INTRODUCTION

Some of the advantages of a highly parallel system are well known. With several computers and memories available, double or triple redundancy can supply extremely high reliability where such reliability is a necessity. The loss of one computer will mean a reduced computing rate but no complete breakdown of the system; this desirable attribute has been called graceful deterioration. We can also have modular enhancement when the user who wishes to solve his problems more rapidly has but to add another computer module or two in parallel with his existing installation, if his problems are responsive to parallel treatment. Complex real-time problems of supreme urgency will require a high degree of parallelism if they are to be solved. Of course all of these advantages of parallel systems assume an exceedingly flexible means of programming and control for the system.

What is not so generally well known is that a high degree of parallelism will likely be the key to the economic mass fabrication of logic units in a mass fabrication technology such as cryogenics. Here the cost of a design is largely determined by its complexity rather than by the total number of elements involved. A memory plane seems economically feasible because of its low complexity, i.e., repetitive pattern. But the much less repetitive patterns of conventional arithmetic and logic units become excessively expensive for the complex design. This argues for an elementary type of arithmetic unit, repeated many times, provided that the simple controls and programming are available.

Several proposals within the field of parallel computer organization have been made. Most of these have been in the direction of parallel programming, i.e., multiprogramming, where a single computer deals with several programs, hopefully to increase throughput. H. Hellerman proposes, in addition, a multiprocessing function, taking advantage of a small associative memory for memory address transformation. Unger has proposed a parallel network system with central control; problems of a high degree of symmetry could be solved, with different parts of the problem being done, synchronously, in different modules. The programming for this system appears reasonable, but the very stringent symmetry relationship limits the type of problem. The Slotnick proposal for the Solomon Computer proceeds along somewhat similar lines. Holland has proposed a machine with distributed control for solving more general types of problems. His organization appears to have very severe difficulties in programming.

THE ALPS ORGANIZATION

The use of associative memories for data and instructions, and associative logic to control the parallel computers, offers an attractive solution to the programming difficulties usually encountered with conventional systems. The result is an organization that will permit a very high degree of parallelism, both in processors and
problems, with practicable programming. Figure 1 shows the block diagram of an Associative Logic Parallel System (ALPS).

ALPS, like most computing systems, will consist of five functional parts: input; output; memory; control; and computing. However, in ALPS, functional parts such as multiple computer and memory modules may be utilized without any central control. This ability of non-central control and non-distributed control (in the Holland sense), is a novel approach to parallel computing systems. These multiple functions are designed, using associative techniques, to execute one or more problems in parallel. In addition, these functions may be added or subtracted without making essential changes in the programs.

The block diagram of ALPS in Figure 1 shows the general interconnections of the functional parts. The control and computing appear in the diagram as one unit called the computer module. The interconnections will be designed so that the transfer of data among many functional parts may be carried on simultaneously.

The input section of the system will simultaneously take data from any number of input sources and store it in the storage units. Such data may come from punched cards, magnetic tape, disks, manually operated keys, sensors, radar, etc.

The output section simultaneously takes information from the storage units and puts it at the operator's disposal. The output may be in many forms such as punched cards, magnetic tape, printed forms, indicator lights, or displays.

The memory section accepts, holds, and distributes information. For this section, the block diagram shows several associative memory modules, each with an input queue. Each associative memory has the first-in first-out (FIFO) and last-in first-out (LIFO) functions, as well as many of the more usual associative features such as ordered retrieval, selective write, write into the first vacant register, etc. The associative memories will be designed to operate independently.

Each computer module can decode instructions it receives; send requests for data to the associative memories; operate on the data with the usual arithmetical and logical operations; and compute Tags for retrieving successor instructions and operands.

**BINARY TREE PROGRAM**

One programming method employs a binary tree structure. This program structure is based on earlier work for symbol manipulation. Several processors can be simultaneously working down the branches of the tree with the traffic controlled and the instructions and data retrieved by associative Tags and Status marks. Multidirectional, simultaneous, conditional and nonconditional branching is provided for maximum effectiveness of the parallel mode. Memory efficiency is maintained by multiple memories and memory queues for minor operations directly on words in memory.

The memory word as used in the tree program structure is set up in four fields:

<table>
<thead>
<tr>
<th>Status</th>
<th>Tag &amp; subscripts</th>
<th>Code</th>
<th>Item &amp; subscripts</th>
</tr>
</thead>
</table>

From the collection of the Computer History Museum (www.computerhistory.org)
The Tag with its subscripts is the principal field for associative retrieval, although the associative operation will at times be applied to one or more of the other fields. The principal part of the Tag gives the name of the tree, while the subscripts show where that particular word falls in topological relationship to the rest of the tree. The Status characters are used to indicate such things as “Ready” for computation, “Branching” structures, “Constants” and “Program” constants, and “Marked” words. The Code field essentially describes the contents of the Item field; that is, whether the item is an “Operation” (+, −, etc.), a “Number”, a reference to another “List” or to “Data”, or is an “If” statement. If the item is a reference to another list (i.e., tree) or sublist, it will contain the Tag of that list with any necessary subscripts.

**Tree Example**

The organization and application of the tree structure described will be made clear by the following simple example:

\[ Y = AB + CX \]

In conventional programming, this would be represented by a serial list such as:

- Load Multiplier \( L(A) \)
- Multiply \( L(B) \)
- Store \( L(E) \)
- Load Multiplier \( L(C) \)
- Multiply \( L(X) \)
- Add \( L(E) \)
- Store \( L(Y) \)

There would also be four words in memory giving the numerical input values for \( A, B, C, \) and \( X \). The programmer would decide which multiplication to perform first, although the order is immaterial. Note that these two multiplications can be done simultaneously if two arithmetic units with appropriate control are available.

To perform the same operations in our parallel system, the program, in tree form, would be as follows:

```
<table>
<thead>
<tr>
<th>Y001 D A</th>
<th>Y011 D B</th>
<th>Y101 D C</th>
<th>Y111 D X</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Y01 OP</td>
<td></td>
<td>Y11 OP</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Y1 OP</td>
<td></td>
</tr>
<tr>
<td>A N 1.2</td>
<td>B N 2.3</td>
<td>C N 3.4</td>
<td>X N 6.7</td>
</tr>
</tbody>
</table>
```

* indicates multiplication
significant. The second subscript, 1, and first subscript, 0, are significant representing the right branch of a left branch. Hence, given the Tag for any word, it is easy to compute the Tag of (and thus retrieve associatively) the succeeding, preceding, or complementary word. This is a salient feature of the ALPS system. The four words with Tags A, B, C, X are the input data words. They are not shown as part of the tree.

The parallelism of the two multiplications is obvious: with two computers, we can start at the top of the tree and work our way down along the two branches. When the value for A is loaded, an interrogation on Item A will retrieve program word Y001 and generate the new Ready word, 

\[
\begin{array}{ccc}
R & Y001 & N 1.2 \\
\end{array}
\]

The Ready symbol R will indicate that Y001 is ready for computation. Similarly the words Y011, Y101 and Y111 will be generated with the Ready symbol R indicating that they are Ready for computation with the numbers 2.3, 3.4, and 6.7, substituted for B, C, and X, respectively. At this point a free computer interrogating the memory for Ready symbols can destructively retrieve word Y001. Three registers in each computer module are provided to deal with triples of words, in this case Y01 and Y011 in addition to Y001. From the Tag Y001 the Tags Y01 and Y101 are easily computed. The Tag Y01 enables the retrieval of the instruction word \[
\begin{array}{ccc}
Y01 & OP & * \\
\end{array}
\]

With Tag Y011 and Code N, an interrogation is made to determine the availability of the appropriate word. If the loading of the B word has been completed, the retrieval is successful, the multiplication will be completed, and the result 2.8 retained temporarily in one of the three registers, now with the Tag Y01. While this multiplication was taking place, another computer may have found the Ready marks on words Y101 and Y111 and will be working on the multiplication called for by Y11. The first computer will now retrieve instruction word Y1 and, with Tag Y11 and Code N, will make an interrogation to see if the appropriate word is available for its other operand. With this interrogation unsuccessful, the first computer will dump (FIFO) its partial result in memory with the Ready status marked and with Tag Y01. The first computer is then free to look for other words with Ready marks, while the second computer will find the necessary partial result available when it makes its interrogation. Thus the second computer can continue to the final result.

### Iterative Loop

Another example of a tree-structured program involves a square root loop. The square root loop is an iterative loop where the number of iterations is based on a maximum predetermined absolute error. This program uses the novel feature of simultaneous multiple branching, which is available in ALPS.

Figure 2 shows a tree representation of a program to solve the following equation.

\[ Y = 6 - \sqrt{K} \]

Where

\[ \sqrt{K} = X_{j+1} = X_j - \Delta_j \text{ if } T - | \Delta_j | > 0 \]

\[ T = \text{predetermined absolute error} \]

\[ \Delta_j = \frac{X_j^2}{2X_j} \]

\[ \sqrt{x + x_{j+1} + x_j - \Delta_j} \text{ if } T - | \Delta_j | \geq 0 \]

\[ \Delta_j = \frac{x_j^2}{2x_j} \text{ GIVEN } T, K, \text{ AND } X_0 \]

---

**Figure 2. Tree Representation of** \[ Y = 6 - \sqrt{K} \]

---

From the collection of the Computer History Museum (www.computerhistory.org)
The basic operations for this program are the same as those described earlier for the simple example. Operations will be initiated by the loading of T, K, and X. However, several important additional functions come into the operation.

Generated data words are normally read destructively as they are retrieved. However, if they are derived from input constants (Status C), they are held in memory for reuse and are deleted only when a new value is brought in from the input. For example, the word

| C | Y11101 | N | 0.1 |

is generated from the input T. Program constants and program instruction words are deleted by program control only when all use of them has been completed.

The Unconditional Branch (note in Figure 2 the word with Status B and Tag Y11011) has a meaning different from that used in conventional programming. Here we have a two-way branch and we want in each case to go in both directions in parallel, employing the number generated at the branch point. One branching takes us down the tree in the usual fashion; the other follows the curved line to the word with Tag Y111111. This connection is found because Item Y11011 of this word agrees with the Tag Y11011 of the word initiating the branch. The word is found by associative interrogation on the Item field. The Code L indicates that the Item is a reference to another list.

The Conditional Branch appears in the word with Tag Y111, Code IF. This is a one-way versus four-way conditional branch. The conditioning is determined by the sign of the operand generated at the right branch. If the sign is positive, our iteration is complete, and we proceed down the tree employing the value given by the left branch. But if the sign of the right operand is negative, the iteration must be repeated and branching occurs four ways in parallel as indicated by the dotted curved lines. The value given by the left branch is used in all four cases. For this IF instruction, the Item X serves to locate, associatively, the four words with Items X, thus closing the loop.

The loading of a new value for K can serve to reinitiate the computation; or all three parameters, T, K, and X, may be reloaded to start a new block of computations.

SUMMARY

We have presented a brief description of the organization of an associative logic parallel system. This organization provides for the concurrent solutions of one or many problems using more than one computer. One approach to the programming of such a system was described, and two examples were presented.

The main feature of the system is its autonomous control implemented through associative logic. It is anticipated that the programming will not be too difficult, nor will there be substantial changes in programming because of modular expansion.

REFERENCES
