RETRIEVAL OPERATIONS AND DATA REPRESENTATIONS IN A CONTEXT-ADDRESSED DISC SYSTEM

Stanley Y.W. Su, George P. Copeland, Jr. and G. Jack Lipovski
University of Florida, Gainesville, Florida

ABSTRACT

This paper attempts to demonstrate that simple expansion of the processing capabilities of fixed disc read and write heads can avoid the multi-level mappings from high-level retrieval language to machine language and from user oriented data representation (information structure) to machine oriented data representation (storage structure) which are found necessary in conventional von Neumann computers. The processing capabilities built in the disc read and write heads for each disc track allow information files to be segmented and data items stored on all segments to be searched, modified, inserted, deleted, rearranged and rewritten simultaneously as a set of discs are rotating. Information structures such as the network structure, the hierarchical or tree structure and the relational structure are discussed and their implementations and basic search operations in the disc system are described.

1. Introduction

Most existing information systems are implemented on general purpose von Neumann type computers. Von Neumann processors have serious inherent limitations when they are applied in non-numerical information processing. We shall discuss some of the limitations with respect to both retrieval language and storage of data in information systems.

In every information system, a retrieval language is designed to facilitate the user's access to the data stored in the data base. The language can be very close to the set of basic data manipulation functions constructed for searching, inserting, deleting, rearranging, etc., operations. It can also be a high level language which is then translated into the set of basic data functions. Recent works (Kellogg 1971, Dostert and Thompson 1972, Lefkovitz 1969, and Codd 1971) favor the latter approach of using natural language or other high-level retrieval languages for the obvious reason that it is easier for the user to use. However, the price that has to be paid is the inefficiency introduced by many levels of language mapping from the high-level retrieval language to the machine language level. Generally, the high-level retrieval language is first translated into some type of intermediate languages such as calculus statements or procedural statements which invoke a set of retrieval functions implemented in a programming language. The retrieval functions are in turn compiled or assembled into machine language. This time-consuming multi-level language mapping is necessary because a computer processor can only recognize the set of basic operations such as load register, shift register and store. Unfortunately, basic operations of von Neumann processors are very different from the basic operations of non-numerical information processing. If the basic operations of non-numerical processing can be implemented directly in hardware, the gap between the retrieval language and the machine can be closed up. Greater processing efficiency can then be achieved.

The other limitation of von Neumann processors can be shown in dealing with the issue of information representation. It is now widely accepted that the provision of data independence is one of the major objectives of a data base system (CODASYL report 1971a-b, Engles 1970, and Guide/Share report 1971). The user of a data base system should not have to be concerned about all aspects of data representations (for example, how data is physically recorded and how data is logically connected for efficient access) and access strategy. The information structure at the user's level is quite different from the data structure designed for efficient data access which is the data representation at the access path level. The data structure is then implemented and mapped into machine dependent storage structure. These three levels of data representation are found to be essential in the design of a file system (Wang and Lum 1971). At the access path level of design, pointers, cross reference indexes and inverted records are often introduced and incorporated into the data structure in order to speed up data access. These are extra data which require extra storage, access time and processing time for their construction and maintenance. Updating of the data when new information is introduced into the system is the most time-consuming and troublesome process.

Moreover, these pointers can have (software) errors. Such errors are hard to check because they are designed to be opaque to the user, who may be entering the data. This misinformation has led to lost accounts and other horrors. In order to avoid such errors, confirmation of the data is required in data acquisition. This is a costly operation. If data were in a natural format, without added pointers, the cost of data acquisition and confirmation, which is by far the largest cost of the computer operation, could be sizably reduced. And the data would be more reliable.

Transformation of data from user's information structure level to access path level and then to physical level is done to regain some efficiency which is lost in the use of conventional von Neumann processors. Data is modified so that only a small amount has to be searched. But as we noted above, this leads to lower reliability and efficiency. This multi-level data mapping would be unnecessary if data can be stored in a form very close to the information structure as viewed by the user, and searched by hardware directly without data transformation. Closing the gap between information structure and storage structure will also ease the problems dealing with data base translation (Sibley and Merton 1972) and data sharing in computer networks (Roberts, et. al., 1970) since complex data structure at the access path level is eliminated.

This paper describes the application of a context addressed disc system designed to perform content as well as context searches on a set of discs simultaneously. Data is organized on discs in a form very close to the various suggested information structures. Records of a file are stored on disc tracks each of which has a read and a write head with considerable processing capabilities. The disc system is capable of performing data manipulation functions on a disc independent of the central processor using the discs. This paper intends to demonstrate that changing the architecture of a popular secondary storage device such as a disc may alter the picture and issues of retrieval language, information representation, data reliability of information retrieval systems. The second section outlines the design of a disc system with associative processing capability. The third section discusses the problems of associative storage and retrieval with respect to information representations and retrieval operations. The fourth section illustrates the implementation of hierarchical structures, relational structures and network structures on discs and disc retrieval operations. A summary is given in the last section.

2. The Architecture of a Disc System with Associative Processing Capability

We shall first familiarize the reader with the basic construction and operations of a disc system designed for handling large data bases. In the following description, we shall give only the architectural features and operations which are related to the discussion on retrieval language and data representation to be presented in the next section. A detailed description of the hardware and various sophisticated content and context search methods on discs will appear in a paper submitted by the authors to the 1st Annual Symposium on Computer Architecture to be held in December, 1973 (Copeland et. al. 1973). The paper will be made available to the attendants of the interface meeting.

The disc system described here follows the general concept of performing associative searches on discs presented by the previous works (Hollander 1956, Minsky 1972, Parhami 1972, Parker 1971, Fuller et.al. 1965, Healy et.al. 1972). However, the system is designed to do more sophisticated string, tree, set and directed graph searches as well as hardware garbage collection.

The disc system in its design consists of a set of fixed head discs with a read head and a write head per disc track. The set of discs can be visualized as a memory device containing a long string of bits in which files are stored. The disc tracks physically break the files into segments. Each segment (corresponds to a disc track) may contain only a part of a record, a whole record, or several records of a file. Figure 1 shows the relationships among records, segments and disc tracks. Data is not accessed by a location number, but rather all addressing information (such as structural parameters and attributes) is stored in memory along with data. Comparison logic is used to search for the specified addressing information within memory. Memory is scanned from one end to the other while the comparison logic searches for the address.

As shown in Figure 2, a one bit wide random access memory is used as a marker memory for data items stored on each track so that items may be marked for further processing or input-output. A counter initially set to zero at the beginning of each disc revolution is used as the memory address register. If the beginning of each data item indicated that the counter was to be incremented (using a special delimiter bit or symbol), then the counter would point to a unique marker bit for each data item in a disc track. We have a 1-1 onto mapping of marker bits to data items. Furthermore, the mapping would be the same for each revolution of the disc since the counter is reset at the beginning of each revolution. Although a random access memory is being used, data items are not tied down to a location and items may be of variable length. Only their relative positions in the sequence are important. When insertions and deletions are made, space can be provided by delay hardware between the heads, and the contents of the one bit random access memory can be shifted forward or backward from the point of insertion. Thus storage allocation and garbage collection are neatly and efficiently handled in hardware for variable length data items.

The above brief description of the disc system is meant to give the reader some idea about the hardware for associative storage and retrieval. Much detail is purposely left out. The hardware for storage allocation and garbage collection on discs, error correction, instruction fetch and data modification is not shown in Figure 2. However, examples for data organization on discs and the search methods for trees, relational tables and directed graphs will be given in section 4.

3. Associative Storage and Retrieval

We shall now deal with two major problems of

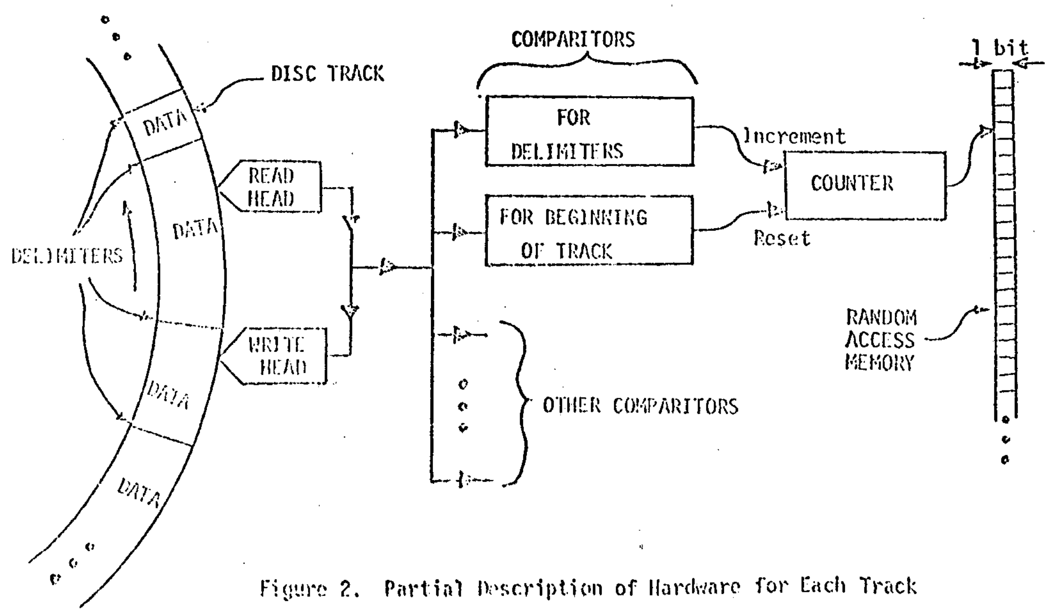

Figure 2. Partial Description of Hardware for Each Track

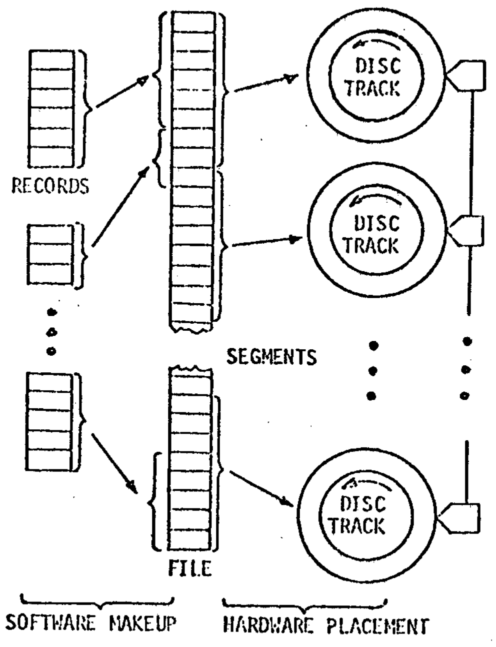

Figure 1. Storage of Records as Segments

associative storage and retrieval using the architecture of the disc system described in the preceding section as a test base. The first problem is to determine the data organization on discs which is a direct representation of the various information structures as the user sees them. Our goal is to close the gap between the physical storage structure and the information structures in order to avoid data mapping in various levels of data representation as described in the introduction.

The second problem is to determine the set of basic operations (or instructions) for the disc system which corresponds to the queries that the user of an information system uses to access data. Our goal is to find the set of high level retrieval queries which can be implemented as basic instructions to be carried out by the disc read-write heads efficiently and without excessively increasing the hardware implementation cost.

In non-numeric processing, several information structures have become useful in representing information. They are the directed graph or network model (CODASYL 1971a), the relational model (Codd 1970, 1971), and the tree or hierarchical structure, which is commonly used in data processing systems. Information is represented in each of these structures in general as a set (record) of attribute-value pairs in each node or table entry. These are called information structures because the user views his data as being displayed most naturally in these structures, and because operations of his data involve specifying parameters that are also parameters of the structures.

It has been suggested by the proponents of these models that the logical relations among data items of any record can be represented by the specific information structure they propose, and data of different structures can be mapped into the uniformed structure proposed. However, our view is that the user of an information system should be allowed to use all different information structures to describe the data they have. There are two reasons for taking this view. First, one information structure may be easier to describe a certain kind of data than the other. For example, a network structure is more suitable for describing data related to circuit design and semantic information processing of natural languages (Shapiro and Woodmansee 1969, Kay and Su 1970, Su 1971, 1972, 1973), and hierarchical structure is more suitable for describing library information since most libraries have adopted hierarchical indexing systems. Secondly, normalizing information representation involves data mapping from one structure to the other and may cause the loss of the original structural properties. For example, the tree structure is a special kind of graph. If information which can be conveniently represented by tree structure is mapped into a network model, the mapping operation will be necessary each time the user queries the data base. This translation is further complicated by the fact that the natural form of the query often involves structural parameters (such as level numbers) or search operations (such as the preorder, postorder, and endorder predecessor or successor functions) that are not defined for a general directed graph. This may require using a sophisticated and expensive data definition and management system with subsequent loss in reliability or forcing the user to think in terms of the new structure. Moreover, inefficiencies in search time may be introduced because information about the properties of a tree may be lost.

Therefore, our disc system is designed to implement the various information structure models without losing their structural properties, and content as well as context searches are performed directly on the various structures using a set of basic disc operations.

Just like the instruction set is to a computer processor, a set of basic disc operations need to be identified before the disc hardware can be implemented. The contents of this set would very much depend on the types of queries that will be used by the user of an information system to retrieve and manipulate the data base. The types of queries, in turn, depend on the information structure viewed by the user.

In order to see more concretely which operations are performed on the information structures of non-numeric processing, let us examine an example inventory file taken from J.C. Date (1972) in his tutorial description of Codd's work on relational files. The file involves a many-to-many mapping of suppliers and parts (each supplier supplies many parts and each part is supplied by many suppliers). Date lists four queries to be satisfied:

(a) find part numbers for parts supplied by supplier 2;
(b) find part names for parts supplied by supplier 2;
(c) find supplier numbers and status for suppliers in London;
(d) for each part find part number and names of all cities from which the part may be obtained.

E.F. Codd's (1970) normalized relational form of this file involves three subfiles in the form of tables as in Figure 3. Each table defines a relation with the domains of the relation shown as the headings of the columns. The supplier-part (SP) subfile shows how the many-to-many mapping is handled in the relational model using redundant data values to link subfiles by contents rather than by addresses. If the user is supplied with the skeletal description of the table arrangement as in Figure 4, he may specify each of the above queries in a non-procedural statement similar to that developed by Codd (1971b) and given in Date:

(a) SP.P# : SP.S# = 2
(b) P.PNAME :∃SP((SP.P# = P.P#)∧(SP.S# = 2))
(c) S.S#, S.STATUS : S.CITY = 'LONDON'
(d) P.P#, S.CITY :∃SP((P.P# = SP.P#)
∧(S.S# = SP.S#)), ∀P.P#

The data items in the above statements are specified by qualified names as those used in COBOL or PL/1. The expression on the left of the colon indicates what is to be retrieved and the expression on the right is a qualification. For

Supplier (S) subfile

attribute record (set)

value records (sets)

S#	SNAME	STATUS	CITY
1	Smith	20	London
2	Jones	10	Paris
3	Blake	30	Paris
4	Clark	20	London
5	Adams	30	Athens

Part (P) subfile

attribute record (set)

value records (sets)

P#	PNAME	COLOR	WEIGHT
100	nut	red	12
200	bolt	green	17
300	screw	blue	17
400	screw	red	14
500	cam	blue	12
600	cog	red	19

Supplier - part (SP) subfile

attribute record (set)

value records (sets)

S#	P#	QTY
1	100	3
1	200	2
1	300	4
1	400	2
1	500	1
1	600	1
2	100	3
2	200	4
3	300	4
3	500	2
4	200	2
4	400	3
4	500	4
5	500	5

Figure 3. Inventory file in Codd's normalized relational form.

Inventory file

S (S#, SNAME, STATUS, CITY)

P (P#, PNAME, COLOR, WEIGHT)

SP (S#, P#, QTY)

Figure 4. Skeletal description of file in Codd's normalized relational form.

example, the first statement is a query for retrieving all part numbers (P#) supplied by supplier number 2 (S#=2).

It has been demonstrated by Codd (1970) and described in Date (1972) that information represented by hierarchical and network structures can be represented by the type of normalized relational form shown in Figure 3 and the basic operations necessary to search and retrieve data from the relational data tables are called "projection" and "join" operations. These two operations are defined as follows:

Projection: "To project a relation over specified domains, we strike out the domains (columns) not required (not specified) and remove redundant duplicate tuples (rows) from what remains."

Join: "Two relations with a common domain, D say, can be joined over that domain. The result is a relation in which each tuple consists of a tuple from the first relation concatenated with a tuple from the second D-value (except that we eliminate one of the two identical D-values)."

It was suggested (Codd 1971) that statements of the type shown above (a-d) can be translated into a sequence of operations consisting of joins and projections. For example, the relational calculus statement (C) can be translated into the following sequence of operation as shown by Date. Note that the additional relation R of Figure 5 is necessary.

Join S and R over CITY
Project the result over (S#, STATUS)

R

CITY
LONDON

Figure 5. Relation Table R

From the above discussion of Codd's and Date's works, many interesting points need to be noted. First, Codd's relational data tables express the structural relationship among data items by using redundant domains in the relational tables. For example, the domain S# in S and SP relational tables and P# in SP and P tables. The cross references are specified by content rather than by the conventional approach of using addresses or indexes. This information structure can be implemented very efficiently in the associative processing disc system described because data is searched by contents rather than by addresses. The actual implementation of relational files on conventional machines will involve time-consuming operation of intersecting tables and building intermediate tables. This operation can be prohibitively costly if the data tables are large. But, on the disc system, the search operation involves only the marking and reading out of the marked data items by all disc read/write heads simultaneously without the intervention of the CPU. The size of tables has very little effect on the processing time. Secondly, the examples show that the queries formulated by the user depend heavily on the information structure as seen by the user. The proposed relational calculus statements are queries which can be implemented as the basic operations of the disc system. This will be illustrated in the next section.

4. Implementation of Information Structures

In this section we shall describe the implementation of the various information structures widely used. The organization of data and the search operations in the disc system will be described.

4.1 Trees or Hierarchical Structures

Information is represented in a tree or hierarchical structure as a set, record or tuple of attribute-value pairs in each node of the tree. A tree can be linearized in several ways (Knuth Vol. I 1969). If the tree is written in preorder with level numbers included with each node, then it is uniquely specified. Also, for a given node at level 1, its ancestor at level K (k<1) is the last occurrence of a node with level k before reaching the given node. This provides a convenient method for marking backwards (up the tree) in the sequence. For example, an ancestor node can be marked if a search within one of its successors is successful. This can be done in the following way.

The ancestor is encountered first because of the preorder in which the tree is stored. The random access address of the ancestor mark bit can be saved for reference until the successor is searched later in the sequence. If the search is successful, then the ancestor mark bit can be set because its random access address was saved. With the tree stored in preorder together with level numbers, the above algorithm simply involves remembering the random access address of the last node at level k. If a particular member of the set or record within the ancestor node is to be marked, the random access address of the member can be saved in a similar manner. Marking forward (down the tree) is much simpler. If a decendent is to be marked whenever an ancestor is successfully searched, the only thing to be remembered is whether or not the search was successful. A similar communication can be used between set members of the same tree node.

Thus we have the capability of marking a node or node member if another node or node member satisfies a given condition. This can be done in one disc revolution if the communicating elements are along the same path of the tree. Although a random access memory is being used here, it is used strictly for internal hardware implementation of an instruction and is totally transparent to the user. The user speaks only in terms of his natural information structure. Now let us see how the hierarchical structure of Figure 6 can be stored and searched in the disc memory.

Items within a record are stored physically together as a unit with their level number. Special attribute records can be used to eliminate

Attribute Record set for level 1	S(S#	SNAME	STATUS	CITY)					
Attribute Record set for level 2					P(P#	PNAME	COLOR	WEIGHT	QTY)
	1	Smith	20	London					
					100	nut	red	12	3
					200	bolt	green	17	2
					300	screw	blue	17	4
					400	screw	red	14	2
					500	cam	blue	12	1
					600	cog	red	19	1
Value records (sets)	2	Jones	10	Paris					
					100	nut	red	12	3
					200	bolt	green	17	4
	3	Blake	30	Paris					
					300	screw	blue	17	4
					500	cam	blue	12	2
	4	Clark	20	London					
					200	bolt	green	17	2
					400	screw	red	14	3
					500	cam	blue	12	4
	5	Adams	30	Athens					
					500	cam	blue	12	5

Figure 6. Inventory file in a hierarchical form

```
Inventory file
    S(S#   SNAME   STATUS   CITY)

        P(P#   PNAME   COLOR   WEIGHT   QTY)
```

Figure 7. *Skeletal description of file in a hierarchical form.*

redundant storage of attributes. These attribute records are stored first, one for each level. Then the value records are stored in the preorder indicated by the vertical positioning of Figure 6. If the user is given the skeletal description of the file as in Figure 7, he may express each of the queries in a non-procedural statement similar to that developed by Codd:

(a) S.P.P# : S.S# = 2
(b) S.P.PNAME : S.S# = 2
(c) S.(S#, STATUS) : S.CITY = 'LONDON'
(d) S.(P.P#, CITY), $\forall$ P#

The qualification on the right is simplified by the fact that specific hierarchical dependencies are given in the expression on the left using parentheses. To satisfy query (b), two comparitors are needed. One searches for the item to be marked (S.P.PNAME) and the other searches for the condition that must be satisfied (S.S# = 2). As memory is scanned from one end to the other, both comparitors continuously search. When S.S# = 2 is satisfied, this information is remembered for the duration of the search of that subtree so that all PNAME's within the subtree can be marked. This is an example of forward marking. Backward marking is involved in the following query: all supplier names that supply part number 200. The non-procedural expression of this query is S.SNAME : S.P.P# = 200. Here one comparitor searches for S.SNAME. Whenever it is found, its random access address is remembered for the duration of the search of that subtree for S.P.P# = 200. If S.P.P# = 200 is satisfied, then the mark bit at the address that was remembered is set.

4.2 Tables, Graphs, and the Relational Data Structure

The tree hardware can also be used to implement tables like those of Figure 3 in section 3 by providing a means of communicating between elements within the tables. If a table is at tree level i, then each row (record) in the table is at level i+1. All data items in the same row are members of the same tree node. Also, several tables may be grouped hierarchically. A special attribute record can again be used to eliminate redundant storage of attributes.

General graphs or networks can be implemented by setting up a table for each node of the graph. Each table would contain the node names of nodes pointed to by the table node along with their corresponding relation or arc name. Alternatively, a table could be set up for each relation or arc name. Here each table would contain as rows the node name pairs that are connected by the table relation or arc. Also, each graph node may contain an additional set of data items which provide further information about a node or relation (arc). This set of data items can be included as table entries in the same manner as the node names or relation names. Figure 3 is an example of such a set of tables, where each table corresponds to a relation name. Communication between tables is more involved and time-consuming than communication between nodes of a tree, however two methods are described below that are still rather efficient.

The most obvious method is to pick up the node names to be traversed from the source table and use these names to context search the destination table. This method has the advantage of using only natural data pointers, but it can be very time-consuming if many names must be compared. The second method stores in the source table the random access addresses of the mark bits of the node names in the destination table. These random access addresses may be stored in another table column just as data is stored. The advantage of this method is that all marking between tables can be done in one revolution. The disadvantages are that time is needed to store the random access addresses and insertions and deletions are more time-consuming because pointers must be changed. Now let us look specifically at how the relational data structure of Figure 3 can be stored and searched in the disc memory.

Items within a record are again stored physically together as a unit with their tree level number. Attribute records are stored immediately before the value records which they describe. The value records within a table are stored in any order. Also, ordering of the tables themselves is not relevant. Query (a) is satisfied in the same way as before since only one table (SP) is involved. Query (b) involves relating two tables (SP and P). During the first disc revolution, a comparitor searches for SP.S# = 2. Whenever this is satisfied, the SP.P# in the same row is marked. These values can be picked up on the second revolution and used to search for P.P# in the third revolution. During this third revolution, a second comparitor searches for P.PNAME. Whenever the P.P# search is successful, the P.PNAME of the same row is marked. If the number of items communicating between tables is large, many comparitors or many revolutions are necessary to make all the comparisons necessary. An alternative method is to store in the source table the random access address of the destination values. The procedure to implement query (b) using this scheme is as follows. The first revolution is the same. During the second revolution, whenever SP.P# values are found to be marked, the prestored random access addresses are used to immediately set the mark bits for items in the destination table. A third revolution marks PNAME's if P#'s are marked.

From the above description of the implementation of various information structures, one point becomes clear. The processing of hierarchical structures can be implemented directly on the disc and is more efficient than processing other structures on the disc processor. This is because the hierarchical structure can be linearized and the structural information can be stored with the data and be used in query statements formulated by the user. The process suggested by Codd of mapping all structures into the relational data structure is found to be an unnecessary step in our disc system, since, as we have shown above, various information structures may be implemented as they are and all data items in any structure can be used directly as a search key.

5. Summary

We shall summarize this paper in the following central points:

(1) It is desirable to close the gap between physical structure and the information structure as viewed by the user in order to avoid multi-level data mapping which reduces processing efficiency and data reliability. The expansion of disc read/write head processing capability shown in this paper allows various information structures to be stored and processed without many levels of data mappings.

(2) It is desirable to have the hardware carry out the basic high-level retrieval functions required in information systems. The disc system described in this paper is capable of reading, modifying, inserting, deleting and rewriting data and collecting unused cells on discs without the intervention of the CPU using the disc system. Since data is searched directly on secondary storage, excessive paging of data in and out of the main memory is eliminated.

(3) Information systems often require the processing of data in large data bases. Processing operations to be carried out on conventional von Neumann processors are often time-consuming. The proposed disc system allows files to be physically segmented and the segments searched and processed simultaneously over all disc tracks. Thus, the disc system offers a cost-effective means of implementing the associative processing technique.

(4) Different information structures are suitable for representing different types of data as viewed by the user. Thus, the user should be allowed to use the structures as they are without having to transform the data into a normalized structure as proposed in the relational model. The example given in section 4 shows that hierarchical or tree structure can be handled more efficiently in the disc system than by transforming the structure into its equivalent relational data tables.

BIBLIOGRAPHY

1. CODASYL Systems Committee: 'Introduction to "Feature Analysis of Generalized Data Base Management Systems"', CACM 14,5 (May 1971b), pp. 308-318.

2. Codd, E.F., "A Relational Model of Data for Large Shared Data Banks," CACM 13,6 (June 1970) pp. 377-387.

3. Codd, E.F., "Normalized Data Base Structure: A Brief Tutorial," Proceedings of ACM SIGFIDET Workshop on Data Description, Access and Control, (Nov. 1971a), pp. 1-17.

4. Codd, E.F., "A Data Base Sublanguage Founded on the Relational Calculus," Proceedings of ACM SIGFIDET Workshop on Data Description, Access and Control, (Nov. 1971b), pp. 35-68.

5. Copeland, G.P., Lipovski, G.J., and Su, Stanley Y.W., "The Architecture of CASSM: A Cellular System for Non-numeric Processing," paper submitted to the First Annual Symposium on Computer Architecture, December, 1973.

6. Data Base Task Group of CODASYL Programming Languages Committee: Report, April, 1971a.

7. Date, C.J., "Relational Data Base Systems: a Tutorial," paper presented at the Fourth International Symposium on Computer and Information Sciences, December 1972.

8. Dostert, B.H. and Thompson, F.B., "The REL System," paper presented at the Fourth International Symposium on Computer and Information Sciences, December 1972.

9. Engles, R.W., "A Tutorial on Data Base Organization," IBM Technical Report TR 00.2004, IBM, Poughkeepsie, N.Y., March, 1970.

10. Fuller, R.H., Bird, R.M. and Worthy, R.M., "Study of Associative Processing Techniques," AD-621516, August, 1965.

11. Guide/Share Data Base Task Force, "Data Base Management System Requirements," Share, Suite 750, 25 Broadway, NY, November 1971.

12. Healy, L.D., Doty, K.L. and Lipovski, G.J., "The Architecture of a Context Addressed Segment Sequential Storage," Proceedings of FJCC, Vol. 41, part I, 1972, pp. 691-702.

13. Hollander, G.L., "Quasi-Random Access Memory Systems," Proceedings of EJCC, 1956, pp. 128-135.

14. Kay, M. and Su, Stanley, Y.W., "The MIND System: The Structure of the Semantic File," RM-6265/3-PR, The RAND Corporation, Santa Monica, California, 1970.

15. Kellogg, C., Burger, J., Diller, T. and Fogt, K., "The Converse Natural Language Data Management System: Current Status and Plans," Proceedings of the Symposium on Information Storage and Retrieval, April, 1971, pp.33-46.

16. Knuth, D.E., Fundamental Algorithms, Vol. 1, Addison-Wesley, 1969.

17. Lefkovitz, D., File Structure for On-line Systems, Spartan Books, 1969.

18. Minsky, N., "Rotating Storage Devices as Partially Associative Memories," Proceedings of FJCC, Vol. 41, Part I, 1972, pp.587-596.

19. Parhami, B., "A Highly Parallel Computer System for Information Retrieval," Proceedings of FJCC, Vol. 41, Part I, 1972, pp. 681-690.

20. Parker, J.L., "A Logic per Track Retrieval System," IFIP Congress, 1971, pp. 146-150.

21. Roberts, L.G. and Wessler, B.D., "Computer Network Development to Achieve Resource Sharing," AFIPS Conference Proceedings, May 1970, pp. 543-549.

22. Shapiro, S.C. and Woodmansee, G.H., "A Net Structure Based Relational Question Answerer: Description and Examples," Proc. Int. Jt. Conf. Ant. Intel., Washington, D.C.,1969, pp.325-345.

23. Sibley, E.H. and Merten, A.G., "Transferability and Translation of Programs and Data," paper presented at the Fourth International Symposium on Computer and Information Sciences, December 1972.

24. Su, Stanley Y.W., "Managing Semantic Data in an Associative Net," Proceedings of the Symposium on Information Storage and Retrieval, April, 1971, pp. 105-116.

25. Su, Stanley Y.W. and Griffith, J., "Text Analysis and Associative Retrieval," Invited paper presented at the First Annual Creativity Center Consortium Workshop and Computer Future Applications Seminar, November, 1972.

26. Su, Stanley Y.W., "A Technique for Automatic Text Analysis and Associative Retrieval and Its Application to Drug Counseling and Education," presented at the Computer Science Conference, Columbus, Ohio, 1973.

27. Wang, C.P. and Lum, V.Y., "Quantitative Evaluation of Design Tradeoffs in File Systems," Proceedings of the Symposium on Information Storage and Retrieval, April,1971.

QUESTIONS

Richard E. Nance:

Can you very briefly describe what additional requirements this will make on languages or what additional capabilities it will give system designers in terms of languages?

Copeland:

As I illustrated, since the disk performs very high level search functions, we feel that it would be quite easy to map from a natural language type statement of something similar to SQUARE to the type of basic function that is needed to be performed. Certainly, we will need some type of translator or compiler to map from the query language into the basic operations. But I believe the associative disk will remove several of the levels of mapping and translation that now exist, and our mapping will be a much simpler one. We really have not thought too much about the language that the user might employ.

Edward M. McCreight:

Do you have any idea at this point how much the hardware for each additional head on the disk will cost?

Copeland:

We don't know exactly how much the cost will be, but the LSI technology seems to be reducing costs very quickly. Each year the cost of the same amount of hardware on an LSI chip is reduced by a factor of two. The machine that we are talking about is not only designed to facilitate the software end but is designed to fit the hardware technology using bulk storage as our only storage medium, and the only logic used is one-chip type used many times. This is the cheapest way you could build hardware.

Dennis E. Huaman:

Has this hardware been implemented yet?

Su:

No, it has not. We have finished the architectural design, and we are seeking support to build a prototype.

Dennis E. Huaman:

Would you envision the hardware keeping account of the number of "hits" on a search?

Copeland:

Yes, we can do that by simply implementing a counter recording the results of the search.

Huaman:

What about the use of Hamming codes for error detection and correction?

Copeland:

We plan on using an error correction capability within the cell logic as we read and write.

Huaman:

When the system is working, could you queue the queries with this king of hardware approach, or would you need additional random access memory, or would you need a read/write memory?

Copeland:

I think initially the machine will operate with a small mini-computer as the front end feeding instructions to the disk.

Huaman:

You mean a minicomputer driving the disk controller?

Copeland:

Yes.

Huaman:

Don't you think that will take more time as the "mini" processes requests going from the main computer to the disk?

Copeland:

No, I mean to bypass the main computer completely. The minicomputer will serve as the total front end.

Leo A. Bellew:

It seems that there are some similarities between what you are doing and paging techniques. For example, the use of the RAM to mark things has some relationship to marking the last page accessed. I am wondering if you dug far enough, you might find that taking specific things and placing them in a cache memory might not be equivalent to what you are doing. In fact, you might find that you have come out with paging from a different angle.

Su:

In the case of paging, what you are doing is to isolate certain segments of a random access memory in order to perform efficient searching. But our disk system is geared for very large data bases, and we are avoiding the memory hierachy type of approach by accomplishing search of a large data base in parallel. By the way, are you thinking about conventional paging schemes or the use of associative memories for paging? The GE 645 uses an associative memory attached to the segmentation mechanism to do an associative search of the segment and the page number to find the physical block. The idea here is to search simultaneously for all entries. I am afraid I do not see a very close relationship, however, between what we are doing and that type of thing.

DISCUSSION SESSION

Margaret B. Webster:

I was not quite clear about the difference between the "segment" and the "track". Is a "segment" a concept?

Su:

You might consider a segment as being a disk track, that is if we use a disk for implementation. This segment is a physical record, where you could have several physical records on a disk track.

Margaret B. Webster:

You're proposing to have a read/write head per segment.

Su:

That is correct. The crucial question here is the disk synchronization, i.e. being able to read and write properly on the same track.

A. A. Brooks:

With respect to the question of cost, about four years ago Jack [Lipovski] gave a talk on the cost of associative memories. At that time the cost of a large memory was approximately two to three times the cost for the von Neumann type of machine. Is there any more current estimate of the order of magnitude of the extra cost for this hardware?

Su:

The talk to which you refer is one in which Jack proposed to construct a cellular automata. He has come to the conclusion that they are simply too expensive. The real cost is in the logical control, and our indications are that this can be fitted into an LSI chip. These chips will essentially be identical.

Copeland:

Initially, the chip will cost several thousand dollars. But under mass production, the cost will probably go down to something around ten dollars.

A. A. Brooks:

Are we talking about ten dollars per track?

Su:

Yes, and what we are proposing is well within the hardware technology.

A. A. Brooks:

Basically, it would simply be the cost of adding this circuitry to a fixed head swapping disk?

Su:

That is right. But we should consider more than simply the hardware cost. Reduction in response time and the simplification of the software should be considerable. We are avoiding many levels of language mapping by making the hardware function a very high level one.

Tom Kibler:

First a comment. The IBM 1500 used one read head and one write head for refreshing their scope. It did perform read and write operations on the same track. My question involves the simplicity that your system will have when searching down a usual logical statement, i.e. expressed in terms of some set of keys. When you get to a query that involves both a key and a qualifier (a field that is not defined as a key), then your system requires another processor such that the disk can pull certain records but a sequential search must still be performed. Is that the way it would work?

Su:

Yes, certainly we will not be able to handle all types of retrieval operations. We shall seek to handle the most common and most frequent types. For sometime, as you describe, you will have to use the processor and search in the conventional way rather than utilizing the disk.

Copeland:

We have another paper that describes more fully the relationships between the cells. If you are interested in the hardware details, I would suggest that you read it.

Edward McCreight:

One of the most endearing qualities of the scheme is that it has such a wide cost/performance potential payoff. It would seem that with about twenty copies of your hardware gadget, you could search a IBM 3330 in about ten seconds.

Su:

You're suggesting using the same logic on many disks?

Edward McCreight:

No, I am simply suggesting that instead of the fixed head disk using a moveable head disk. In effect what you are doing is time-multiplexing your hardware over many tracks.

Copeland:

What we would prefer however would be the cheapest technology available duplicated many times so as to have one cell per track. This would make data base searches independent of data base size.

Leo Bellew:

Is your primary thrust to develop a piece of hardware, or is your primary effort to develop some primitives which are planted in the associative direction?

Su:

Our biggest concern is along the second line. We do not only wish to design a piece of hardware, but we wish to make it suitable to this particular type of application, i.e. information retrieval.

Leo Bellew:

So that it would not matter essentially what type of devices were being used; only that you could bring things into a memory that would allow you to do operations in parallel?

Su:

I do not believe that it would work on tapes, for the memory must be circular.

Leo Bellew:

Are you saying that the primitives you are developing are oriented toward a particular type of device?

Su:

It must be circular in nature; disk, drum, magnetic bubble memory, etc.

Leroy Lacy:

It does not apply to two separate heads?

Copeland:

Yes, the problem is that we find that it cost three times as much to put two heads on a track for they must be perfectly aligned. We shall be using every other track, and in one revolution, one track will be reading and the other will be writing. One further advantage of this scheme is that we have a complete copy of the data base both before and ofter the operation, and if errors occur we can do it over.

Margaret B. Webster:

Well, where will the logic chips be housed, in the controller?

Su:

Yes.

Copeland:

The actual logic chips can be placed anywhere you want them. We will probably put them on a rack somewhere close to the disk.

Webster:

Then I do not see how the concept can apply to the 3330 as was mentioned previously. I just do not see how it fits in here.

Su:

If I understand the questioner correctly, what he is saying is that you do not have to build a unit of logic per track. You can use movable heads rather than fixed heads, and he is considering the IBM 3330 rather as an example than a specific device.

Edward McCreight:

Yes, that is what I meant.

A. A. Brooks:

To really get the advantage from this, you must have multiple heads because the whole game is parallel processing. To move from track to track on a 3330-type device, I would suspect to be accomplished with the conventional hierarchical indices more quickly.

Edward McCreight:

I'm not so sure. The problem with 3330 is that it takes a long time to scan.

Tom Kibler:

I think the problem is very well stated in that if you have two or three 3330-type packs, you still have 60 searches in parallel. To accommodate the same kind of information capability, you would have to have a fixed head device with 6,000 independent logic units. I'm not sure that the 6,000 is the actual number, but it is something of that magnitude. I'm not sure that the cost differential between 60 and 6000 is worth it.

Copeland:

From the hardware point of view we can afford to be very generous with parallelism. The cost of the device is dependent on the square root of the number of units because of the learning curve property. So if we go from 1 to 100 we are only increasing the cost factor by about 10.

Kibler:

You prime your search process with some message from the main frame. Isn't it true that these different type of messages going out may cause problems?

Su:

No, that is not a problem. You give the instruction to the entire disk at one time.

Unidentified questioner:

How does this relate to the Parker disk and the other disk mentioned in your paper?

Copeland:

The major difference is that they handle only a very simple data type, e.g. name/value pairs. This takes a lot of software, and is very inefficient. Also, no "garbage collection" facility exists.

Donald Chamberlin:

What happens when a user wants to define a new table on the fly? Doe that invoke a procedure of initializing tracks?

Copeland:

We would mark a subset of the table rather than copy. We do not want to duplicate information. We do not create new tables from the existing tables.

Donald Chamberlin:

What if a user wished to define a new table that did not exist before? Would that be a complicated process?

Copeland:

It should be very simple because the storage structure is almost the same as the structure of the table. We would simply have to linearize the table.

Donald Chamberlin:

Then is there a catalog of the existing tables, so that when you wish to query you can check to see if the proper table is loaded into core?

Su:

Each table is identified by a name and is loaded into the disk. You search for tables by name. Both the table name and the data field that would go there would then be marked.

Donald Chamberlin:

Then is the data self-describing?

Copeland:

It is variable in records and each item is of variable length.

Tom Kibler:

What is the scheme for handling the updating of information on the disk? Suppose I submit a list of records to be updated and they come in from different disks. I change some and then they are to be restored. Do they return to their original location or to new locations?

Copeland:

We never remove the records, for we update on an item basis. We can access any item within a record.

Tom Kibler:

How do you do a selection that goes across several tracks?

Copeland:

I am afraid this is a little too difficult to explain with the time limitations.

A. A. Brooks:

Is this logically equivalent to the associative memory with the "and-rail" and "or-rail"? Are they functionally equivalent?

Copeland:

The cellular processor developed from the associative memory idea. And our scheme does involve rails so in that context it is drawn from the previous ideas.

Su:

It is sort of a compromise between the fully parallel processor and the existing sequential type. The emphasis in our case is on the logic per segment rather than per word. The concept of rails is utilized.

Stewart A. Schuster:

After the records meeting a particular query are marked, are they issued one at a time or do you have a work area and dump all of them into this area?

Su:

The marked records are transmitted one at a time.

Stewart A. Schuster:

Could there be a capability for working with the output from one query while resubmitting another query?

Su:

We do have a set of marked disks specifically for output purposes. It seems that you can handle all the requests of one person, but that you cannot begin work on a second person without the first being moved out to another storage area.

Copeland:

This problem is more of balancing the queueing of requests and output than of anything else. We can do that by having more than one marked output disk and queue them in the hardware itself.

Unidentified Questioner:

You keep proposing a disk, is it because a disk is cheaper? Would it not be just as reasonable to talk about drums?

Su:

We have really thought about disks, and what we would like to do is to take the concept to implementation using a disk. We feel that it is important to work from the software end also. Our approach is basically top-down, and we want to understand the problems and work toward a hardware supported solution.