APPLICATION OF APL FOR RAPID VERIFICATION OF A DIGITAL SYSTEM ARCHITECTURE

Behrooz Parhami

Department of Mathematics and
Computer Science
Arya-Mehr University of Technology
Tehran, Iran

Abstract: An instruction set for a special-purpose associative processor, designed for information storage and retrieval applications, is defined in APL/360. These definitions are then used to test the validity of the system's architecture for the proposed applications. This is accomplished by writing and simulating the execution of sample programs on a sample data set and selectively observing the corresponding transformations performed on the data set.

INTRODUCTION

This paper reports on an experience with APL gained in attempting to verify the correctness and consistency of a digital system design at an architectural level. Specifically, a set of string-manipulating instructions of the system were considered and the effects of each were defined as a function in APL/360. These functions were then employed in writing programs which served two purposes:

1. Verifying that the instructions (and programs) have the expected effects, in the sense that they transform their arguments in the manner predicted.
2. Verifying that desired searches and transformations on data can be accomplished using the given instructions; i.e., that the instruction set has sufficient power for our purposes.

It is assumed, of course, that the APL/360 functions representing the instructions are totally equivalent to their word descriptions generated with the system design. The instructions are so simple, however, that this assumption is reasonable.

A BRIEF DESCRIPTION OF THE SYSTEM

The digital system referred to above is called RAPID (Rotating Associative Processor for Information Dissemination) which is a special-purpose associative processor designed for information storage and retrieval applications. A complete description of RAPID has been published previously (1), (2). Here, we summarize those features of the system which are relevant to this discussion.

Figure 1 shows a block diagram of the RAPID system. The information stored on a head-per-track disk unit is viewed as several (independent) character strings by the Cells, each Cell processing one of these strings. Hence, it is sufficient for this discussion to consider only one such string. By a character we mean a pair (q,s), where s is a symbol from an alphabet X (i.e. $s \in X$)

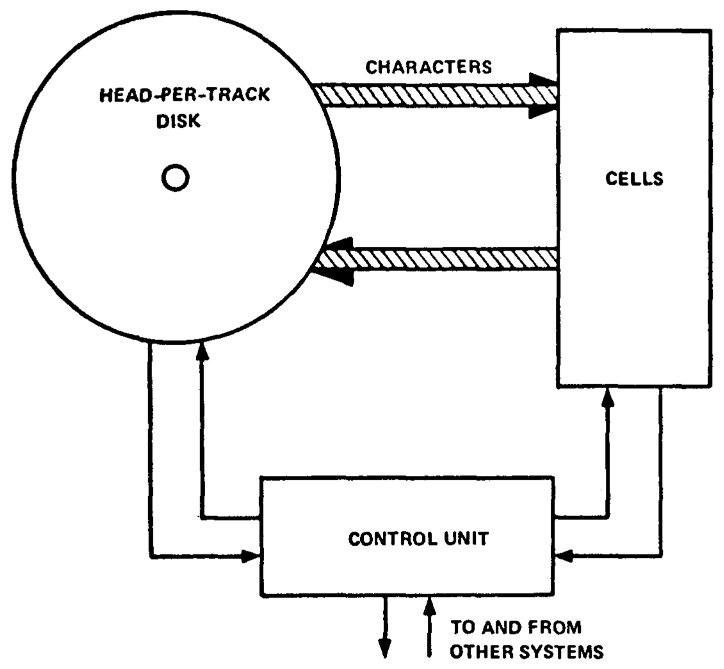

Figure 1. Overall Organization of RAPID

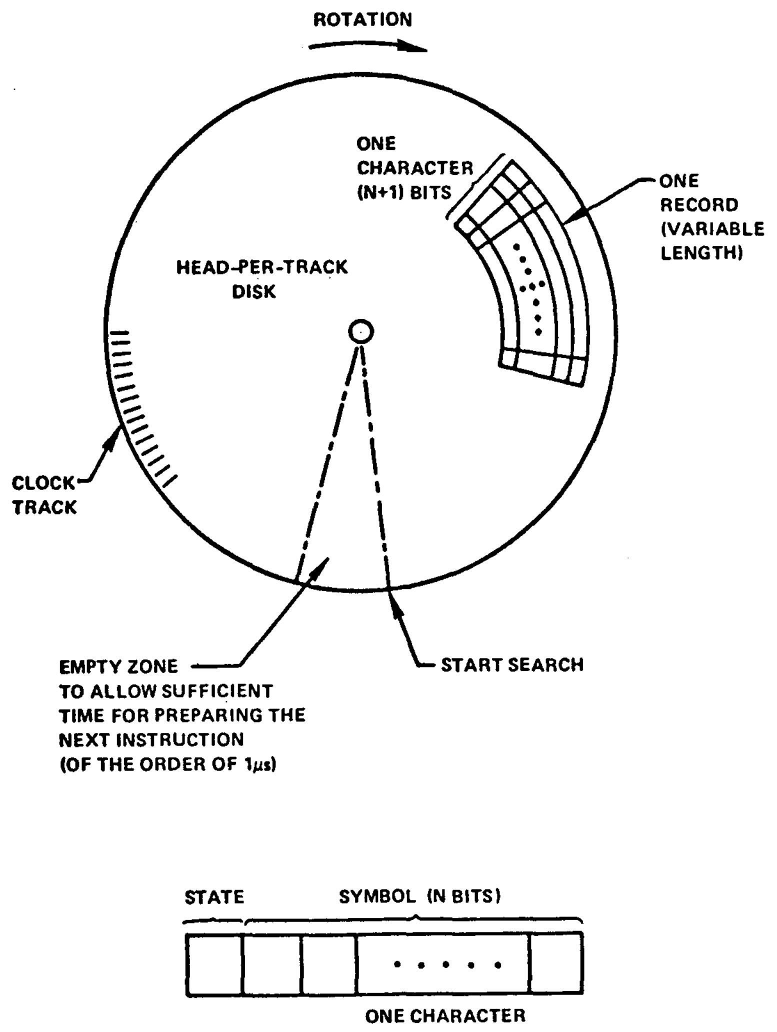
ROTATION
ONE CHARACTER (N+1) BITS
ONE RECORD (VARIABLE LENGTH)
HEAD-PER-TRACK DISK
CLOCK TRACK
EMPTY ZONE TO ALLOW SUFFICIENT TIME FOR PREPARING THE NEXT INSTRUCTION (OF THE ORDER OF 1μs)
START SEARCH
STATE
SYMBOL (N BITS)
ONE CHARACTER

Figure 2. Storage of Characters and Records

which is represented by a binary vector of length N, and q $\varepsilon\{0,1\}$ is a control symbol (flag) associated with s which is called its state (Figure 2). Setting the state of a symbol s means making the corresponding q equal to 1.

To be able to represent our record structures as a symbol string, we need some special symbols to act as delimiters. Table I shows these special symbols along with their definitions and APL/360 equivalents used in our simulation. Figure 3 shows a possible record format utilizing these symbols. The length field gives the length of the corresponding record in characters. We assume that this field has a fixed length itself in order to allow symbol-by-symbol comparison of the record length to a given number for the purpose of inserting a new record. The flag field contains three characters; two for holding the results of searches (scratchpad storage), and one which contains a record type flag (ε or υ). An ε indicates that the record has been marked for deletion (i.e., that it is empty) while a υ denotes a non-empty record. A don't-care symbol δ is provided which can be used whenever the exact value of a symbol is not known (e.g., the middle initial of an author). In the search instructions to be described, δ satisfies any search criteria. The symbol τ can be used as a temporary substitute for other symbols during processing. It is normally replaced by the original symbol at the termination of processing. Finally, β denotes the blank symbol.

Some string-manipulating instructions which can be easily implemented on RAPID and which have been found to be useful for information retrieval are given in Table II. As an example of the utility of this instruction set, the simple program of Table III marks any record (sets the flag of the last character) containing in its title field, designated by the name 'TI', a word having 'magnet' as its first six characters and having three to ten non-blank characters after that.

Of course, in a practical system, the end user is not expected to write programs in terms of these instructions. A high-level language may be developed which allows simple expression of common user queries and which can be easily translated into the machine language of RAPID. However, here we are only interested in verifying the correctness of the system's architecture and, therefore, will not deal with these software problems.

TABLE I

Reserved Symbols and Their APL/360 Equivalents

Symbol	APL	Description
λ	⎕	Indicates start of length field.
ρ	ρ	Indicates end of a record.
σ	⌈	Separates name and information subfields in a field.
ϕ	_	Indicates end of a field.
ε	ε	Designates the end of an empty record.
υ	⊤	Designates the end of a non-empty record.
δ	⌊	Is the don't-care symbol.
τ	~	Is used as temporary substitute for other symbols.

TABLE III

A Sample Search Program for RAPID

search for ϕTIσ
expand to ϕ
search for marked βmagnet
expand 10 or to β
contract 3
propagate to ρ
propagate 3
search for marked υ

TABLE II

Definition of an Instruction Set for RAPID

1. search and set s: Find all occurrences of the symbol s and set their states.

2. search for $s_1s_2...s_n$: Find all occurrences of the string $s_1s_2...s_n$ and set the states of the symbols which immediately follow s_n.

3. search for marked $s_1s_2...s_n$: same as the previous instruction except that for a string to qualify, the state of its first symbol must be set.

4. search for marked R s: Search for symbols whose states are set and have the relation R with s. Then, set the state of the following symbol. Possible relations are $<$, $\leq$, $>$, $\geq$, and $\neq$.

5. propagate to s: If the state of a symbol is set, reset it and set the state of the first s following it.

6. propagate i: If the state of a symbol is set, reset it and set the state of the i-th symbol to its right.

7. expand to s: If the state of a symbol is set, set the states of all symbols following it up to and including the first occurrence of the symbol s.

8. expand i: If the state of a symbol is set, set the states of the first i symbols to its right.

9. contract i: If the state of a symbol is reset, reset the states of the first i symbols to its right.

10. expand i or to s: If the state of a symbol is set, either set the states of the next i symbols or set the states of the symbols following it up to and including the first occurrence of the symbol s, whichever comes first.

11. add s: Add the numerical value of s to the numerical value of any symbol whose state is set.

12. replace by s: If the state of a symbol is set, replace the symbol by s.

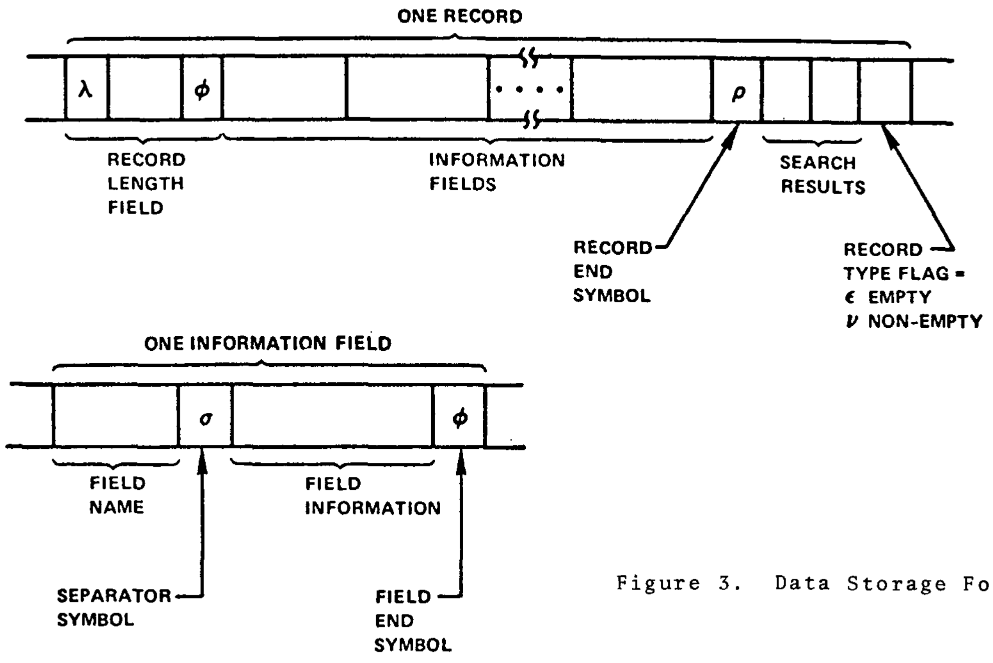

Figure 3. Data Storage Format

THE APL/360 SIMULATION OF RAPID

The simulation to be described in this section has been performed at the instruction level. The string of symbols is denoted by the vector S and the corresponding state vector which consists of zeros and ones by Q. The two vectors S and Q are of the same length. A vector SYM defines the correspondence between symbols in the alphabet X and natural numbers for the purposes of comparison and arithmetic operations; i.e., SYM [i] corresponds to the integer i (index origin is zero).

Table IV shows the APL/360 definitions of RAPID instructions. In these definitions, A and I denote character-string and integer arguments, respectively. Note that there are two dyadic functions SEARCHRELATIVETO and EXPANDORTO. These are equivalent to the fourth and tenth instructions in Table II, respectively.

Table V shows some examples of the execution of these instructions in a reference retrieval system. In these examples, a simple S vector is used in which the field names are to be interpreted as follows:

A	Author
T	Title
Y	Year of publication
M	Month of publication
S	Source
P	Publisher
C	City of publication

For example, the first record in S has a total length of 99 characters. It represents a paper with two authors, 'C.G.BELL' and 'M.W.PIRTLE', entitled 'TIME-SHARING BIBLIOGRAPHY'. It was published in December of 1966 in 'PROCEEDINGS OF THE IEEE'. Of course, other information such as volume number, issue number, and page numbers could be added to the record. One important feature of the system is that the fields need not appear in any particular order since each field is uniquely labeled. In addition, the number of fields and the length of each field is completely variable. As a second example, the last record has a length of 113 characters. It represents a paper by 'J.G.TRYON' entitled 'QUADDED LOGIC' which appeared in a book called 'REDUNDANCY TECHNIQUES FOR COMPUTING SYSTEMS' published in 1962 by 'SPARTAN' in 'WASHINGTON'.

The programs in Table V are self-explanatory. Each one of them leaves the symbol string S unchanged (except possibly for the scratchpad storage areas consisting of two symbols immediately following the symbol ρ) and marks the state vector Q (sets some of its elements to one) in the positions corresponding to the last characters of the records which satisfy the search criteria, so that these marked records can be recognized in a readout operation. Essentially, the RAPID system acts as a filter between a mass storage device (disk) and the central processor. By marking and thus preselecting a small set of records which are then processed by the CPU, substantial savings in CPU time can be accomplished.

CONCLUSION

It should be evident from these pages that APL is a very helpful tool in the initial verification of a digital system design at an architectural level. It is particularly convenient for parallel processing systems because of its ability to manipulate large arrays of data with simple instructions (3), (4). Adding new features can increase its usefulness even further. For example, in digital system designs, frequently instructions are constructed from components coming from various sources. In such cases, it would be helpful to have a feature which allows

TABLE IV

APL/360 Definition of RAPID Instructions

```
      ∇ SEARCHANDSET A
[1]     Q←(S=A)∨S='⌊'
      ∇

      ∇ SEARCHFOR A;I
[1]     A←(¯1+(I←0)+⍴A,'1')⍴A
[2]     Q←0,¯1↓(Q∨I=0)∧(S=A[I])∨S='⌊'
[3]     →((I←I+1)<⍴A)/2
      ∇

      ∇ SEARCHFORMARKED A;I
[1]     A←(¯1+(I←0)+⍴A,'1')⍴A
[2]     Q←0,¯1↓Q∧(S=A[I])∨S='⌊'
[3]     →((I←I+1)<⍴A)/2
      ∇

      ∇ P SEARCHRELATIVETO A;X;Y
[1]     X←SYM⍳S
[2]     Y←SYM⍳A
[3]     Q←0,¯1↓Q∧((P∊'<≤≠')×X<Y)∨((P∊'>≥≠')×X>Y)∨((P∊'≤≥')×
        X=Y)∨S='⌊'
      ∇

      ∇ PROPAGATETO A;T
[1]     Q←(Q∧S=A)∨0,¯1↓Q∧S≠A
[2]     →(∨/Q∧S≠A)/1
      ∇

      ∇ PROPAGATE I
[1]     Q←(I⍴0),(-I)↓Q
      ∇

      ∇ EXPANDTO A;T
[1]     T←Q
[2]     T←Q∨0,¯1↓(Q←T)∧S≠A
[3]     →(∨/T≠Q)/2
      ∇

      ∇ EXPAND I;T
[1]     T←1
[2]     Q←Q∨0,¯1↓Q
[3]     →((T←T+1)≤I)/2
      ∇

      ∇ CONTRACT I;T
[1]     T←1
[2]     Q←Q∧0,¯1↓Q
[3]     →((T←T+1)≤I)/2
      ∇

      ∇ I EXPANDORTO A;T
[1]     T←1
[2]     Q←Q∨0,¯1↓Q∧S≠A
[3]     →((T←T+1)≤I)/2
      ∇

      ∇ ADD I
[1]     S←SYM[(⍴SYM)|(SYM⍳S)+Q×I]
      ∇

      ∇ REPLACEBY A
[1]     S←SYM[(Q×SYM⍳A)+(~Q)×SYM⍳S]
      ∇
```

TABLE V

A Sample Simulation Session

```
    ⍝  THE FOLLOWING IS AN EXAMPLE FOR THE USE OF RAPID
    ⍝  INSTRUCTIONS.

    ⍝  THE ORDERING OF THE SYMBOLS IS GIVEN BY:

    SYM
0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ ()[]+-×÷=,;.:?'∘/|\⎕ρ
    ⌈_∊⊤⌊~

    ⍝  WE HAVE CONSTRUCTED A SIMPLE S VECTOR CONSISTING
    ⍝  OF FOUR RECORDS.

    S
⎕099_A⌈C.G.BELL_A⌈M.W.PIRTLE_T⌈TIME-SHARING BIBLIOGRAPHY_Y
    ⌈1966_M⌈12_S⌈PROCEEDINGS OF THE IEEE_ρ00⊤⎕108_A⌈C.G.
    BELL_A⌈A.NEWELL_T⌈COMPUTER STRUCTURES: READINGS AND
    EXAMPLES_Y⌈1971_P⌈MCGRAW-HILL_C⌈NEW YORK_ρ00⊤⎕096_A⌈
    T.KOHONEN_T⌈CORRELATION MATRIX MEMORIES_S⌈IEEE TRANS
    ACTIONS ON COMPUTERS_Y⌈1972_M⌈04_ρ00⊤⎕113_A⌈J.G.TRYO
    N_T⌈QUADDED LOGIC_S⌈REDUNDANCY TECHNIQUES FOR COMPUT
    ING SYSTEMS_P⌈SPARTAN_C⌈WASHINGTON_Y⌈1962_ρ00⊤

    ⍝  THE FOLLOWING PROGRAM MARKS ALL THE RECORDS WHOSE
    ⍝  AUTHOR IS C.G.BELL:

    SEARCHFOR 'A⌈C.G.BELL'
    PROPAGATETO 'ρ'
    PROPAGATE 3

    ⍝  TO SEE WHICH RECORDS HAVE BEEN MARKED, WE FORM

    Q/⍳416
98  206

    ⍝  THESE CORRESPOND TO THE LAST SYMBOL (FLAG) FOR
    ⍝  FIRST AND SECOND RECORDS.

    ⍝  A MORE INTERESTING EXAMPLE IS TO FIND ALL THE
    ⍝  RECORDS WITH MULTIPLE AUTHORS.

    SEARCHFOR '_A'
    REPLACEBY '~'
    SEARCHANDSET '⎕'
    PROPAGATETO '~'
    REPLACEBY '⌈'
    SEARCHFOR 'ρ'
    REPLACEBY '1'
    SEARCHANDSET '~'
    PROPAGATETO 'ρ'
    PROPAGATE 1
    ADD 1
    SEARCHANDSET '~'
    REPLACEBY '⌈'
    SEARCHFOR 'ρ20'

    ⍝  TO SEE WHICH RECORDS HAVE BEEN MARKED, WE FORM

    Q/⍳416
98  206

    ⍝  AGAIN, THE FIRST TWO RECORDS HAVE RESPONDED.

    ⍝  AS A FINAL EXAMPLE, CONSIDER FINDING REFERENCES
    ⍝  TO ALL DOCUMENTS PUBLISHED AFTER 1963.

    SEARCHFOR 'Y⌈197'
    PROPAGATETO 'ρ'
    REPLACEBY '~'
    SEARCHFOR 'Y⌈196'
    '≥' SEARCHRELATIVETO '3'
    PROPAGATETO 'ρ'
    REPLACEBY '~'
    SEARCHANDSET '~'
    REPLACEBY 'ρ'
    PROPAGATE 3

    ⍝  TO SEE WHICH RECORDS HAVE RESPONDED, WE FORM

    Q/⍳416
98  206  302

    ⍝  THESE NUMBERS CORRESPOND TO THE LAST SYMBOL OF
    ⍝  THE FIRST, SECOND, AND THIRD RECORDS.
```

APL expressions to be constructed in a similar manner and then executed. For example, if we had a monadic operator, Σ, which causes the execution of an APL statement given to it as a character-string argument, the definition of the SEARCHRELATIVETO instruction could be simplified to:

```
  ∇P SEARCHRELATIVETO A
[1] Σ'Q←0,¯1↓Q∧((SYM⍳A)',P,'SYM⍳S)=''⌊'''
  ∇
```

A more detailed simulation is necessary before an actual implementation is attempted. Such a simulation will not be feasible with APL because of the unacceptably slow execution and high storage requirements for data sets of practical sizes.

ACKNOWLEDGEMENT

Parts of this research were performed while the author was with the Computer Science Department, University of California, Los Angeles, California 90024, U.S.A., with support from the U.S. National Science Foundation, Grant GJ 33007X.

REFERENCES

(1) Parhami, B., "A Highly Parallel Computing System for Information Retrieval," AFIPS Conference Proceedings, Vol. 41 (1972 Fall Joint Computer Conference), AFIPS Press, Montvale, N.J., U.S.A., pp. 681-690.

(2) Parhami, B., "Design Techniques for Associative Memories and Processors," Technical Report UCLA-ENG-7321, Computer Science Department, University of California, Los Angeles, California, U.S.A., March 1973, Chapter 3 (NTIS No. PB-220714).

(3) Foster, G. H., "APL as a Descriptive Language for Associative Processing and Systems Design," Proceedings of the APL Congress 73, Copenhagen, Denmark, North Holland Publishing Company, Amsterdam, 1973, pp. 133-140.

(4) Parhami, B., "Storing Extensible Tables in Associative Memories with Fixed Word-Lengths," Proceedings of the Seventh Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, California, U.S.A., November 1973, pp. 439-443.