Some thoughts on associative processing languages

by WILLIAM W. PATTERSON

Rome Air Development Center
Griffiss AFB, New York

INTRODUCTION

Much effort has been expended in developing array and associative processors (AP's). The most notable of the former are Burroughs' ILLIAC IV and Honeywell's PEPE, while the present representative of the latter technology is the STARAN built by Goodyear Aerospace Corp. However, very little has been published on higher order languages which take advantage of the unique characteristics of these architectures. There is at least one effort to develop techniques which extract the parallelism in ordinary FORTRAN code, as well as a number of efforts to formally describe the parallelism in algorithms. Examples are in References 2 and 3. It is true that many algorithms can be put into efficient parallel code using these techniques; however, there is a large body of problems which must be re-examined and recast into new algorithms which match the parallelism of the machine to the natural parallelism of the problem. These new algorithms will require a new language which gives the programmer the flexibility to use the features of the machine directly. The PFOR language developed for PEPE is probably the only existing language for an array processor, and some preliminary work for the RADC AP project is the only published attempt on AP languages. This paper will look at associative processing from the point of view of a programmer who has tried to write programs for an AP, and therefore will propose constructs which are convenient for the programmer and not necessarily for the compiler writer. They do, however, stem from a reasonable knowledge of the basic architecture of the AP, and hence will tend to parallel it.

It is necessary at this point to talk about the general architecture of the AP and, in the process, define some terms that will be used in the paper. The two main units in the AP are the control unit and the associative processing elements. The control unit includes a mainframe memory, which holds the programs, constants and common single valued variables; local arithmetic capability that can be used to perform processes which concern only the common variables; and control logic to drive the associative processing elements. The associative processing elements each consist of an associative word of 256 or more bits, and logic to process the data in that word. The associative word is divided into several fields of either fixed or varying length. Normally, if the fields are of varying length, they are defined by the variables of the problem. Since the conventional computer performs operations sequentially, the term "sequential" will be used in the sequel to distinguish the conventional computer; however, the modifiers "sequential" and "associative" may be left out if the context makes it clear which is meant.

LANGUAGE

In designing a language such as this, two choices are possible; one can design a complete new language, or he can modify an existing language to include the proper constructs. The latter approach was chosen because very few problems are completely associative and it is anticipated that most installations that include an AP will also have a conventional computer with facilities for communication between the two. This allows the programmer to write all his code in the same basic language, identifying the associative parts. The language chosen is PL/1 for several reasons, not the least of which is the author's familiarity with the language. However there are more cogent reasons; the first is that the basic block structure of PL/1 lends itself to segregation of sequential and associative tasks into separate routines. Other reasons include its basic self-documenting qualities, its extreme flexibility which allows its use for a large number of problems and the fact that it has a degree of parallelism already built in which might be exploited.

Declaration

The associative tasks in the problem should be segregated from the sequential tasks on a procedure level. This is easily facilitated by defining an associative procedure as qualified procedure much like the presently defined recursive procedure. The form of the procedure declaration statement would be:

label: PROCEDURE (parameters) attributes
ASSOCIATIVE nr_entries;

or alternately:

label: PROC (parameters) attributes ASSOC nr_entries.
This statement would tell the compiler that procedure (label) should be compiled into AP code. The parameters and attributes fields are optional and follow the same rules as in PL/1. The optional nr_entries field tells the number of associative entries needed by this procedure,* where an associative entry is one of a number of identical sets of variables, each of which has a unique set of values and each of which will be processed in parallel with all the others. Associative entries are distinct from associative words, since it is quite possible to have more than one entry per word or to have one entry fill more than one word. However, the task of controlling these configurations is best left to the computer. This allows the programmer to specify an associative entry of the length appropriate to the problem, and conceptually think of an entry as an associative word (this being the case, no distinction will be made between the two in the sequel). An example of a set of associative entries is a radar track file which keeps a record of all the tracks being monitored by a radar set or system (for example, an air traffic control radar). Each entry stores all the information about one track, such as position coordinates, track quality and any keys or flags which give additional information about the vehicle being tracked.

It should be noted here that if a sequential procedure calls an associative procedure or vice-versa, and there are no data dependencies, then the two procedures can run concurrently on the two machines. Therefore, suitable WAIT statements must be inserted when the calling procedure needs data from the called procedure. This will cause the calling procedure to wait at this point in its execution until the procedure it called has finished, thus assuring that the data required is properly updated.

The question now arises as to what variables should be passed between associative and sequential procedures. The normal PL/1 convention is that a variable declared in a procedure is available to all procedures it calls but not to procedures that call it. This seems impractical in this case since any call from one type of procedure to the other implies that the data must be passed over a physical channel. For this reason, the author favors the restriction that only formally declared parameters be passed between different types of procedures, with the normal rules applying to calls between two procedures of the same type. This dichotomy should not be troublesome to the programmer, since he must know that he is working on two different machines.

In associative processing, there are two basic types of variables. The first type is the common variables and constants which are single valued and therefore are held in the mainframe memory. The second type is the associative variables which have a value for each of the associative entries. The distinction between these two types of variables will be made in the variable declaration of the associative procedure. The variables of the first type would be declared in the normal manner, where the STATIC attribute would indicate that the variable would be assigned a static location in the AP mainframe memory. Associative variables would be declared in the same manner except that the keyword ASSOCIATIVE (abbreviated ASSOC) would be appended after the declaration. This declaration then defines a field in each associative word. For example, the variable which represents the range coordinate of the tracks in the track file mentioned above would be declared:

```
RANGE FIXED BIN(x1,x2) ASSOCIATIVE;
```

This indicates an associative variable called RANGE, a fixed binary quantity of precision x1 (i.e., a field of x1 bits in each associative word) with fractional part x2.

Dimensionality of associative variables has to be handled differently than with sequential variables, of course. The first dimension of all the associative variables is the number of associative entries. Since this number is contained in the procedure declaration, it would not appear in the variable declaration. If arrays are desired in each entry, then, of course, the normal convention would apply. Clearly, if an array is passed from a sequential procedure to an associative procedure and is stored as an associative variable, then the associative declaration should not include the first dimension of the array.

Association

The major unique feature of the AP is the ability to make associations; that is, it is possible to make parallel comparisons either between a comparand held in the control portion of the AP and a field in each of the associative words or between two fields in each of the associative words. Normally the processing which follows is performed only on those words that give an affirmative response or, in the jargon "respond," or on a subset of these words. It is convenient to talk about the words that have responded as being active. There are two basic association actions to be performed. The first is to associate only on those words that are active, and thereby further reduce the number of active words. This will be performed by an ASSOCIATE statement. The second type of association is used to reactivate a number of words after a series of ASSOCIATEs. This is accomplished by performing an association on all associative words and is invoked by the ACTIVATE statement. Most AP's have the capability to associate on the basis of the three basic relational operators, greater than (>), less than (<), and equal (=).

An ASSOCIATE statement would consist of the keyword ASSOCIATE followed by a conditional statement which include conditions using one or a combination of the above relations or logical combinations of the conditions using logical AND (&), logical OR (+), and logical NOT (¬). As an example of this type of statement, suppose that the associative variables RANGE_DIST and AZ_DIST hold the polar vector distance of each track from a current radar return, and we wish to determine which of these tracks is within a certain distance from the point of the return, where

---

* If this field is absent, then the default would be that nr_entries = the number of associative words in the machine.
that distance is held in mainframe memory in locations called RANGE_WINDOW and AZ_WINDOW. The statement would be:

```
ASSOCIATE RANGE_DIST <RANGE_WINDOW & AZ_DIST <AZ_WINDOW;
```

This statement would leave active all associative words which contained a track which fell within the window and deactivate all others.

There are two special capabilities of the AP that should be included in this section. These are the capability to find the minimum value in a given field and to activate all words which have that value in that field and an analogous capability to find the maximum. These capabilities can be invoked by an ASSOCIATE statement as follows:

To find the minimum value:

```
ASSOCIATE MINIMUM variable_name;
```

and to find the maximum value:

```
ASSOCIATE MAXIMUM variable_name;
```

Where “variable_name” is the name of the variable assigned in that field.

The second type of association statement is used to reactivate a larger set of associative words. It is the ACTIVATE statement and it works the same as the ASSOCIATE statement except that the keyword at the beginning of the statement is ACTIVATE, and it activates all words that meet the condition, not just the previously active ones. In addition, the keyword without any condition has meaning and that is that all associative words should be activated.

Pursuing the radar tracking example to illustrate the conditional ACTIVATE statement, suppose that you wish, after a series of associations which identified a small number of tracks, to reactivate all associative words that contain valid tracks and that each track entry contains a one bit flag called BUSY which is 1 for every valid track and 0 for all others. The statement:

```
ACTIVATE BUSY;
```

would activate all valid tracks.

There are instances when it is necessary to activate one and only one associative word in a group but it is not critical which word is activated. In this case, the associative processor has the provision to activate the first word which meets the conditions of the search. This provision can be invoked by adding the keyword FIRST to the ASSOCIATE or ACTIVATE statement. For example, suppose that in the radar tracking problem, we wish to establish a new track, using the first empty word. The word can be activated with the statement:

```
ACTIVATE FIRST = BUSY;
```

Another feature that is quite useful is the capability to select a subset of the active words for execution of a short series of instructions without deactivating the other words.

This can be invoked by using a FOR statement, which has the same syntax as the normal IF statement, except that the keyword FOR is substituted for IF. This signifies that all associative processing elements which meet the condition would execute the code between the THEN and END brackets. If an ELSE portion is included in the statement, it would be executed by all active words which do not meet the condition. This means that both parts of the statement are executed each time the statement is encountered.

**Assignment**

The last type of statement that will be discussed is the assignment statement. The simplest type, of course, is the move:

```
X = Y;
```

Let us now consider the four possible combinations of variables. If both X and Y are common variables, then the move is carried out in the normal manner. If both X and Y are associative variables then the statement constitutes a move of data from field Y to field X in each of the active words. If X is an associative variable and Y is a common variable, the statement constitutes a broadcast of the data in location Y to field X in all active associative words. If X is a common variable and Y is an associative variable, only one word can supply data, since there is only one location in mainframe memory to receive it. The source chosen is field Y of the first active word.

For more complex assignment statements, containing two or more variables on the right side of the equal sign, it is clear that if any one of the variables is associative, then the operation must be performed in the associative processing elements. Once the answer is found, the assignment will follow the rules shown in the preceding paragraph. For example, suppose we wish to calculate the distances used earlier for association, given that the information on a new report is stored in a structure in mainframe memory called REPORT. The code would be:

```
RANGE_DIST = ABS( REPORT.RANGE - RANGE );
AZ_DIST = ABS( REPORT.AZ - AZ );
```

Since RANGE and AZ are both associative variables, these calculations would both be performed in the associative processing elements; and since RANGE_DIST and AZ_DIST are both associative variables, the result would be transferred directly to the proper fields in each associative word.

**CONCLUSION**

This paper has proposed language forms, based on the PL/1 language, which will give the programmer the capability to directly use all of the features of an associative processor without having to revert to assembly language coding. Many of the statements will translate into one or two lines of machine code, but this is necessary to use of the full power
of the machine. The capability is important, because there are a class of problems which will require new algorithms to effectively use the machine by matching the parallelism of the machine to the parallelism of the problem.

REFERENCES


