

The associative parallel processor (APP)\(^5\) is an array computer exhibiting parallelism of instruction execution similar to that obtained in SOLOMON type machines. It differs from SOLOMON machines in having much less storage (e.g., a single associative memory word) dedicated to each array element. Logic hardware at each array element is minimized by use of a novel computing technique termed Sequential State Transformation. By this technique, data stored within an associative memory having a multiwrite capability may be transformed according to any Boolean function of stored and external variables.

Cell cost for APP is two to three orders of magnitude less than for a SOLOMON type cell. Instruction execution times for APP cells are longer than for SOLOMON cells by one to four orders of magnitude, dependent on instruction complexity. An APP should be more efficient than SOLOMON type machines in applications having small cell memory requirements and simple data manipulation at each cell. Such applications include picture processing, information retrieval, signal correlation and simple matrix manipulations (e.g., solution of assignment matrices). Solutions of complex field problems, particularly when variable ranges dictate floating point number representation, are not normally efficient on APP, even though these problems allow parallel computation requisite to efficient use of an array computer. Problem solution time for an APP can only approach that of a SOLOMON array if the APP contains, and may effectively use, many more cells than the SOLOMON array. An APP would typically contain several thousand cells as opposed to several hundred for a SOLOMON array.

In regard to generality of the various processors discussed in this session, the conventional uni-processor employs a single instruction stream operating on a single data stream and is thus uniformly effective on all problems capable of digital solution. An aggregate of conventional uni-processors employs multiple independent instruction streams, each operating on a single data stream. This organization is effective whenever a problem can be partitioned into several subproblems, each of which allows of independent solution. An aggregate of uni-processors is less general than a single uni-processor in that problems which do not allow partitioning do not effectively utilize the machine.

An array processor employs a single instruction stream operating simultaneously on many data streams. This mode of operation can be achieved with an aggregate of conventional uni-processors by supplying the same instruction sequence to each uni-processor. An array processor is thus less general in application than an aggregate of uni-processors.

Among array processors, an APP is further restricted in application than a SOLOMON type processor, in that floating point arithmetic operations are not available within APP. An APP thus has restricted utility in some scientific computations for which a SOLOMON machine is well suited.

It is felt that within its useful range of application, the efficiency of APP exceeds that of other organizations discussed in this session. To illustrate this point, we shall, in the following section, review the organization of APP and discuss its utility in picture processing, a task to which the APP is well suited.
Processor organization and command set

To illustrate the concept of sequential-state-transformation, consider an associative memory which stores two operands, A and B, in each word of memory. We desire to add operand A to operand B, simultaneously in some subset of these words. Processing is serial by bit, and parallel by word, starting at the least significant bit of each field. Each word has an auxiliary storage bit, C, stored within the memory array. Bits within operand field A are designated \( A_{ij} \) (j = 1, 2, . . ., N), where N is the field length. Bits in field B, are similarly designated. The truth table defining the addition is as follows:

<table>
<thead>
<tr>
<th>State number</th>
<th>Present state</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( A_{ij} )</td>
<td>( B_{ij} )</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Note that variables \( B_{ij} \) and C differ in their present and next states only in states numbered 2, 4, 5, 7. The search and multiwrite operations may be used to perform the addition of all number pairs, starting at the least significant bit, as follows:

1. Search words having \( A_{ij} = 1, B_{ij} = 1 \) and \( C_i = 0 \).
   For these words, multiwrite \( B_{ij} = 0, C = 1 \).
2. Search words having \( A_{ij} = 0, B_{ij} = 0 \) and \( C_i = 1 \).
   For these words, multiwrite \( B_{ij} = 0, C = 0 \).
3. Search words having \( A_{ij} = 0, B_{ij} = 1 \) and \( C_i = 1 \).
   For these words, multiwrite \( B_{ij} = 0 \).
4. Search words having \( A_{ij} = 1, B_{ij} = 0 \) and \( C_i = 0 \).
   For these words, multiwrite \( B_{ij} = 1 \).

Steps (1) through (4) are repeated at each bit of the operands. Within each bit time, processing is sequential by state over present states which differ from the next state in one or more variables. All words in a given present state are transformed simultaneously to the desired next state.

Sequential-state-transformation, used to perform the above word-parallel, bit-serial addition, is evidently a very general mode of associative processing. It allows transformation of memory contents according to any Boolean function of stored and external variables. It makes full use of comparison logic, implemented at the bit level within an associative array, and thereby simplifies logic required at the word level.

In the following we describe the organization and command set for a processor using the sequential-state-transformation mode of associative processing.

Figure 1—Structure of the associative parallel processors

Elements of an associative processor are shown in Figure 1. The format for associative commands is shown in Figure 2. Each associative command effects a primitive transformation of state variables as discussed above. The left-most bit identifies the command as associative. The two adjacent bits define the initial state of match flip-flops in word logic units (i.e., the detector plate). Other bits define search and rewrite criteria for the A field, the B field, and for each of four tag bits. The right-most bit controls rewrite into matching words or their next lower neighbors. Functions of these bits are described in Figure 2.

To illustrate the utility of this command, consider the task of searching the associative memory for words matching the data register over a field having its upper limit stored in the A limit register and its lower limit stored in the A counter. Matching words are to be tagged in tag bit 1.
The following command accomplishes the desired tasks:

1. Set the match flip flop for word 0 to "1."
2. 1 N S L D W S — — — S — W N
3. 1 N S L D W S — — — S — W L

A Control    B Control    Tag 1

4. If no match, exit: otherwise go to (3).

Instruction (2) writes into word 0; instruction (3) writes sequentially into each remaining associative word.

Nonassociative commands are provided to load the A and B counters and limit registers, to branch from linear instruction sequencing either unconditionally or when specified conditions are met, and to input or output data.

**Associative pattern processing**

The parallel processing capability of an associative processor is well suited to the tasks of abstracting pattern properties and of pattern classification by linear threshold techniques. Threshold pattern recognition devices execute a given operation independently over many data sets, and thus allow the parallelism necessary for efficient associative processing. Associative processing affords the accuracy of digital number representation, and is thus unlimited in fan-in and dynamic range of weights. Weights are simply altered by changing memory contents. Wiring and components are regular and are thus amenable to low-cost, batch-fabrication techniques. The set of measured pattern properties is changeable by changing memory contents, rather than by rewiring as for analog units. Adaptation is thus possible in measured properties as well as in classification.
Figure 3—Analog model of pattern recognition system

Figure 3 represents the model of the pattern recognition system to be realized. Binary valued sensor outputs are summed, with weights ±1, into some or all of the thresholding logic units. A threshold level is established for each logic unit. If the sum of binary weighted inputs exceeds the threshold, the unit becomes active and its output is "one"; otherwise the output is "zero." Each logic unit has a weighted connection to some or all of the response units. Weights of active logic units are summed and thresholded at each response unit. A pattern is classified according to the set of activated response units.

To realize this model, the associative memory is organized into three sections containing, respectively, a connectivity matrix defining connections between sensors and logic units; the system of weights associated with inputs to response units, and the target vectors associated with patterns to be recognized. The general organization of the associative memory is shown in Figure 4. Processing takes place as follows: In Phase (1), the set of logic units activated by the input pattern is determined, using the input pattern and the stored connectivity matrix. Logic unit outputs are formed in the detector plane. In Phase (2), the inputs to the response units are calculated, using logic unit outputs and the weights stored in the second sector of the associative memory. This yields the response unit outputs in the detector plane. In Phase (3), the response unit outputs are compared with the target vectors stored in the third sector associative memory, and the pattern classification is determined.

Figure 5 illustrates pattern recognition times for an associative processor having an execution time for associative commands of 0.8 microseconds. Here, "N" represents the number of sensor units at the input and "M" the number of patterns distinguishable by the processor. The APP can classify an input pattern employing 400 sensors and 512 logic units in approximately 3 milliseconds. Some 2000 words of associative storage are required.
CONCLUSION

The associative parallel processor, described in this paper, achieves considerable generality with simple word and bit logic through the use of the sequential-state-transformation mode of associative processing. An important feature of the parallel processor, when used as a pattern recognition device, is the ability to modify its functional structure, through alteration of memory contents, without change in its periodic physical structure. This adaptive feature has importance in applications where patterns change with time, or where the processor is used as a prototype of subsequent machines having fixed recognition capabilities.

Linear threshold pattern classifiers of the type here presented are beginning to find many applications. To date, these types of pattern classifiers have been studied and/or implemented for character recognition, photointerpretation, weather forecasting by cloud pattern recognition, speech recognition, adaptive control systems and more recently, for medical diagnosis from cardiographic data. Other possible applications include terminal guidance for missiles and space vehicles and bomb damage assessment.

In addition to the pattern recognition task herein described, the APP has also been applied to the tasks of job shop scheduling, optimum commodity routing and processing electromagnetic intelligence (ELINT) data. In each instance significant speed gains have been shown possible over conventional sequential digital computers. It is interesting to note that the processor described in the second section of this paper may be applied to this variety of tasks without significant changes in organization or command structure.

REFERENCES

2 Ibid. Some applications for content-addressable memories Proc. Fall Joint Computer Conference pp. 495-508 Nov. 1963
4 R. G. EWING and P. M. DAVIES An associative processor Proc. Fall Joint Computer Conference pp. 147-158 1964
5 R. H. FULLER and R. M. BIRD An associative parallel processor with application to picture processing Proc. Fall Joint Computer Conference Nov. 1965
6 R. WIDROW et al Practical applications for adaptive data processing systems 1963 WESCON Aug. 1963
7 C. H. MAY Adaptive threshold logic Rept. SEL 63-027 (TR 1557-1) Stanford Electronics Lab, Stanford, Calif. Apr. 1963
8 G. NAGY System and circuit designs for the tobermory perceptron OTS AD 604 450 Sept. 1963