Iowa State University
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

1981

Memory system for a relational database processor

Vijaya Kumar Konangi
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

Part of the Electrical and Electronics Commons

Recommended Citation

Konangi, Vijaya Kumar, "Memory system for a relational database processor " (1981). *Retrospective Theses and Dissertations*. 7439.
https://lib.dr.iastate.edu/rtd/7439

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

المنارة للاستشارات

www.manaraa.com

INFORMATION TO USERS

This was produced from a copy of a document sent to us for microfilming. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help you understand markings or notations which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure you of complete continuity.

2. When an image on the film is obliterated with a round black mark it is an indication that the film inspector noticed either blurred copy because of movement during exposure, or duplicate copy. Unless we meant to delete copyrighted materials that should not have been filmed, you will find a good image of the page in the adjacent frame. If copyrighted materials were deleted you will find a target note listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed the photographer has followed a definite method in "sectioning" the material. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For any illustrations that cannot be reproduced satisfactorily by xerography, photographic prints can be purchased at additional cost and tipped into your xerographic copy. Requests can be made to our Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases we have filmed the best available copy.

University Microfilms International

300 N. ZEEB RD., ANN ARBOR, MI 48106

8209139

Konangi, Vijaya Kumar

MEMORY SYSTEM FOR A RELATIONAL DATABASE PROCESSOR

Iowa State University PH.D. 1981

University
Microfilms
International 300 N. Zeeb Road, Ann Arbor, MI 48106

Memory system for a relational

database processor

by

Vijaya Kumar Konangi

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1981

TABLE OF CONTENTS

	Page
INTRODUCTION	1
REVIEW OF LITERATURE	4
OBJECTIVE AND OVERVIEW OF THE RESEARCH	27
ARCHITECTURE OF THE MEMORY SYSTEM	32
EVALUATION OF THE SYSTEM	55
CONCLUSIONS	79
BIBLIOGRAPHY	83
ACKNOWLEDGMENTS	85

INTRODUCTION

An overall trend which is visible in database management today is the following: users are becoming increasingly oriented toward the information content of their data and decreasingly concerned with its representation details. Increasingly, the user interface of a modern database management system deals with abstract information rather than with the various bits, pointers, arrays, lists, etc., which may be used to represent information. Responsibility for choosing an appropriate representation for the information is being assumed by the system and is not exposed to the end user; indeed the representation of a given fact may change over time without the user being aware of the change. The general term for this trend away from representation details is data independence.

Considerable attention is being paid nowadays to n-ary relations as a tool for database management. Codd was the first to give a rigorous definition for n-ary relations in the database context and to emphasize their advantages. Codd introduced concepts which have set the direction for research in relational database management (7).

The relational data model makes it possible to eliminate representation-dependence from the user interface. In the relational model, information is represented in only one way

at the user interface: by data values. User requests become free of any dependence on internal representation, and hence may be framed in a high-level, nonprocedural language. At the same time, the system becomes free to choose any physical structure for storage of data, and to optimize the execution of a given request.

There is much to be said on the limitations imposed by use of conventional Von Neumann processors for nonnumeric applications such as database management. The use of a single processor, even a high speed one, for searching and manipulating data in large databases is simply too slow to meet the response time requirement of many applications. Software techniques such as data structures, file organizational techniques, directories, cross reference pointers, computed addressing, etc., alleviate the speed problem to a certain extent. However, they introduce undesirable side effects such as excess storage requirement, the problem of updating pointers, index files and directories, data inconsistency, and above all greater software overhead. Recent research efforts toward a high-level relational data model further show the limitations of conventional computers in supporting the high-level user's view of data and processing requirements. The reason for these limitations is quite obvious. A data model or a language which is more

convenient for the user typically is more difficult to implement on a machine which was not designed mainly for supporting the high-level data model or the language. Another very serious problem in the existing data base management systems is that data are not stored at the place where they are processed. To "stage" the data into the main memory for processing is very time consuming. It often ties up the important resources of a computing system such as communication lines, channels, and data buses. Ideally, data should be processed at the place where they are stored to avoid spending time in moving data between the main memory and secondary memories. There is definitely a pressing need for research and development work in database machines whose main functions are to efficiently carry out search, retrieval, update, insertion, and deletion.

The purpose of this dissertation is to study an associative memory with content addressing capability for a relational database processor. Given the search operand, the memory system retrieves the relation or the tuples of a relation which satisfy the search operand. The performance of the proposed system is evaluated in comparison to conventional software-based database management computing systems and its economic viability assessed.

REVIEW OF LITERATURE

The three popular approaches to database access can be classified as hierarchical, network, and relational.

The network approach may be characterized as a host-language (usually Cobol) embedded network system; it supports index sequential, hash, direct, and set-location access modes. For example, a record about a city could be accessed through the state in which it is located (via set), alphabetically (index sequential), or through the country in which it is located (via set). The advantage of the network approach is high performance, because all of the access paths can be defined and built (by using pointers) at database creation time. The disadvantage is that it is a relatively low-level language and involves the user in storage management and detailed record access. It is also limited to queries that can be satisfied by the predefined access paths and the approach is therefore inflexible. The storage structures, generally constructed of pointers to linked lists, tend to be quite complex.

In contrast, the relational data model approach is a high-level data retrieval and manipulation language that shields the user from data formats, access methods, and storage management. Access paths do not have to be predefined. The lack of predefined physical access paths means

that relational databases must be exhaustively searched to satisfy a query. Since this searching is very slow on conventional computers, the user is often allowed to optionally specify a prior access path, even in relational systems, to obtain acceptable performance. But, the benefits of a relational user interface can be obtained without the drawbacks of predefined access paths by providing special purpose associative processing hardware to speed-up table searching.

The hierarchical approach is a subset of the network approach in that groups of records can be addressed by only one logical path. A hierarchical database is restricted to a single owner for each member set; however, a network database can have multiple owners for a member. The hierarchical approach has the same advantages and disadvantages as the network approach.

Relational Databases

The considerable attention paid to n-ary relations as a tool for general database management dates from a 1970 paper by Codd (7). Codd was the first to give a rigorous definition for n-ary relations in the database context, and to emphasize their advantages for data independence and symmetry of access when compared to the hierarchical and network models.

Codd's paper introduced concepts which set the direction

for research in relational database management for several years to come. The paper defined a data sublanguage as a set of facilities, suitable for embedding in a host programming language, which permits the retrieval of various subsets of data from a data bank. The paper noted that a standard logical notation, the first order predicate calculus is appropriate as a data sublanguage for n-ary relations. The paper also introduced a set of operators (join, projection, etc.) which were later developed into the well-known relational algebra. Finally, the paper explored the properties of redundancy and consistency of relations, which laid the groundwork for Codd's later theory of normalization.

The term relation may be defined as follows: Given sets $D_1, D_2, \ldots, D_n$ (not necessarily distinct), a relation R is a set of n-tuples each of which has its first element from D_1, second element from D_2, and so on. The sets D_i are called domains. The number n is called the degree of R, and the number of tuples in R is called its cardinality.

It is customary (though not essential) when discussing relations to represent a relation as a table in which each row represents a tuple. In the tabular representation of a relation, the following properties, which derive from the definition of a relation, should be observed:

1) no two rows are identical;

2) the ordering of the rows is not significant; and

3) the ordering of the columns is significant.

When a relation is represented as a table, its degree is the number of columns and its cardinality is the number of rows. The columns of the table are called attributes. The individual entries in each tuple are called its components. A column or set of columns whose values uniquely identify a row of a relation is called a candidate key. When a relation has more than one candidate key, one of them is arbitrarily designated as the primary key.

A first normal form relation is defined as a relation in which each component of each tuple is nondecomposable; i.e., the component is not a list or a relation. Relations in first normal form may be used with any of the relational languages. However, a relation in first normal form may exhibit three kinds of misbehavior, which are called update anomalies, insertion anomalies, and deletion anomalies. All these anomalies arise because more than one concept may be mixed together in the same tuple. These anomalies can be avoided by normalizing the relations.

Normalization theory begins with the observation that certain collections of relations have better properties in an updating environment than do other collections of relations containing the same data. The theory then provides a

rigorous discipline for the design of relations which have favorable update properties. The theory is based on a series of normal forms - first, second, third, and fourth normal form - which provide successive improvements in the update properties of a database. An important objective of normalization is the elimination of update, insertion, and deletion anomalies.

Attribute B of relation R is functionally dependent on attribute A of R if, at every instant of time, each value in A has no more than one value in B associated with it under R. Suppose D and E are two distinct subcollections of the attributes of a relation R and E is functionally dependent on D. If, in addition, E is not functionally dependent on any subset of D then E is said to be fully dependent on D in R. Any attribute of R which participates in at least one candidate key of R is called a prime attribute of R. All other attributes of R are called nonprime attributes.

A relation R is in second normal form if it is in first normal form and every nonprime attribute of R is fully dependent on each candidate key of R. The second normal form is an improvement compared to the first normal form, but sometimes it also exhibits the anomalies associated with the first normal form. Therefore, the second normal form is of little significance except as a stopping-off place on the way

to the third normal form.

The third normal form has been defined in a variety of ways. The original definition was given by Codd (4). The third normal form is also referred to as the Boyce-Codd third normal form. Later writers have proposed alternate definitions which framed the same concept in simpler terminology.

The third normal form is defined as follows: A relation R is in third normal form if it is in first normal form and, for every attribute collection C of R, if any attribute not in C is functionally dependent on C, then all attributes in R are functionally dependent on C. This definition is a formal way of expressing a very simple idea; that each relation should describe a single "concept", if more than one "concept" is found in a relation, the relation should be split into smaller relations.

The design of a database in third normal form depends on the knowledge of the functional dependencies among the attributes of the data. This knowledge cannot be discovered automatically by a system (unless the database is completely static), but must be furnished by a database designer who understands the semantics of the information. In fact, there is no unique third normal form representation for a given database. Codd briefly addressed the problem of

choosing an optimal third normal form from among the various alternatives (4).

Multivalued dependencies, which are a generalization of functional dependencies, lead to the fourth normal form for relational databases. Multivalued dependencies provide a necessary and sufficient condition for a relation to be decomposable into two of its projections without loss of information. The original relation is guaranteed to be the natural join of the two projections and the projections taken together never contain more information than the original relation. The concept of multivalued dependency was first introduced by Fagin who also defined the fourth normal form (10).

Let U denote the set of all attributes on which a relation R is defined and let X, Y, and Z be disjoint sets of attributes of the relation R. Y is said to be multivalue dependent on X for R(XYZ) where (X,Y,Z) is a partition of U, if for every XZ-value xz that appears in R, R(xzY) = R(xY). By definition, a functional dependency is also a multivalued dependency but the converse is not necessarily true. Also, a multivalued dependency that holds for R(XYZ) depends not only on the values of X and Y, but also on the values of Z. So, a multivalued dependency is said to be context-sensitive. On the other hand, a functional dependency is not context-sensitive.

The concept of multivalued dependency leads directly to the fourth normal form. A relation R is in fourth normal form if, whenever a nontrivial multivalued dependency, Y is multivalued dependent on X, holds for R, then so does the functional dependency, A is functionally dependent on X, for every column name A of R.

Intuitively all dependencies are the result of keys. In particular, a fourth normal form relation can have no nontrivial multivalued dependencies that are not functional dependencies. If a relation is in the fourth normal form then it is also in the Boyce-Codd third normal form. A relation which is not in the fourth normal form can be decomposed without loss of information into a family of fourth normal form relations. Decomposing a relation into the fourth normal form does not necessarily decompose it "as far as possible". For example, assume that a relation R(A,B,C,D) has no dependencies other than the functional dependencies that are the result of A being the key. Then R is in the fourth normal form although it is possible to decompose R without loss of information into its projections. Although every binary relation (a relation with exactly two column names) is in the Boyce-Codd third normal form, it is not true that every binary relation is in the fourth normal form.

Multivalued dependencies significantly extend the

understanding of the logical design of relational databases. Multivalued dependencies provide a necessary and sufficient condition for a relation to be decomposable into a family of relations without loss of information. Multivalued dependencies lead to the fourth normal form which is strictly stronger than the Boyce-Codd third normal form. At the present time, the fourth normal form is the optimum representation, in terms of avoiding the update, insertion, and deletion anomalies, for relational databases and any relation in the first normal form can be transformed into a family of fourth normal form relations without loss of information.

Database Processor Architecture

Shortly after the invention of the stored program electronic digital computer in 1946, storage and retrieval of nonnumeric information became an important application. With only a few exceptions, the early file access systems starting in the middle 1950s and the database systems starting in the late 1960s were mapped onto a conventional von Neumann computer. Although the desirable way to access nonnumeric data is by value, the von Neumann architecture precludes this. Therefore, a number of artificial methods are used to convert a value into an address. These artificial methods include sequential, indexed, hashed, and set access

methods. In spite of their indirection, these access methods have successfully met industry needs until the present time. There has, however, been constant research into many aspects of file and database systems in the general areas of improved functionality, improved performance, and improved availability.

At least one early exception to the use of von Neumann architecture to retrieve nonnumeric data existed; this was the Univac File Computer. This system, first delivered in 1954, allowed the addressing of data in mass storage by value rather than by address. This was done by storing the value of the desired key in a search register, and then comparing this value sequentially to values on a drum. With this capability, records of up to 120 characters could be stored anywhere on the drum and retrieved by value; no access method was needed. The importance of this capability is only now being rediscovered.

The quest for improved functionality has led to database systems as we know them today--initially using network structures, and now with growing interest in relational structures. Initially the performance requirement was met by brute-force improvements in hardware speed. The one element that did not change was the architecture. It is only recently that attention has been focused on the need for specialized architecture for database management systems.

Over a dozen relational database systems have been implemented since E. F. Codd introduced the relational model of data in a series of pioneering papers between 1970 and 1971. A number of prototype systems (such as MITs MADAM, GMRs RDMS, IBMs SEQUEL) were implemented primarily to demonstrate the feasibility of supporting high-level, nonprocedural data languages based on the relational algebra or the relational calculus. At about the same time, a number of other prototype systems (such as IBM's RM/XRM, GAMMA-O, and University of Toronto's ZETA/MINIZ) were developed for use as low-level, database access and storage subsystems for implementing high-level, nonprocedural, relational data languages. More recently, efforts have been directed toward implementing more comprehensive systems (such as IBM's SYSTEM R) which incorporates solutions to various specific problems which have been identified. A number of systems which provide natural language interfaces for casual users (University of Toronto's ZETA/TORUS, University of Illinois PLANES, and IBMs RENDEZOUS) have also been implemented. A few cellular associative processors coupled with rotating storage devices (University of Toronto's RAP, University of Florida's CASSM, and University of Utah's RARES) have been developed as alternatives to the conventional von Neumann processors for supporting the relational model of data. Table 1 lists, in approximately chronological order, database

Table 1. Relational database management systems

Name	Year	Machine	Language	Status	Type	Implementors
MADAM	1970	H6000	PL/1	imp/inact	inst	MIT Project MAC
RDMS	1971	H6000	PL/1	imp/act	inst	MIT EE Dept.
IS/1(PRTV)	1971	IBM 360,370	PL/1,MP3	imp/act	inst	IBM UK SC, Peterlee Eng.
RDMS(REGIS)	1972	IBM 360,370	PL/1	imp/act	inst	GM Research, Warren, Mich.
RD/XRM	1972	IBM 370	assembly	imp/act	inst	IBM Camb SC, Cambridge, Mass.
DAMAS	1972	-		des/inact	inst	MIT CE Dept.
GAMMA-O	1973	-		des/inact	inst	IBM SJ
SEQUEL	1974	IBM 370	PL/1	imp/inact	inst	IBM SJ
RISS	1974	PDP 11	Basic-Plus	imp/act	inst	Forest Hospital, Des Plaines, Ill.
GMIS	1975	IBM 370	PL/1	imp/act	inst	MITSSM & IBM Camb SC
ZETA/TORUS	1975	IBM 360,370	PL/1	imp/inact	inst	Univ. Toronto, Canada
OMEGA	1975	PDP 11		des/inact	inst	Univ. Toronto, Canada
PLANES	1975	PDP 10	assembly	imp/act	inst	Univ. Illinois, Urbana
MAGNUM	1975	PDP 10	BLISS	imp/act	comm	Tymshare, Inc. Cupertino, Calif.
INGRESS/CUPID	1975	PDP 11	C	imp/act	inst	Univ. California, Berkeley
RARES	1975	-		des/inact	inst	Univ. Utah
SQUIRAL	1975	-		des/inact	inst	Univ. Utah
GXRAM	1975	IBM 370	PL/1	imp/act	inst	IBM SJ
RAP	1976	-		imp/act	inst	Univ. Toronto, Canada
RENDEZVOUS	1976	IBM 370	APL(PL/1)	imp/act	inst	IBM SJ
QUERY BY EXAMPLE	1976	IBM 370	PL/1	imp/act	comm	IBM YH
LEECH	1976	-		des/act	inst	Glasgow, England
CAFS	1976	-		imp/act	inst	ICL, Stevenage, England
DBC	1976	-		des/act	inst	Ohio State Univ. Columbus
SYSTEM R	1977	IBM 370	PL/1	imp/act	inst	IBM SJ
DB85	1977	INTERDATA-85	assembly	imp/act	inst	Univ. Kansas, Lawrence, Kan.
SDD-1	1977	-		des/act	inst	CCA, Cambridge, Mass.
CASSM	1978	-		imp/act	inst	Univ. Florida, Gainesville
DIRECT	1978	PDP 11	C	des/act	inst	Univ. Wisconsin, Madison

systems which have been designed or implemented to support the relational model of data (11). In the table, the year is when an implemented system became operational or when the design of a system which has not been implemented was first reported. The machine is the computer on which a system has been implemented. RAP, CASSM, RARES, LEECH, CAFS, and DBC are designs for specialized processors; SDD-1 is a distributed system under development; and DAMAS, SQUIRAL, and GAMMA-0 represent proposals for implementing a component of a system. The language is the programming language in which a system has been implemented. The status of a system is designated as either implemented (imp) or only designed (des), and as either active (act), that is, currently under development or in use, or inactive (inact). The type of a system is designated as either institutional (inst), that is, the system is developed as a research vehicle or for internal use, or commercial (comm). As is evident from the table, most of the systems have been software implementations on existing machines and there are very few specialized processors. The rest of this section highlights the most noteworthy features and contributions of some of the systems.

In 1970, MADAM (MacAIMS Data Management System) became operational as the first relational system. MADAM was implemented on MULTICS utilizing the large, addressable virtual memory and flexible access control capabilities of MULTICS.

The most interesting feature of MADAM is the division of the storage space into the relation space and the domain space, and the division of software into a set of procedures which operate on the relation space and a set of procedures which operate on the domain space.

A novel feature of IS/1-PRTV (Information System/1 - Peterlee Relational Test Vehicle) is its microprogrammed implementation of the data compression/decompression procedures. The microprogram implementation is reported to have reduced the CPU overhead to 5 percent. Although it has been estimated that good data compression techniques may achieve 20 to 80 percent savings in storage space, only PRTV and INGRES (Interactive Graphics and Retrieval System) have implemented such techniques. The most important feature of PRTV is its optimizer. The optimizer transforms an ISBL (Information System Base Language) expression into an algebraically equivalent expression which can be more efficiently evaluated. Next it attempts to find an optimal set of access paths for evaluating the transformed expression by considering the estimated costs of various alternative access paths.

The importance of RDMS/REGIS (Relational Data Management System/Relational General Information System) lies in the fact that it is one of the few decision-support systems which have been developed around relational database systems. A

decision-support system is a generalized information system which provides not only the basic database query and manipulation facilities but also appropriate data anslysis and plotting capabilities to assist policy makers in reaching managerial decisions.

In 1974, the SEQUEL system was implemented at the IBM Research Laboratory, San Jose, California. It was primarily intended to determine the feasibility of supporting the SEQUEL data language. Experience with the SEQUEL prototype and many of the ideas developed for GAMMA-O, a hypothetical database access and storage subsystem, provided the foundation for the development of SYSTEM R.

In 1976, the QUERY BY EXAMPLE system became operational at the IBM Thomas J. Watson Laboratory, Yorktown Heights, New York. It supports the very novel data language called QUERY BY EXAMPLE. Development of this system, as in the case of SYSTEM R, significantly benefited from experience with the SEQUEL prototype implementation. The most noteworthy feature of the QUERY BY EXAMPLE system is the QUERY BY EXAMPLE data language it supports. QUERY BY EXAMPLE has been announced as an IBM Installed User Program. Along with MAGNUM, it is one of two commercially available relational systems.

Relational systems which have been implemented on

minicomputers include MAGNUM, PLANES, RISS, DB85, and INGRES. MAGNUM is a commercially available system which was developed in 1975 by Tymshare Incorporated, Cupertino, California. It was intended to be used as a database subsystem for a generalized information system and provides extensive computational and report generation facilities.

The objective of a natural language interface for a database system is to allow casual users to interact with the system without the need to learn artificial data languages such as SEQUEL and QUERY BY EXAMPLE. The three well-known natural language database systems which support the relational model of data are TORUS, PLANES, and RENDEZVOUS.

Specialized Hardware Systems

CASSM, RAP, and RARES are designs for cellular associative processors for performing the query, data manipulation, and data definition activities in the relational context. CASSM (Context Addressed Segment Sequential Memory) has been under development since 1973 at the University of Florida, to support not only the relational model of data, but also hierarchical and network models of data. RAP (Relational Associative Processor) has been developed at the University of Toronto. RARES (Rotating Associative Relational Store) was designed at the University of Utah.

Basically, the design of these specialized processors consists of an array of cellular associative processors which are driven in parallel by a central controller. Each cellular associative processor (commonly called a cell) is composed of a microprocessor (or simple logic) and a segment of a rotating secondary storage device (such as a track of a drum, disk, CCD, or bubble memory). The processing element of each cell performs an operation directly on its associated memory segment.

In CASSM, data are laid out along the track or loop of the rotating storage device in variable-length blocks (8, 18, 20). Each block, which can contain one or more rows of a relation, is treated as a sequence of 40-bit words. Thirty-two bits can be used to store either a delimiter, a column-value pair, a character string, or (to support non-relational applications) pointers and instructions. The remaining bits are used as a tag to identify word content, as mark bits, and for internal processing. CASSM stores a relation as a two-level tree. The first level corresponds to the entire relation and is represented by a delimiter word giving the relation name and the level number. The relation rows are stored following this delimiter.

A row delimiter preceding each row gives the relation name and the level number. One word is then used for each nonnull value in the row. From the word's 32-bit data field,

16 bits are used to encode the column name and 16 bits to encode the value. Since encoding space is needed for column names but none is wasted for null values, storage effectiveness depends on the relations null-value ratio.

CASSM uses auxiliary storage to mark rows of relation and to support its strategy for rewriting a track. When CASSM simultaneously selects more than one row for output, it uses an output arbiter to output one of them and marks the remaining rows for output on subsequent revolutions. CASSM uses a bit-addressable RAM associated with each track for this. To rewrite a track, CASSM uses two physical tracks per logical track. Data is read from the first, analyzed and written to the second, then rewritten to the first with all desired modifications.

Like CASSM, RAP lays its data along the tracks of its storage device, but the similarity ends there; RAP uses a fixed-length representation for the rows of a relation (14, 17). This length can vary from relation to relation, but within a relation all rows must use the same amount of storage. Only one type of relation can be stored on a given track. Within a track rows are stored one per block, and the end of each block is marked by a delimiter.

The beginning of a track has a special track marker, followed by two header blocks. The first block gives the name of the relation stored on the track, and second gives the

column names in the order in which they will appear in the row representations. The blocks following these header blocks contain the rows of the relation. The row blocks contain the concatenated values of the row in the order given by the second header block. These concatenated values are preceded by a string of mark bits. All names and values of the track are encoded as 32, 16, or 8-bit strings, each preceded by a 2-bit code indicating its length.

Like CASSM, RAP uses an output arbiter to select a single row for output when two or more are contending. However, RAP uses the mark bits preceding each row to indicate which rows must be output on subsequent revolutions. To rewrite a row RAP uses two heads per track, a read head connected by a buffer to a write head. The length of this buffer determines the maximum size of a block, since it must hold an entire row. A row is read by the read head, processed in the buffer, and then rewritten with any necessary changes back to the track.

RARES uses a very different organization from CASSM and RAP (13). It lays out relation rows across tracks (along the radius of a disk) in byte-parallel fashion; the first byte of a value is placed on a track, the second byte of the value is placed in the same position on the adjacent track, and so on. The decision to use a byte-parallel rather

than a bit-parallel organization was based on the speed of the logic available to process a row laid out along a radius, given the rotation time of the disk. Each set of tracks used to store a relation in this fashion is called a band. The number of tracks in the band may vary; the size of the band is determined by the width of a row. Relations with wide rows may use more than one radius to store a row. This format is called an orthogonal layout.

The orthogonal layout means that fewer rows can come into contention for output. However, some contention is still possible, so RARES also needs an output arbiter. It uses a fast memory, called a response store, associated with each band to mark rows to be output on subsequent device revolutions. Since RARES was developed only as a query support facility, storage requirements for row rewriting were not specified.

A survey of the access methods used in relational database systems does yield a few interesting and definite trends. First, designers of most of the systems have elected to support the sequential (i.e., nonkeyed) file structure along with one or two types of keyed file structures. In keyed file structures, the storage locations of a group of tuples are determined by the values of the tuples' key. Keyed file structures which have been chosen include hashed, indexed-sequential, and inverted structures, as well as binary

or ternary search trees. Multilist, controlled list-length multilist, and cellular-partitioned structures have never been used.

The use of keyed file structures presents the database administrator with yet another difficult problem, namely, the task of determining which column or combination of columns should be keyed. The problem of determining an optimal set of columns to index for an inverted file structure has received some theoretical as well as empirical treatment. However, selection of columns to be keyed, like the design and evaluation of storage structures and access paths, has depended on the intuition of the database administrator. The costly overhead of monitoring and maintaining a sufficient set of statistics on database usage pattern and internal database characteristics, and the difficulty in analyzing such a set of statistics may be the main reasons for the conventional, intuitive approach to this problem.

Within the context of an inverted file structure, the notion of a clustered index, which has been articulated by the implementors of SYSTEM R, is worthy of discussion. A clustered index is an index through which tuples whose indexed column values are "close" are stored physically "near" to one another--"near" in the sense that they are in the pages which reside on the same track or cylinder of a disk pack. Through a nonclustered index tuples tend to be

scattered at random, regardless of the "closeness" of their indexed column values. The superior performance of a clustered index has been demonstrated by Blasgen and Eswaran (2).

Second, the emphasis placed on increased storage space utilization by the early systems (MADAM/RDMS, RM/XRM, PRTV, and RISS) has been drastically reduced in the more recent systems (INGRES and SYSTEM R). The early systems exhibit a division of storage space into what has been termed relation space and domain space, whereby each distinct data item in any relation (stored in the relation space) is represented by a numerical identifier which references the corresponding value (stored in the domain space). Although integer values of data items are stored as they are, this approach may result in an increased storage space utilization, since multiple-byte character strings are converted to shorter, fixed-length, numerical identifiers. However, this approach potentially results in a deteriorated response time, since in order to retrieve and output qualifying tuples, access must be made not only to the relation space, but also to the corresponding domain space. Because of this drawback, and also because of the rapid decline in memory costs, this division of storage space no longer appears fashionable. In the future, variable-length character strings are expected to be directly stored in the relations.

PRTV and INGRES are the only systems which have implemented some data compression/decompression techniques in order to increase storage space utilization. In view of the high processing time overhead demanded by the data compression/decompression techniques--20 percent of the CPU time in PRTV--it seems unlikely that future systems will implement such techniques.

OBJECTIVE AND OVERVIEW OF THE RESEARCH

Existing computer architecture and hardware does not provide efficient nonnumeric computation for applications such as database management. Most machines are better at numerical computation by orders of magnitude than nonnumerical computation. There is no standardization at the machine level of any class of nonnumeric operations; a simple pattern matching, searching, deleting or retrieving operation when encoded at the machine level can look quite complicated compared to a reasonably complex arithmetical assignment statement. During the last two decades, significant improvements have been attained in the size and speed of primary memory systems, but because of the increase in the sizes of the data sets in practical applications most of the information must still reside in the secondary memory systems. The existing architecture of computers are inadequate to handle nonnumeric computations efficiently because of the need to transfer blocks of information back and forth from the CPU to secondary memory devices. The users of machine independent high-level language processors for nonnumeric operations have to depend, therefore, on expensive and time-consuming software systems.

The approach to overcome this difficulty of repeated block transfers is to employ parallel processing on different

data blocks with associative processing in each block. Therefore, clearly, the memory system has a vital role to play in the efficient performance of the computing system for database management.

This dissertation work involves the analysis and study of a memory system for a relational database processor. The relational model of databases seems to be the best way to support database management at the present time. The memory system for a processor supporting relational databases should be capable of retrieving the desired relation or the tuples of the relation from the database so that the processor could further manipulate them. The memory system is analyzed in terms of relational database processing in a typical university environment; but the same general principles can be applied to any other situation involving relational databases. The important criteria used are explained next.

It is assumed that only one level of indexing is used. An index on the relation name is maintained in the memory. Given the name of the relation the index points to the location of the relation. The principal advantage of using relational databases as opposed to the hierarchical and network approaches is that the method is very suitable for hardware implementation via associative memories. This

advantage is lost if multiple levels of indexing are used. On the other hand, if no indexing at all is used there is the likelihood that performance will deteriorate because a disproportionate amount of time is needed to first locate the relations. Hence, a good compromise is to use one level of indexing.

In the study no restriction is placed on the length of tuples or attributes of a relation; in other words tuples can be of arbitrary length to satisfy the needs of the particular data processing function. This is an important departure from the earlier implementations like CASSM and RAP (8, 14).

In both CASSM and RAP the value codes are obtained by encoding the actual value items (12). It has been pointed out that in practice actual values may have to be used instead of encoded values, particularly in the light of unsatisfactory encoding algorithms. To avoid these complications the proposed memory system does not use encoding algorithms.

Vacant space assignment and garbage collection are challenging problems when tuples and attributes have arbitrary length. An attempt has been made to address them in this study.

If a rotating associative memory which is disk based is

used, the desired relation or the tuples of the relation can be read out within two rotations of the memory, if properly organized.

It is assumed that the relations are stored in the memory system in the fourth normal form (10). It is assumed that it is the responsibility of the appropriate functional unit to transform the relations into the fourth normal form before being stored in the memory system. The fourth normal form was chosen because it is the optimum form at the present time, as far as update, insertion and deletion anomalies associated with unnormalized relations are concerned.

The logic-per-track approach in which content-addressing is implemented by providing search logic for each track of the disk memory is used (12). This logic is given a search operand by the CPU. As the device rotates, the search logic for each track sequentially compares the tuples scanned by the read head with the search operand. All tuples matching the operand are eventually output to the CPU.

The last criterion is that the performance, measured by the time taken to retrieve data from the memory system, should be superior to that required by traditional software implemented relational database management systems.

The principal contribution of this dissertation is the study and analysis of an associative memory, with content-addressing capability, for a relational database processor.

The logic-per-track approach is taken. The tuples and the attributes are allowed to have an arbitrary length, no encoding algorithm is used, and the system utilizes one level of indexing. The performance of this system is analyzed and it is demonstrated that it is superior in comparison to software-based database management computing systems.

ARCHITECTURE OF THE MEMORY SYSTEM

An associative memory design is the result of design choices from a myriad of individual techniques or approaches. These individual choices involve compromises or trade-offs, and the choices are not independent of each other. In any complete system the individual techniques selected tend to support each other. This aspect of design is generally highly intuitive; thus, system design is more of an art than a science. Therefore, when presenting the resulting system design, the reasons for the selected approach are explained.

The basic idea behind the memory system architecture is to devise a large scale, associative storage system by adding content-addressing hardware to rotating storage devices.

The proposed memory system is a hierarchial system employing some of the techniques of classical virtual memory systems.

Figure 1 shows the overall system in terms of block diagrams. It consists of a relational processor, the secondary memory, the track buffer, and the main memory.

The relational processor (which is outside the scope of this dissertation) performs the traditional operations of relational algebra (or calculus) like Project, Restrict, Join, etc., on the data which it obtains from the memory

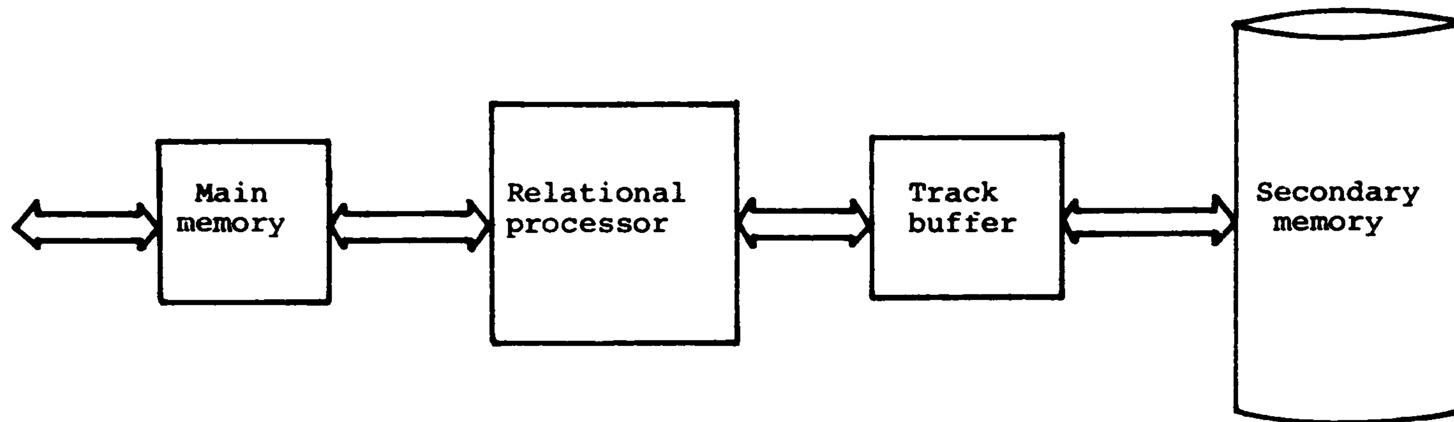

Figure 1. Block diagram of proposed memory system

system. It may also be responsible for transforming un-normalized relations into the fourth normal form before the data are stored in the memory system. The processor can communicate with the main memory and the track buffer.

The secondary memory is assumed to be a disk storage system similar to the IBM 3350 or 3370 system. The relational data in the fourth normal form are stored along the tracks of the disk memory unit.

Each track of the disk memory unit is provided with search logic to implement content-addressing capability. The track buffer acts as a cache for the secondary memory.

The index, which points to the location of the relations in the secondary memory, resides in the main memory. When a query reaches the processor, the processor searches the index in the main memory to find the address of the relation. This address is then used to locate the relation in the secondary memory.

Secondary Memory

In this project the secondary memory has been modelled around a disk storage unit similar to the IBM 3350 or 3370 system, with a movable read/write head. This was chosen because at the present time, direct access disk storage units provide advantages of low cost and large capacity

which are not matched by CCD or bubble memories. The near-term technological prospects favor the fixed-head magnetic disk technology. The technology for read/write heads has progressed to the use of magnetic films. The day is not too far off when a batch fabricated transducer can be used for reading, writing, and sensing track position information.

Bubble technology suffers from a bit rate problem at present, i.e., the data transfer rate is not greatly better than a disk. Consider a 512 K-bit bubble chip, a 0.1 MHz rotating field, and one sensor per chip. One major loop emptying, which is equivalent to a disk revolution time becomes about 8 milli-seconds, about half of the 16-2/3 milli-seconds for disk storage systems at 3600 revolutions per minute. Charge-coupled devices do not have a bit rate problem, but they are volatile storage media; hence they are susceptible to power disturbances or temporary outages, unless defensive measures are taken such as standby batteries. As the power requirements of charge-coupled devices diminish with further technological developments, trickle-charged standby battery power systems may also solve the CCD volatility problem. In view of the current limitations of CCD and bubble memories it was decided that a disk storage unit would be assumed for the secondary memory. At a later date, when bubble memories or other devices become

cost-effective and comparable in performance the same general principles applied in this project can be applied to them.

The data which are in the fourth normal relational form are formatted as per the format in Figure 2. This format is a modified version of the format used by the Symbol 2R computer to store structures (16). No restriction is placed on the length of the attributes or the tuples of a relation, in keeping in line with the criteria for this project.

These data are then laid along the tracks of the disk memory unit in chained blocks of 256 bytes or greater. Each relation is stored along a track. If the storage capacity provided by a track is insufficient, then the relation is to be stored on the tracks of the same cylinder. The reason for using the same cylinder is to eliminate the seek time delay that would have to be incurred if the same relation is stored along tracks of different cylinders.

A relation is essentially a matrix representation of the data, in addition to having other properties that are unique to relations. The format used to represent the data in this project is a linear representation of a matrix, i.e., it is similar to laying the tuples of the relation end-to-end. Hence, all the inherent properties of tuples and relations are still preserved. Additional data items like link and

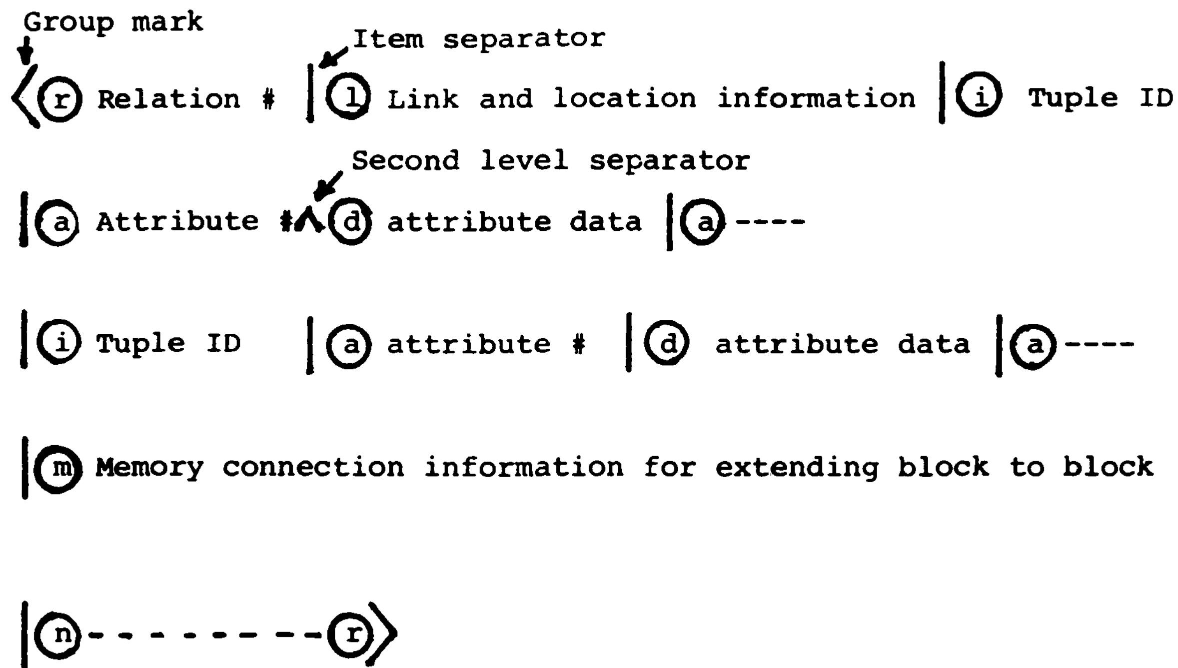

Figure 2. Format of data stored in secondary memory

location information; memory connection information, etc., have been included for ease of processing.

The relation # in the format could be either the name of the relation itself, which by definition of a relation is unique for each relation, or a unique number assigned to each relation via for example a hash code. By the same token, the tuple ID could be either the primary key or a unique number. Obviously, it is more advantageous to eliminate the use of hash codes but the format does provide this option.

In this format, a relation is enclosed between the group marks "< >" and item separators are used to separate the data items. Special characters like ⓡ, ⓛ, etc. are used to indicate the nature of the data that follows the characters. For example, the character ⓡ indicates that the data that follows are the relation #. The character ⓛ indicates that the link and location information follows it, the character ⓘ indicates that the tuple ID follows it, the character ⓐ indicates that the attribute # follows it and the character ⓓ indicates that the attribute data follows it. The character ⓜ is followed by the memory connection information for extending block to block and the ⓝ character indicates that the remaining bytes are blanks and may be ignored.

An extended 8-bit ASCII code is utilized to represent the data. The standard characters of the ASCII code are represented with a leading zero attached to their normal ASCII depiction. The group marks, item separators, and special characters are represented as follows:

Group marks	<	1110	0001
	>	1110	0010
Separators	|	1010	0001
	∧	1010	0010
Special characters	ⓡ	1100	0001
	ⓛ	1100	0010
	ⓘ	1100	0011
	ⓐ	1100	0100
	ⓓ	1100	0101
	ⓜ	1100	0110
	ⓝ	1100	0111

Decoding the left-half byte will indicate the nature of the character; that is, whether it is a group mark, separator, special character, or standard ASCII character. This information may be used to set flags and route the data to the appropriate logical unit. Thus, from an implementation viewpoint, this representation is very attractive.

Blasgen and Eswaran have demonstrated that the speed

of evaluation of a query depends on whether the relation is clustered or unclustered (2). The proposed index, layout of data along the tracks and, if necessary, along the tracks of the same cylinder, and the format ensures that the database is indeed clustered. In this case, the main reason that the relation is clustered is because the data are laid along the tracks of the same cylinder.

To understand the importance of clustering, suppose that a sequence of M tuples corresponding to an interval of key values in index I of relation R is to be accessed. If the relation is clustered, then this sequence can be obtained by accessing only (approximating to a first degree) $(M/|R|)D$ pages, where D is the number of data pages of relation R and $|R|$ is the size (number of tuples) of R. If the relation is unclustered, the M data pages will be accessed. If $|R| = 100$ tuples/relation, $M = 40$ tuples, and $D = 20$ pages; then for a clustered relation 8 pages will be accessed and 40 pages for the unclustered case. The difference in performance is considerable.

Track Buffer

A Track Buffer is the most important unit of the proposed memory system and is based on the logic-per-active track approach. It is composed of comparison logic and local

storage. The logic-per-active track approach takes advantage of the fact that rotational storage devices with an active read head per surface may have all the data of a relation available for inspection per revolution time.

This particular method has been chosen for study because a significant performance enhancement compared to relational database management systems implemented on conventional processors can be expected by using this technique. This argument is based on the following observations. First, all track buffers process one or more given operations in parallel over the entire storage device. Second, the need to access and maintain auxiliary data structures such as indexes and pointers necessary for mapping the logical relations onto their physical counterparts is substantially reduced. Third, the relational interface optimizer, which decomposes the highly data-independent and concise relational query and data manipulation expressions into a sequence of calls to the database access and storage subsystem, is likely to be considerably simplified.

The overall architecture of the track buffer is shown in Figure 3. The most important functions of the track buffer are searching, storing, and retrieving. Several comparator elements form the basis of the associative addressing architecture of the track buffer. The comparators

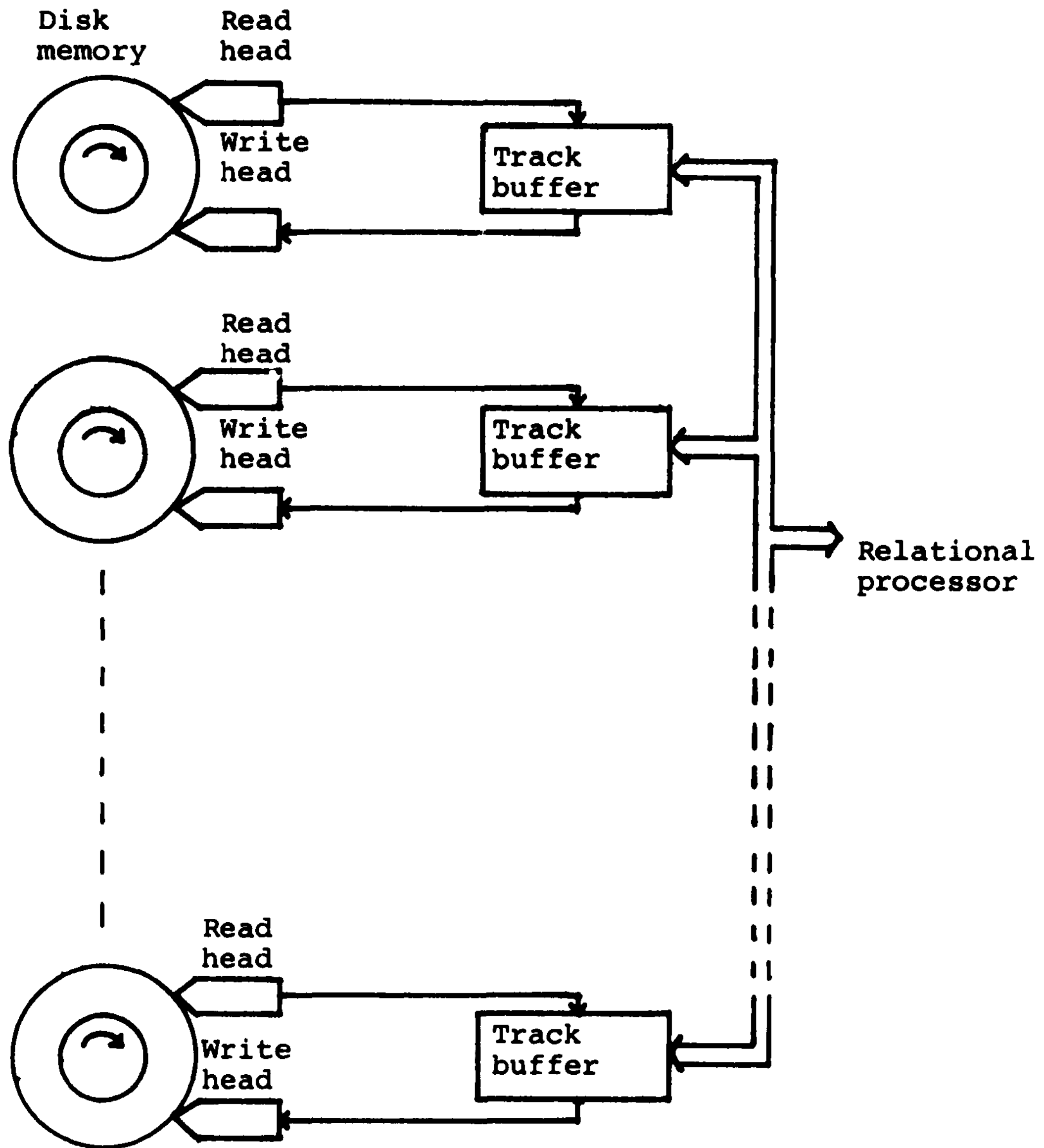

Figure 3. Overall architecture of track buffers

can independently test the contents of one attribute in the database against several literals or several attributes each against different literals. The true or false results of the comparison tests on a tuple can be combined into a disjunctive or conjunctive result to determine if the tuple associatively qualifies for further processing.

An analysis of the requirements for university administrative data processing indicates that the distribution of the number of bytes/attribute is bi-modal as shown in Figure 4. Typically, the length of an attribute ranges from 1 to 40 bytes. For text-editing and other allied purposes, a length of 1000 to 5000 bytes/attribute is sufficient. The distribution of the number of attributes/tuple is shown in Figure 5. These distributions will be used later for the purpose of analyses as representative ones.

The track buffer comparison logic for the proposed system is shown in Figure 6. Register 1 holds a constant value which is the search operand and depends on the query. As data streams off the read head it is stripped of its group marks and separators. The data are shifted or parallel loaded into shift register 2. Registers 1 and 2 are assumed to be 32 to 64 bytes long (as a convenient powers of 2). A minimum of 32 or 40 bits could be used for the anticipated attribute lengths.

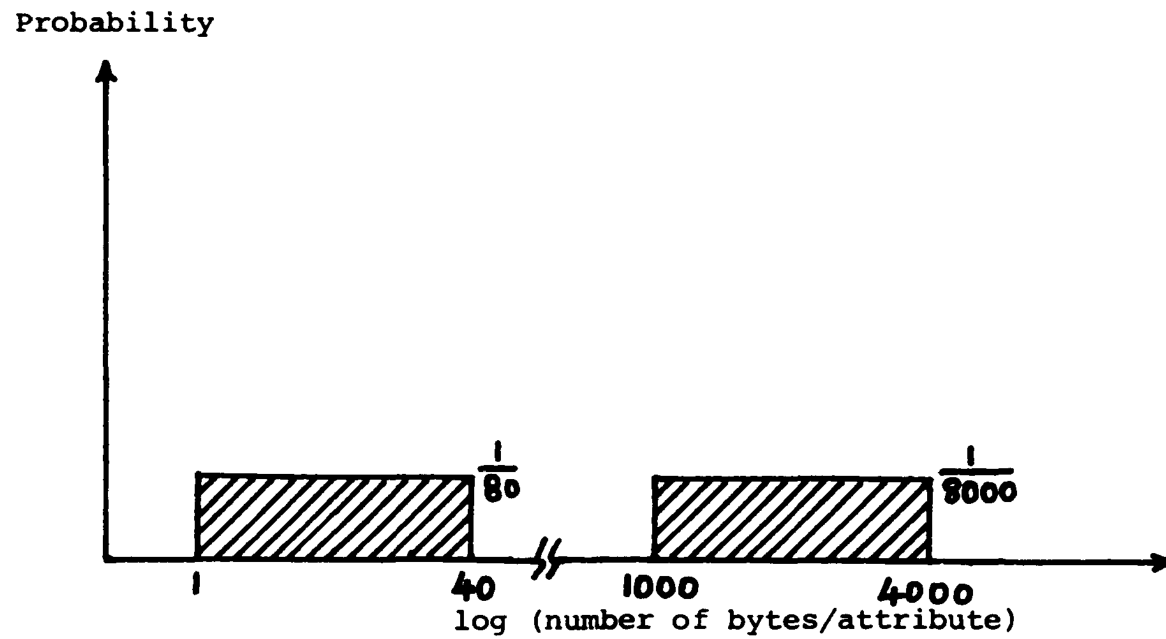

Figure 4. Assumed distribution of number of bytes/attribute

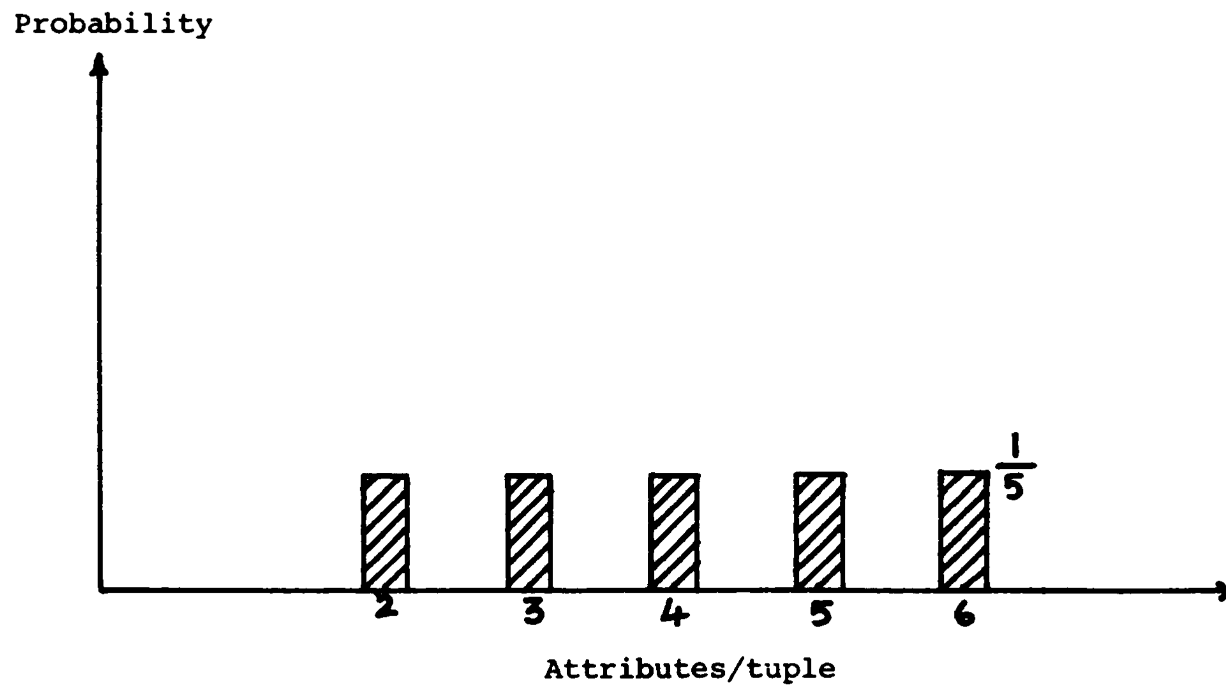

Figure 5. Assumed distribution of number of attributes/tuple

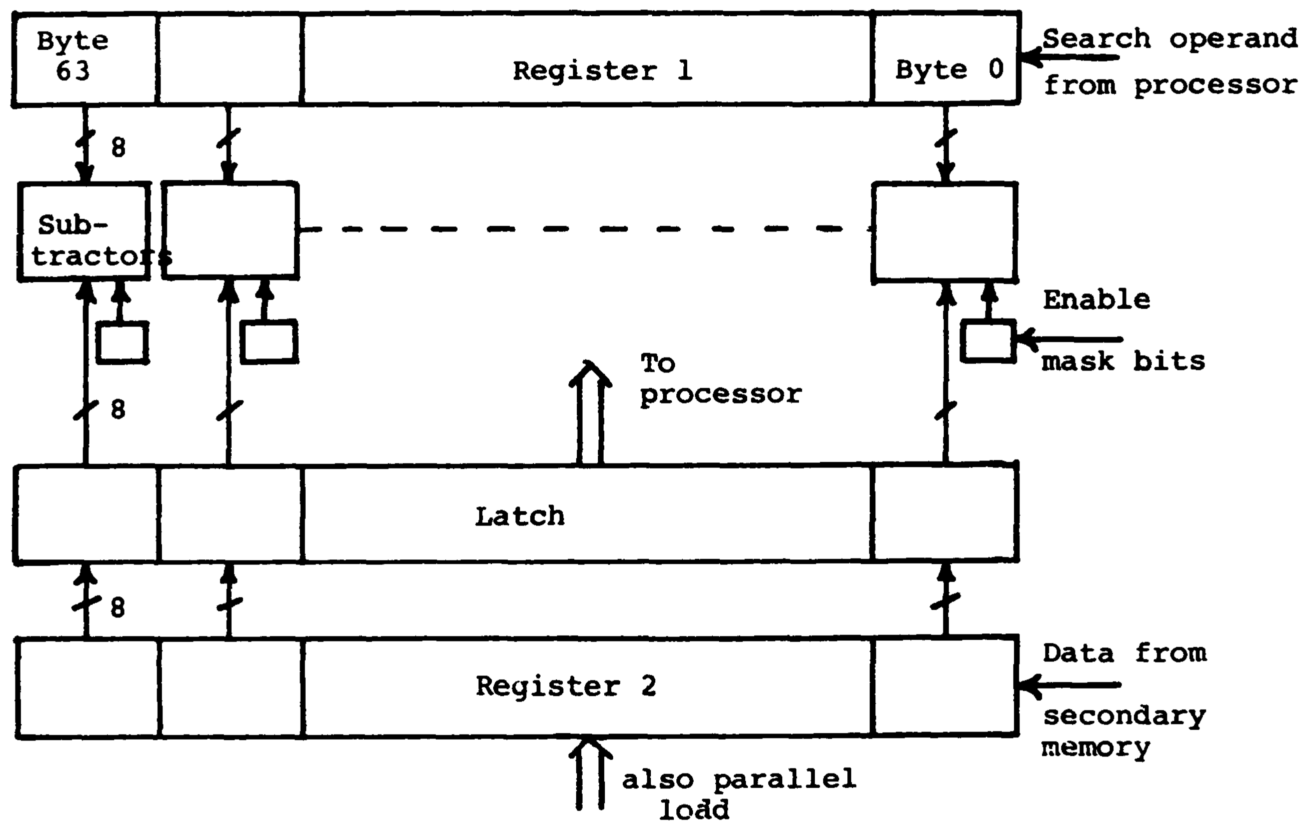

Figure 6. Track buffer comparison logic

The data from shift register 2 are then transferred to a 64 byte long latch. Now, the contents of register 1 and the latch are compared. This comparison is done in parallel over all the 64 bytes. The comparator logic, in the simplest case, consists of 64 subtractors with a cumulative zero flip-flop used to compare the equal condition and a borrow flip-flop to remember the sign of the result.

In this architecture, the comparison of the data in the latch with the contents of register 1 and the loading of data from the storage device into shift register 2 is done in parallel. This parallel operation enhances the rate at which data are searched and retrieved and it ensures that the rate at which comparisons are carried out is faster than the rate at which data is read from the disk.

As noted previously the comparators need to be at least 40 bytes long to accommodate the "normal" attribute. Clearly, a length of 64 bytes, which has been picked, should satisfy a significant majority of queries. The probability of requiring a length greater than 40 bytes is assumed to be low. If a length greater than 64 bytes is required, the comparison is done piece-meal, 64 bytes at a time. In this case, the track buffer logic keeps track of the result of each block of comparison so that the results can be

combined to determine if the data qualifies for further processing. In this case, the time required to select tuples could be greater than a single revolution time.

If the comparison is satisfactory, the data are output to the relational processor for further manipulation. If the data does not fulfill the requirement, the comparison procedure is repeated on the next set of data.

The mask register facility allows one or more of the bytes to be involved in the comparison. For example, certain attributes might be selected from a tuple. By imbedding attribute identification and other tuple structural information in the data stream, the selection can be quickly made in the key byte.

As another more complex example, several alternative representations of the attribute data may be acceptable. For instance, John A. Jones, John Jones, or J. A. Jones might be alternative acceptable spellings of a name. By allowing rapid loading of both the search operand and mask register, alternative spellings could be searched as the data is processed. The tuple number might also qualify the tuple for examination.

Although the 32 to 64 byte comparison logic seems a large amount to allocate per track (800 to 1600 latches or flip-flops, 256 to 512 bits of ALU) this amount can be obtained in a single integrated VLSI circuit. The processing

speed assumed ($\sim$ 200 n.sec. per full comparison) would allow the comparison logic to be shared among tracks. If it were necessary to reduce logic, the buffer logic length could be reduced to 16 bits at the expense of processing time.

There may be an output problem associated with the concurrent processing of data on many tracks. This problem arises because while the data can be read from the storage device in parallel, selected tuples can only be received by the channel sequentially. When tuples are selected simultaneously on several tracks there may not be sufficient time to output these bit-serial tuples in sequence to the channel.

An obvious solution is to use an output arbiter which allows only one of the simultaneously selected (bit-serial) tuples to be output to the channel. The remaining selected tuples are output subsequently by providing additional temporary storage.

A proposed architectural enhancement is to overlap the loading and unloading of the track buffers with comparison processing by pairing track buffer memory segments. The track buffer segment pair configuration is shown in Figure 7. A pair of memory segments are used per track. The pair of segments can be viewed as two memory devices, one being the primary track buffer and the other being the buffer memory. Track connections are reversed as needed. By eliminating the

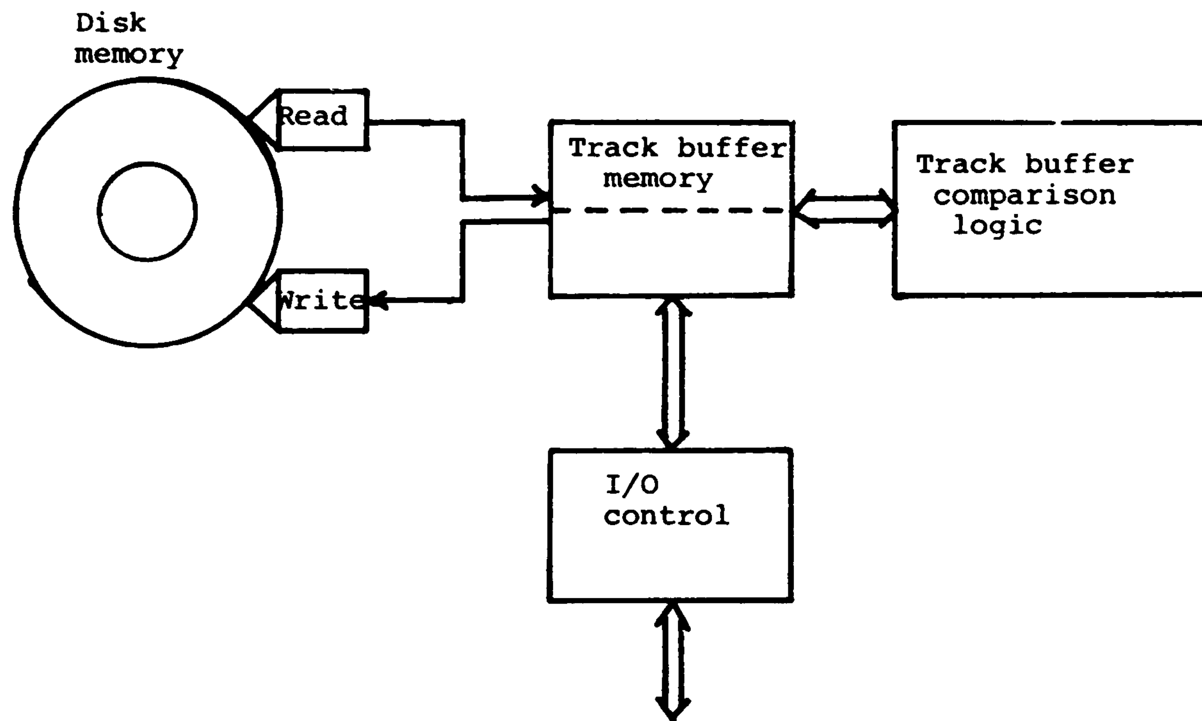

Figure 7. Track buffer segmented local memory

delays caused by loading and unloading the throughput capacity is increased.

The storage associated with the track buffers may in fact be a single high speed memory of sufficient speed to handle the throughput. Storage can be realistically expected to be managed in minimum blocks of 256 bytes. The memory therefore, is assumed to be 1024 bytes with a cycle time less than 100 nanoseconds to handle expected track traffic.

Main Memory

For the proposed system one level of indexing is used with the index residing in the system main memory. Given the name of the relation, the function of the index is to supply the relational processor with the starting address (cylinder and track) of the relation in the secondary memory so that the relational processor can initiate the search for the relation in the secondary memory.

This index is table-driven and its size depends on the size of the secondary memory. Each relation requires two entries in the index, the name of the relation and its starting address. On the other hand, as indicated in Figure 4, attributes are of variable length and typically could be as long as 40 bytes. Also, each tuple can have anywhere from

2 to 6 attributes as shown in Figure 5. Therefore, clearly, the size of the index is much smaller than the size of the secondary memory.

As far as speed is concerned, the time taken to search the index in the main memory should be less than the time taken by the secondary memory and the track buffer to output data to the relational processor to satisfy a query. Even for serial search in a conventional system, this can be conveniently done.

Garbage Collection

When variable length tuples are used, a strategy for garbage collection is needed. A larger tuple clearly cannot be inserted into a smaller tuple position without repositioning other data. One alternative is to not do garbage collection. Whenever the relation is written back to secondary memory, the relational processor deletes the invalid tuples and only the valid tuples are written. In the event of a crisis point (no more available storage), the relations are written back to secondary memory after the relational processor has deleted the invalid tuples. This is a simple and effective strategy for garbage collection when attributes and tuples have variable length and can be easily implemented in hardware. However, because a block

structure is assumed with chaining capability, addition or deletion of a block is simple and straightforward in a cylinder.

Relational Processor

The relational processor has a number of important functions to perform so that the memory system as a whole operates efficiently. The design of the processor itself is outside the scope of this dissertation but it is assumed that the processor can do the following operations.

The relational processor is responsible for normalizing the relations into the fourth normal form and storing this data in the secondary memory in the format shown in Figure 2. When a query reaches the processor, the processor searches the index in the main memory and obtains the starting address of the relation. This starting address is then supplied to the secondary memory so that the search for the relation and the tuples can be initiated. The relational processor provides the track buffer logic with the search operand.

The processor performs update, insertion, and deletion of tuples. The tuples output by the track buffer are used by the processor to perform the standard operations of relational algebra such as join, project, division, and the traditional set operations.

Track Buffer Micro-operations

The proposed track buffer comparison logic is capable of performing the following micro-operations set. By using an appropriate sequence of these micro-operations the content-addressing capability of the track buffer comparison logic can be exploited.

1) shr I D/N - shift-right the contents of register I (I = 1 or 2); either data (D) or null values (N) are shifted into the left-most byte-positions.

2) k shr I D/N - same as previous micro-operation except that the contents are shifted by k byte-positions.

3) shl I D/N - shift-left the contents of register I (I = 1 or 2); either data (D) or null values (N) are shifted into the right-most byte-positions.

4) k shl I D/N - same as previous micro-operation except that the contents are shifted by k byte-positions.

5) REG α (N_1-N_2) ← data/query — data from the secondary memory is loaded into N_1 through N_2 of register 2, if α=2; or load query from the processor into register 1, if α=1. In the default case, i.e., only one byte is to be loaded in, only N_1 is specified.

6) LMR (N_1-N_2) - load mask register so that specified bytes, N_1 through N_2, of register 2 can be masked.

7) Enable/disable mask

8) LAT ← REG 2 (N_1-N_2) - Transfer bytes N_1 through N_2 of register 2 to the latch.

9) BUS ← LAT (N_1-N_2) - bytes N_1 through N_2 of the latch are transferred to the bus. In the default case only N_1 is specified and byte N_1 of the latch is transferred to the bus.

10) COMP (1-3) = REG 1 $\odot$ LAT - bytes of register 1, which are determined by the mask register, are compared with the contents of register 1. The three outputs of the comparator, 1 through 3 (equal-to, greater-than, and less-than), and flags are correspondingly set.

11) LCCAT (α)(β) - load compare count and test. Load the compare count register with a value equal to the number of byte locations where the contents of register 1 matches the contents of the latch. Based on the test condition (α), set the corresponding flag (β).

12) FLAG X_n ← 1,0 - set or clear flag X_n.

13) Branch on FLAGS (a,---) - branch on combination of flags (a,---)

EVALUATION OF THE SYSTEM

In this chapter the proposed memory system is critically evaluated, under typical conditions, in order to justify the usefulness of the system in comparison to existing systems. The performance of the system in terms of processing time is determined and it is demonstrated that the proposed system architecture is superior to present software-based systems. It is also shown that the proposed system appears cost-effective for relational problems.

The efficient organization and management of large storage spaces are the central issues in database management system design. Memory systems hosting databases exist in a three-dimensional space defined by access time, cost, and capacity. The successful design achieves an optimum balance between access time and cost, while the values of these dimensions always maintain a mutually inverse relationship. The designer seeks the fastest possible access time at the lowest possible cost/bit for the entire memory system. The third dimension, total database storage capacity, is primarily determined by start-up costs and end-user application requirements.

In this project the secondary memory, where the relational database is resident, has been modelled around a disk storage unit similar to the IBM 3350 or 3370. This

choice, clearly, meets the general criteria discussed previously.

Two recent developments in hardware technology show promise for providing direct support for the relational database model. These developments are the very-large-scale integration of logic on chips and the development of electronic rotating memory based on the CCD and magnetic-bubble technology. The capability of placing sophisticated and specialized logic can be easily replicated and distributed over data.

The most effective distribution associates logic with small amounts of data. Logic can be distributed over data in any of three ways: 1) by integrating both the logic and the data on a single VLSI chip, 2) by associating the logic with the read/write mechanism of a track or loop of rotating memory, and 3) by configuring a distributed microprocessor based architecture.

The first approach holds little promise, at present, for database applications, since it is too expensive for the large amounts of data that are typically involved. The second and third approaches, however, are worth investigating. In this project, the second technique has been used, rather than the third, because it is cost-effective for a class of problems. If more functions were to be implemented, then the third approach using microprocessors can be utilized.

By associating the selection logic with the read/write mechanism of the memory, the system has the ability to select at the device level. This has two major consequences. First, since each track or group of tracks has its own associated logic, the search and retrieval operations can be performed over all tracks in a cylinder in parallel provided the storage devices are appropriately constructed. Therefore, the system has a high degree of parallelism incorporated into it. Because of this inherent parallelism, the need for indices has been minimized. The system uses only one index which provides the cylinder number for a given relation. Thus, the maintenance of such access structures is also minimized. A second advantage of this approach is that only data meeting the search criteria and selected by the logic are output to the relational processor for further processing. This reduces the data transfer costs and increases the effective utilization of the relational processor. By increasing the utilization of the relational processor the throughput of the entire system is enhanced.

There are two approaches to utilizing content-addressable hardware to achieve very high throughput; that is to complete the content-search retrieval and update the database in the shortest possible time by making use of several content-addressing processors.

The multiple content addressability and single data stream (MCSD) approach merges the data from all the tracks into a single data stream that is content addressed by multiple processors using different conjunctions. The single content addressability and multiple data streams (SCMD) approach processes multiple data streams, one from each track, in parallel. Each processor uses the same conjunction.

In the MCSD approach, one content addressable processor handles the first conjunction while the second processor handles the second conjunction. Typically the i'th processor handles the i'th conjunction. The approach assigns each processor a different conjunction. At any instant, all the processors examine the same data bits against their own conjunction. Thus, at that instant, the entire parameter of the search-retrieve instruction is applied to those data bits. In order to make the same data bits available to every content addressable processor, the system replicates the data bits for each processor. Otherwise, contention over the input bus for the same data bits will be the bottleneck. At the end of a disk revolution, an entire cylinder of records are replicated and content addressed.

In the SCMD approach each content addressable processor uses the same conjunction and each processor examines data

from a different track; thus processor #2 will content address the data from the second track in the cylinder. In the SCMD approach, each content addressable processor examines a different data stream but may be capable of using only a part of the parameter of the search-retrieve instruction, namely a conjunction. To complete the entire parameter a number of passes of the data may be necessary.

The chief advantage of the MCSD approach is that it can process a complete parameter in one disk revolution if all the relations are on a single cylinder. Merging multiple data streams requires a high bandwidth; the number of tracks in a cylinder may thus have to be limited. Moreover, if the search parameter contains only 1 or 2 conjunctions, then a few processors are very busy while most are idle. Finally, since all processors are working on the same parameter, there must be some communication network among them, which adds complexity to the overall system. This approach is very inflexible and severely restricts the possibility of changing the number of disks in a cylinder. This is a serious limitation of the technique, since in a database management system the number of disks in a cylinder may have to be varied to reflect the changing needs for storage capacity.

The major advantage of the SCMD approach comes from associating a content addressing processor with each track.

This makes the design essentially independent of the number of tracks per cylinder and encourages a large cylinder size with a great number of tracks. Since each processor functions independently of the other processors, no interconnection logic between the processors is needed. The approach is very flexible and the number of disks used can be changed if necessary. In addition, the processing of a simple parameter using only 1 or 2 conjunctions is evenly spread among the different processors. The only disadvantage of the SCMD approach is that a parameter of multiple conjunctions may require multiple disk revolutions for processing.

The memory system in this project uses the SCMD approach. Each track of the memory has its own associated track buffer comparison logic which is used for content addressable processing. All track buffers receive the same query from the relational processor and each track buffer examines data from a different track; that is, each content addressable track buffer uses the same search criteria but examines a different data stream. This approach is very flexible and the number of tracks in a cylinder can be changed, to reflect the changing needs for storage capacity, without major changes in the architecture. Besides, no interconnection logic is required between the track buffers.

A large database, such as the one needed for a university administrative data processing, must reside on many cylinders

or even on many disk drives. The management and content addressing of an entire cylinder space pose a problem because a user request may cause the entire cylinder space to be content addressed. In such a case the cylinders must be content addressed in sequence, each cylinder requiring an access-arm movement. To avoid system performance degradation, the address space must be reduced to the few cylinders that will satisfy the search-retrieve instruction.

The information used to reduce the address space may be stored in a directory in the form of indices. In a conventional software-based system, the directory can be as much as one-tenth the size of the database. The architecture in this project uses large, cylinder-sized, content-addressable blocks. An address is only a cylinder number rather than a combination of a cylinder number, track number, sector number, and the name of the relation. As shown in Figure 4, attributes are of variable length and could be as long as 40 bytes. Also, each tuple could have 2 to 6 attributes as indicated in Figure 5. Therefore, clearly, the size of the index is much smaller than the size of the database and can be reduced to as little as one percent of the database. The degradation in performance in the absence of an index and the relatively small size of the index when compared to the database justify the use of an

index to minimize access to the cylinder space. The relational processor can further upgrade the performance. The relational processor can take a Boolean expression of predicates as the parameter of a search-retrieve instruction about a relation and by searching the stored index, generate a list of cylinder numbers to be content addressed.

In any data representation, it is very desirable to close the gap between the physical structure of the data and the information structure of the data as seen by the user. This will avoid multilevel data mapping which reduces system efficiency and data reliability. The relational data model does an excellent job of closing this gap. It is therefore necessary that in a relational database machine the storage representation used be as close to the relational database model as possible so that none of the properties of the relational model are compromised. The format used to represent the data in this project is a linear representation of a relation; it is similar to laying the tuples of the relation end-to-end. Therefore, all the inherent properties of tuples and relations are still preserved. By storing the actual values instead of encoded values the problem of encoding and decoding has been avoided which have been problems in previous relational database systems (12).

The additional data items in the format are for ease of

processing and do not have any effect on the properties of relations. Because of the presence of delimiters either the entire tuple or s specified domains of qualified tuples may be read out by the relational processor. This significantly reduces the data to be transmitted across the channel within a given time interval. Such a reduction will, on the average, leave the channel with spare capacity. This spare capacity is exploited by processing data from many tracks simultaneously. In this way, the transmission rate of selected data is increased.

The system has the capability of storing the data structures very close to the relational data model. The ordering of the tuples is, therefore, not significant. This, in turn reduces the complexity of the logic needed for update, insertion, and deletion which are the three basic operations in a database system. For updating, the updated tuple can be written into the memory. A new tuple can be inserted as the last tuple in the relation since the order of the tuples is immaterial. A tuple can be deleted without effecting the other tuples in the relation. The same can be done for updating, inserting, and deleting relations. When a relation is updated; either some or all of the tuples are updated, or new tuples are inserted in the relation. No change is needed in the index when a relation is updated.

If a relation is inserted into the database, it can be inserted anywhere in the memory where there is sufficient storage capacity for the relation. A relation can be inserted anywhere because the order of the relations is immaterial. An entry corresponding to the relation is needed in the index. Since the order of the entries in the index is insignificant, the entry could be anywhere in the index. When a relation is deleted from the database the corresponding entry in the index is also deleted. Since relations and their entries in the index are unordered, this deletion has no impact on the remaining database. Hence, it is seen that the system is very flexible for update, insertion, and deletion of tuples and relations.

An additional, desirable feature of the format is that no constraint is placed on the length of the attributes, the number of attributes in a relation, and the number of tuples in a relation. Hence, the format used is very versatile.

The performance of the system depends on the following parameters:

1) clustering of the data;
2) parallelism in track buffers;
3) processing a cylinder at a time; and
4) minimum amount of indexing.

Each of these parameters is now examined individually and

the effect on performance enhancement analyzed.

As shown by Blasgen and Eswaran, physical clustering of logically adjacent items is a critical performance parameter in database mangement systems (2). Clustering of logically adjacent data items drastically reduces the cost of accesses, to the storage unit, which is a critical performance parameter in any system. In a clustered database system, tuples which are logically adjacent or close are stored physically near to one another; near in the sense that they reside on the same tracks or cylinders of a disk drive. Unclustered databases tend to be scattered at random regardless of the closeness or logical adjacency of the data items.

A frequent need in database management systems is the ability to access logically adjacent data items. It is therefore crucial to have a clustered database so that the cost of access to the storage is minimized. Clearly, the speed of evaluation of a query depends on whether the database is clustered or unclustered.

In the proposed memory system, the data in a relation are stored along the tracks of the disk. If the storage provided by a track is insufficient for a relation, the relation is stored along the tracks of the same cylinder. The format used for storing the data allows for link and location

information to be included for extending from block to block. Now, if data along all tracks of the same cylinder are processed in parallel simultaneously, the data are accessed at most once and hence the data are essentially clustered. As a consequence of this clustering, the seek time involved in moving the read-head from track to track is reduced to the bare minimum of one seek time. Since large relations are stored along the tracks of the same cylinder, in one seek time the read heads are positioned properly to access the entire relation. The rotational delay is also substantially decreased because the entire data stored along one track can be examined in one revolution. At the same time, since all tracks of a given cylinder are searched in parallel, the entire relation can be content addressed in the time taken for one revolution of the disk memory. As stated previously, in an university administrative database system it is expected that in a vast majority of cases, the storage provided by one cylinder is adequate for storing a relation. Clearly, because of the proposed layout of the data the maximum delay to access an entire relation stored on one cylinder is equal to one seek time plus one rotational delay time.

If a relation is too large for one track and is stored along tracks of different cylinders the performance of the

system will degrade considerably as is now shown. Using an IBM 3350 disk-drive system with movable read/write heads, 25 milli-seconds will be needed for the seek time and there will be a further 8.4 milli-seconds of rotational delay. This 33.4 milli-seconds of delay will be required each time tracks in different cylinders have to be searched. The IBM 3350 has a data transfer rate of 1200 bytes per milli-second and compared to this data transfer rate the above delay is a considerable penalty to be incurred because the database is unclustered. A clustered database will reduce this delay to the minimum possible.

Fundamental to the concept of a database computer is the accessing of data on the basis of value other than the position. In a parallel associative memory, a parallel search is performed over a few thousand bytes in a time of the order of micro-seconds. In a serial associative memory, a serial storage unit like a disk is searched serially to find data meeting the search criteria. Typically it can search megabytes in tens of milli-seconds. Unlike the parallel associative memory, the serial associative memory is not limited to equality searches and hence the serial associative memory is suited for a wider range of applications. The memory system in this project is a serial associative memory.

The system derives its strength primarily because of two capabilities: content addressing and parallelism. When data are resident in the associative memory, the system can access the data by content and can perform search operations in parallel, such as exact match, greater-than, less-than, maximum, minimum, and between limits. And search is fundamental to such operations as retrieval of data, updating data already in the database, sorting, and merging. The system architecture has a high degree of parallelism for performance enhancement, as discussed presently. Parallel processing, also, reduces the software indices required for the system to operate.

By associating the track buffer comparison logic with each track in a cylinder of the disk memory all tracks are searched in parallel. As indicated in Figure 6, shift register 1 contains the search operand which is the query. Data from the memory are shifted in to shift register 2. Data from shift register 2 are then transferred to a 64 byte long latch and compared with the contents of shift register 1. For an exact match, between the data and the query, a match is needed between every byte of the data and the query. Therefore, by using the result of the comparison in every byte the logic can determine an exact match. For greater-than and less-than conditions, the logic can scan the result of the comparison, starting from

the most significant byte and proceeding towards the least significant byte. The highest order byte location where a match does not occur will indicate the relative magnitude of the data with respect to the query. At this location, if the data are less than the query, then the entire data are less than the query. Similarly, if at this location the data are greater than the query then the entire data are greater than the query. Thus, the track buffer comparison logic can check for the three most common conditions of equal-to, greater-than, and less-than. Because of the use of the ASCII representation for alphabetic data and query, the comparison logic can be used for alpha-numeric data and query.

In the track buffer comparison logic, the subtractors are the units which require the largest amount of time to complete their operation. It is therefore advantageous to perform some other operation while the comparison is being done. In the proposed memory system, while the comparison is being done, the next set of data from the memory are shifted into the track buffer memory. If the query for the next set of data is the same as for the previous set of data, the new query need not be loaded in. If the query is different, then the new query is loaded into register 1 during the same time that the data are shifted into shift

register 2. Since the length of the data and the query are the same, by the time the data are shifted in the query can be easily loaded in parallel. Thus, the loading of the data and the query, and the search operations proceed in parallel and this is another feature of the parallelism in the system.

As a result of moving selection logic closer to the data on rotating storage only selected tuples of the relation need be output for further processing. However, if many tuples from different tracks of the cylinder have been selected, two or more of them may contend for the output channel. In such cases, only one tuple can be output and the others must wait for subsequent free time. Using an output arbiter, only one of the simultaneously selected tuples can be output to the channel; the remaining tuples being output later from the track memory may restrict the output rate for selected tuples.

By using a pair of track memory segments as shown in Figure 7, for each track of the cylinder; the loading of data to be searched from the disk memory and the unloading of selected data to the relational processor is overlapped with comparison processing. In this architecture, logically, one track buffer functions as the primary track memory segment while the other is used as the loading memory segment.

The system cannot be delayed by the loading and the unloading of the data as long as the total processing time for the subset of the data being processed is longer than the time to unload selected tuples and to load the next subset of data to be processed. This architectural enhancement further increases the throughput capacity of the system.

An approximate analysis of the system may be done as follows; a more rigorous analysis of the performance effectiveness is done later in the chapter. In the IBM 3350 disk drive system, there are 19 tracks per cylinder and the proposed system would process an entire cylinder in one revolution. But conventional disk systems process one track at a time; therefore, a performance improvement factor of 19 can be expected over conventional disk systems. Because of the minimum amount of software and indexing, an improvement in performance by at least a factor of 1 to 2 can be expected. Since comparison processing takes place in parallel with shifting in of the data from the memory, the performance is enhanced by a factor greater than 1. Further, a pair of track memory segments are used for each track of the memory; this gives rise to additional improvement. Thus, the system is likely to have a hardware processing power which is at least 19 to 152 times that of conventional software-based systems with the same disk system.

Performance Effectiveness

In order to demonstrate the performance effectiveness of the system, the performance of the system is compared to that of an IBM System 370, which is a general-purpose system that can be readily tailored for a variety of applications. If an IBM System 370 is used as a relational database management system then it would be software-based.

The IBM 3350 disk drive system has a data transfer rate of 1.2 million bytes per second. In the analysis which follows, a data trasnfer rate of 2.5 million bytes per second is used, as a worst case value, which will account for improvements in technology in the next few years. At this data transfer rate the memory will transfer 1 byte every 400 nano-seconds and hence will take 25.6 micro-seconds to transfer 64 bytes of data.

In the analysis standard low power Schottky circuits are used for the track buffer logic for analysis purposes. The SN74LS164 which is an 8-bit parallel-out serial shift register can be used for shift registers in the track buffer. It has a maximum guaranteed clock frequency of 25 MHz. If a 20 MHz clock is used, which is less than the maximum guaranteed frequency, the SN74LS164 can shift in 64 bytes of data in 25.6 micro-seconds. Hence, the data can be shifted into the shift registers in the track buffer

at the same rate as they are read from the memory. In this 25.6 micro-seconds, the data should be transferred to the latch, compared and output to the relational processor scanning every possible position of the operand.

In the worst case, the SN74LS164 requires 32 nano-seconds for propagation delay. With a 20 MHz clock, this leaves 18 nano-seconds for set-up on the inputs to the latch. The SN74LS273, which is an octal D-type flip-flop, can be used for the latch. These flip-flops have a maximum guaranteed clock frequency of 30 MHz which is less than the 20 MHz frequency used for the shift registers. Under worst case conditions, the SN74LS273 will need 27 nano-seconds for propagation delay. The SN74LS85, which is a 4-bit magnitude comparator and has outputs for the three conditions of greater-than, equal-to, and less-than, could be used to compare the data with respect to the query. For worst case conditions, the SN74LS85 requires 45 nano-seconds for propagation delay. Thus, even under worst case conditions only (27+45 =) 72 nano-seconds are needed to transfer the data to the latch and perform the comparison processing. This 72 nano-seconds is much less than the 25.6 micro-seconds that the memory takes to transfer the next set of data to the track buffers. That is, the search-retrieve operation can be performed on the data at

the same rate at which the data are read from the memory with modest speed logic.

In order to determine the performance of a software-based conventional system, for comparative analysis, the following assembly language program was coded for execution on the IBM System 370/68. The program searches 64 bytes long subsets of the data and determines whether the data are equal-to, greater-than, or less-than the query. This is similar to the operation of the track buffer comparison logic in the proposed memory system.

The program has 26 instructions which need to be executed each time a search-retrieve operation is performed on the data. The IBM System 370/68 can execute approximately 2 million instructions per second. Therefore, the 26 instructions would require 13 micro-seconds to process the data. Since the time required for processing the data is substantial, there cannot be a continuous transfer of data from the memory to the processor without contention; the processor can keep up with the data transfer rate and the data can be processed at the same rate at which it comes off the memory in this simple case. However, if a 64 byte match were to be made at every byte location, the machine would be a factor of 32 too slow ((13x64)/25.6 ≃ 32). In contrast, the memory system in this project can

```
STMT   SOURCE STATEMENT

   1             PRINT NOGEN
   2 ACHECK      SETUPR
  17             OPEN  (INDATA,(INPUT),RESULT,(OUTPUT))
  25             PUT   RESULT,HEADING
  30 READ        GET   INDATA,BUF
  35             MVC   DATA(64),BUF
  36             GET   INDATA,BUF
  41             MVC   QUERY(32),BUF
  42             SR    3,3
  43             SR    4,4
  44             LA    5,DATA
  45             LA    6,QUERY
  46 COMDAT      CLI   0(5),C' '
  47             BE    COMQUE
  48             LA    3,1(3)
  49             LA    5,1(5)
  50             B     COMDAT
  51 COMQUE      CLI   0(6),C' '
  52             BE    GETLEN
  53             LA    4,1(4)
  54             LA    6,1(6)
  55             B     COMQUE
  56 GETLEN      CR    3,4
  57             BC    B'0100',NOTFOUND
  58             S     4,=F'1'
  59             LA    5,DATA
  60 COMPARE     EX    4,INST
  61             BE    FOUND
  62             LA    5,1(5)
  63             S     3,=F'1'
  64             CR    3,4
  65             BE    NOTFOUND
  66             B     COMPARE
  67 NOTFOUND    MVC   STATUS,NO
  68             PUT   RESULT,DETAIL
  73             B     READ
  74 FOUND       MVC   STATUS,YES
  75             PUT   RESULT,DETAIL
  80             B     READ
  81 EOF         CLOSE (INDATA,,RESULT)
  89             RETURNR
  99 INDATA      DCB   DSORG=PS,MACRF=GM,DDNAME=DATAFL,RECFM=FB,
                       LRECL=80,EODAD=EOF
 153 RESULT      DCB   DSORG=PS,MACRF=PM,DDNAME=RESULT,RECFM=FBA,
                       LRECL=133
```

```
207 INST     CLC   QUERY(0),0(5)
208 YES      DC    CL3'YES'
209 NO       DC    CL3' NO'
210 BUF      DS    CL80
211 HEADING  DS    0CL133
212          DC    C' '
213          DC    CL20' '
214          DC    CL4'DATA'
215          DC    CL60' '
216          DC    CL5'QUERY'
217          DC    CL30' '
218          DC    CL5'MATCH'
219          DC    CL8' '
220 DETAIL   DS    0CL133
221          DC    C' '
222          DC    CL20' '
223 DATA     DS    CL64
224 QUERY    DS    CL32
225 STATUS   DS    CL3
226          DC    CL13' '
227          END   ACHECK
228                =F'1'
```

process the data at a rate which is more than the data transfer rate of the storage system even for the most stringent case. Clearly, for the basic operations that are needed in a relational database management system, the performance of the proposed memory system is far superior when compared to conventional software-based systems.

Cost Effectiveness

The cost effectiveness of the proposed memory system is now demonstrated. The system uses 2 registers which are each 32 to 64 bytes long. These registers would require 64 to 128 SN74LS164 chips if they were constructed of MSI devices. The 32 to 64 bytes long latch needs 32 to 64 SN74LS273 chips and the 32 to 64 comparators require 64 to 128 SN74LS85 chips. A small additional number of SSI chips would be used to combine the outputs of the comparators and for other purposes.

In a real system, a designer would most assuredly turn to LSI or VLSI chips. However, if one assumed an average board cost of $10/IC for the MSI design and an end user multiplier of 6 for end user price, each 32 byte track buffer would cost:

$$\$10 \times (64 + 32 + 32 + 64 + 4) \times 6 \simeq \$12,000.$$

A system of 19 track buffers for a single disk drive system would then have an end user price of $12,000 x 19 = $228,000

or $448,000 for a two disk drive system.

This amount would be about 10% to 20% of the assumed IBM 370 system; this would not be an unreasonable amount for a system primarily dedicated to relational database operation. However, with LSI or VLSI the track buffer cost could be expected to be reduced by 1 or 2 orders of magnitude to very attractive values.

The IBM System 370 CPU provides registers which include the current program-status word, the general registers, the floating-point registers, and the control registers. The 16 general registers are each 4 bytes long. The four floating-point registers are 8 bytes long each. The CPU has provisions for 16 control registers which are each 4 bytes long. In addition, the CPU contains the sequencing and processing facilities for instruction execution, interruption action, timing functions, initial program loading, and other machine-related functions. Most of the hardware cost is in the large main memory however. Clearly, the proposed memory system is cost effective when compared with conventional software-based system if the main job is management of a relational database.

CONCLUSIONS

In order to be considered a true relational system, a database system must posses at least the following attributes (3).

1) All information is represented by data values. No essential information is contained in invisible connections among records.

2) At the user interface, no particular access path is preferred over any other.

3) The user interface is independent of the means by which data are physically stored.

The proposed memory system in this project satisfies all of the above criteria. In addition, the architecture has a number of attractive features which make the system both performance and cost effective. Even though the system was modelled to meet the needs of university administrative data processing system, the results are general in that they could be used for other database management systems.

The format used for the representation of the data is very similar to the normalized relational data model. Thus, the update, insertion, and deletion anomalies of unnormalized relations are avoided. By imbedding structure information in the data representation the processing of the data has been simplified. Because of the presence of

the separators either an entire relation, tuples of a relation, or the attributes of a tuple satisfying the search criteria can be output by the memory system to the processor. The database is clustered and the performance of the system is significantly improved due to this clustering. A minimum amount of indexing is used and the index has one entry for each relation in the database.

The system derives its processing power from parallel processing and content addressing. The logic-per-track approach is used and each track in a cylinder has its own associated logic for content addressing. A cylinder of tracks is processed in parallel and in the track buffer comparison processing of the data with respect to the query proceeds in parallel with the loading of the next set of data. Due to the content addressing capability of the system, only selected data meeting the search criteria are output to the relational processor. This increases the effective utilization of the processor and the overall throughput capacity of the system.

Hardware techniques for data manipulation operations are desirable, feasible, and available. The proposed system can support simple retrieval operations and output data which are greater-than, equal-to, or less-than the query. Due to the ability to mask specified attributes of selected tuples, the projection operation can be done. The proposed system can

easily handle the update, insertion, and deletion of tuples and relations in the database. The lengths of the attributes and the tuples is unconstrained. In addition, the number of tuples in a relation and the number of relations in the database is unrestricted. The number of tracks in a cylinder and the number of disk drives can be changed without any modifications of the architecture. The proposed system is therefore very versatile.

A simple analysis of the performance of the proposed system indicates that for the basic operations that are needed in a relational database management system, the performance of the proposed memory system is far superior to conventional software-based systems. The proposed memory system is also cost effective in comparison to software-based systems if the main job is the management of a relational database.

The principal contribution of this dissertation is the study and analysis of an associative memory system for a relational database management system, with content addressing capability. The proposed system uses a single level of indexing; an index on the name of the relation is required. The suggested format for the representation of the data is a modified version of the format used by the Symbol 2R computer to store structures. The format has structure information

imbedded in it to simplify the processing. The logic-per-track approach is used and the memory system requires one relational processor. An architecture for the search logic has been proposed which enables the system to be performance and cost effective in comparison to traditional software-based relational database management systems. It anticipates the effective use of LSI and VLSI circuits.

Possible Further Investigation

There are several topics for additional investigation related to this work. The traditional set operations of union, intersection, difference, and extended Cartesian product and the special relational operations of join, and division can be hardware implemented. This would provide a complete set of operations for the manipulation of the data. It would be interesting to investigate the feasibility of a distributed microprocessor based configuration where a microprocessor and associated logic is used to process the data from each track in a cylinder. As a logical extension of the single-instruction multiple-data stream associative processor, it is worth examining a multiple-instruction multiple-data stream architecture for supporting an interactive relational database management system.

BIBLIOGRAPHY

1. Babb, E. "Implementing a relational database by means of specialized hardware." ACM Trans. Database Syst. 4, No. 1 (March 1979):1-29.

2. Blasgen, M. W., and Eswaran, K. P. "Storage and access in relational data bases." IBM Syst. J. 16, No. 4 (1977):363-377.

3. Chamberlin, D. D. "Relational data-base management systems." Computing Surveys 8, No. 1 (March 1976): 43-66.

4. Codd, E. F. "Further normalization of the data base relational model." Data Base Systems, Courant Computer Symposia Series, Vol. 6. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

5. Codd, E. F. "Recent investigations in relational data base systems." Information Processing 74. Amsterdam, The Netherlands: North-Holland Publishing Co., 1974.

6. Codd, E. F. "Relational completeness of data base sublanguages." Data Base Systems, Courant Computer Symposia Series, Vol. 6. Englewood Cliffs, N.J.: Prentice-Hall, 1972.

7. Codd, E. F. "A relational model of data for large shared data banks." Commun. ACM 13, No. 6 (June 1970):377-387.

8. Copeland, G. P.; Lipovski, G. J.; and Su, S. Y. W. "The architecture of CASSM: a cellular system for non-numeric processing." Proc. First Annual Symposium on Computer Architecture. Long Beach, Ca.: IEEE Computer Society, 1973.

9. Fadous, R. Y. "Decomposition of a relation into fourth normal forms." Proc. Third COMPSAC Conference. Long Beach, Ca.:IEEE Computer Society, 1979.

10. Fagin, R. "Multivalued dependencies and a new normal form for relational databases." ACM Trans. Database Syst. 2, No. 3 (Sept. 1977):262-278.

11. Kim, W. "Relational database systems." Computing Surveys 11, No. 3 (Sept. 1979):185-211.

12. Langdon, G. G., Jr. "A note on associative processors for data management." ACM Trans. Database Syst. 3, No. 2 (June 1978):148-158.

13. Lin, C. S.; Smith, D. C. P.; and Smith, J. M. "The design of a rotating associative memory for relational data base applications." ACM Trans. Database Syst. 1, No. 1 (March 1976):53-65.

14. Ozkarahan, E. A.; Schuster, S. A.; and Smith, K. C. "RAP: An associative processor for data base management." Proc. AFIPS National Computer Conference, Vol. 44. Montvale, N.J.:AFIPS Press, 1975.

15. Ozkarahan, E. A., and Sevick, K. C. "Analysis of architectural features for enhancing the performance of a database machine." ACM Trans. Database Syst. 2, No. 4 (December 1977):297-316.

16. Richards, H., Jr. SYMBOL 2R Programming language reference manual. Ames, Ia.: Cyclone Computer Laboratory, Iowa State University, 1971.

17. Schuster, S. A.; Nguyen, H. G.; Ozkarahan, E. A.; and Smith, K. C. "RAP.2 - An associative processor for databases and its applications." IEEE Trans. Computers C-28, No. 6 (June 1979):446-458.

18. Schuster, S. A.; Ozkarahan, E. A.; and Smith, K. C. "A virtual memory system for a relational associative processor." Proc. AFIPS National Computer Conference, Vol. 45. Montvale, N.J.: AFIPS Press, 1976.

19. Su, S. Y. W. and Lipovski, G. J. "CASSM: A cellular system for very large data bases." Proc. International Conference on Very Large Data Bases. New York, N.Y.: ACM, 1975.

20. Su, S. Y. W.; Nguyen, L. H.; Emam, A.; and Lipovski, G. J. "The architectural features and implementation techniques of the multicell CASSM." IEEE Trans. Computers C-28, No. 6 (June 1979):430-445.

ACKNOWLEDGMENTS

Dr. Arthur V. Pohm provided invaluable suggestions and criticisms. His patient encouragement and guidance is appreciated.

Thanks to Drs. C. S. Comstock, H. C. Brearley, R. J. Lambert, and T. A. Smay for their counselling at various stages.

The financial support from the Affiliates Program in Electronics and the Department of Electrical Engineering is gratefully acknowledged.