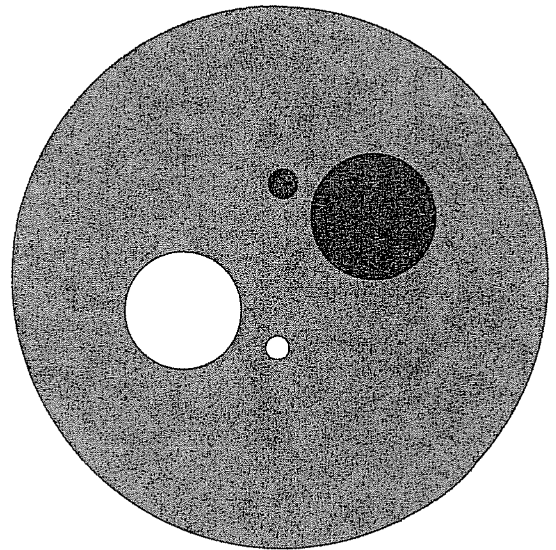
744
73
306

COMPUTER SCIENCES DEPARTMENT
University of Wisconsin-Madison

KURT F. WENDT LIBRARY
COLLEGE OF ENGINEERING
MAR 17 1978
UW-MADISON, WI 53706



DIRECT - A DISTRIBUTED COMPUTER ARCHITECTURE FOR SUPPORTING RELATIONAL DATA BASE MANAGEMENT SYSTEMS

by

David J. DeWitt

Computer Sciences Technical Report #306

October 1977

DIRECT - A Distributed Computer Architecture for Supporting Relational Data Base Management Systems

David J. DeWitt
Computer Sciences Department
University of Wisconsin

ABSTRACT

The design of DIRECT, a distributed computer architecture for supporting relational data base management systems is presented. DIRECT has a MIMD (multiple instruction stream, multiple data stream) architecture. It can simultaneously support both intra-query and inter-query concurrency. The number of processors assigned to a query is dynamically determined by the priority of the query and the size of the relations it references. The size of a relation is not limited to that of the associative memory as in some previous data base machines. Concurrent updates are controlled through address translation tables which are maintained by a controlling processor.

DIRECT is being implemented using LSI-11/Ø3 microprocessors and CCD memories which are searched in an associative manner. A novel cross-point switch is used to connect the LSI-11 processors to the CCD memories. While cross-point switches have proven too expensive for use in general purpose parallel processors, their application in DIRECT demonstrates that these switches can be successfully used in specialized applications.

1.0 INTRODUCTION

Because data bases are increasing in size at a rate which is faster than corresponding increases in processor performance, alternative computer architectures for non-numeric applications must be investigated. One of the first alternative architectures was Bell Labs' XDMS implementation of the CODASYL DBTG network data model [1]. By isolating the function of the data base management system on a separate microprogrammed processor with an instruction set tuned to perform data base management system primitives efficiently, significant performance improvements were achieved. While XDMS demonstrated the feasibility and desirability of the back-end design, its potential for future performance improvements is very limited since it is basically a SISD (single instruction stream, single data stream) architecture.

Since the nature of data base processing lends itself to parallel processing of user queries, several new architectures have been recently proposed which are capable of parallel and/or associative searches of the data base. Each of these efforts is based on the idea of a logic per track device which was first proposed by Slotnic[2] as an alternative to the high cost of fully associative memories. These pseudo-associative devices have been examined as attached processors for associative file management by Parker[3], Healy, Doty, and Lipovski[4], Parhami[5], Minsky[6], and Lin, Smith, and Smith[7] (RARES).

Currently being implementated are two back-end data base processors which exploit the logic per track idea. They both

differ from the research efforts mentioned above in that they deal with all aspects of a DBMS. CASSM, which was first proposed by Su, Copeland, and Lipovski[8,9,10] in 1973, is a cellular processor which is capable of directly supporting all three data models (relational, network, and hierarchical). RAP, an associative processor for data base management, which efficiently supports the relational data model, has been described by Ozkarahan, Schuster, and Smith in [11,12,13].

In RAP, a host system communicates with the user, compiles the user data base query into RAP primitives, and transmits these primitives to RAP. The RAP processor consists of a set of cells each of which is a single disc track on a fixed head disk and a microprocessor for searching the track. RAP also contains a controller to provide interpretation and decoding of RAP primitives and to coordinate the parallel execution of the cells. Also included is a set processor to combine cell results to compute summary statistics. The set of RAP primitives include the following:

1. Selection primitives to locate and mark subsets of record occurrences
2. Retrieval primitives
3. Update primitives
4. Aggregate functions such as MAX, MIN, COUNT
5. Data model definition primitives
6. Insertion and deletion primitives
7. Control primitives

While analytical modeling of a conventional system and the RAP system clearly demonstrated the superiority of RAP (except for the important relational algebra join operator which performed only marginally better), the performance of large implementations of RAP and CASSM may be restricted because they are SIMD (single instruction stream, multiple data stream) architectures. Consider, for example, a RAP processor with 100 cells, when it is executing a query on a relation which occupies only 10 cells. Since RAP is a SIMD processor only 10% of the processing potential will be used. The other 90 cells, whose cellular memories do not contain the relation being referenced in the query, will effectively be idle throughout the duration of the current instruction.

As the logical extension of the SIMD associative processor, we have designed and are implementing a MIMD (multiple instruction stream, multiple data stream) architecture for supporting a relational data base management system. Our original goal was to design a parallel processor system which does not by design waste a large percentage of its processing potential by permitting processors to be idle, particulary in a multi-user environment. Further reflection on the nature of interactive database management systems provided an equally important motivation for a MIMD back-end architecture instead of a SIMD one. Since an important objective of an online DBMS is to permit a large number of users to interact with a data base simultaneously, any new architecture proposed should enhance the performance of a many user system.

Thus, like a timesharing system, an important objective of a DBMS is to permit a large number of users to share a common resource. A MIMD architecture is a much more appropriate approach for achieving this goal since it permits users to simultaneously share the processing power of the back-end.

In this paper we describe DIRECT, a distributed relational computer taxonomy. Among the features of DIRECT are:

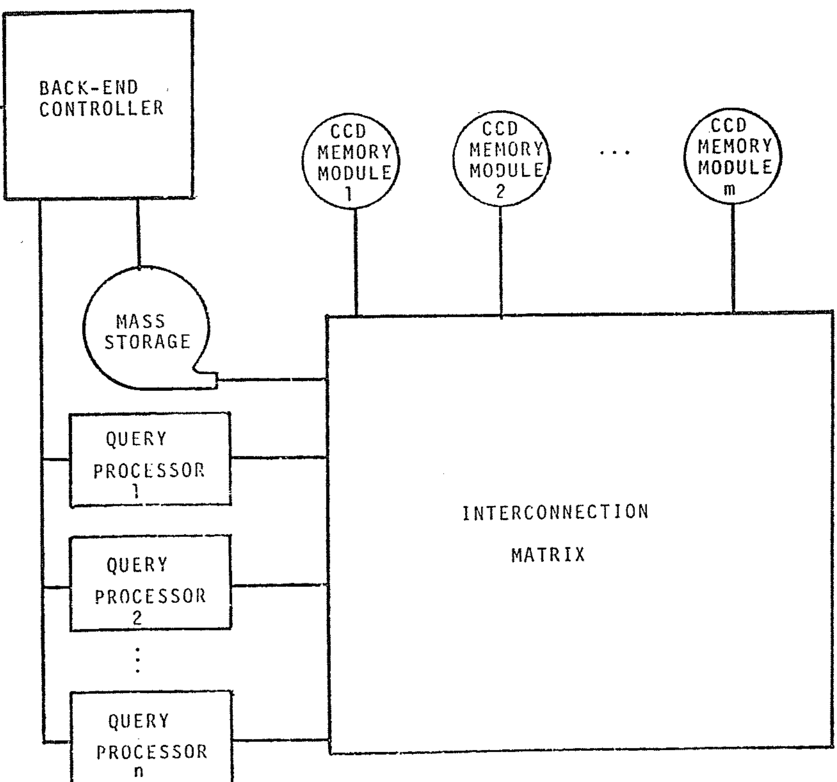
1. Simultaneous execution of relational queries from different users in addition to parallel processing of a single query.
2. Dynamic determination of the number of processors assigned to a query based on the priority of the query and the size of the relations it references.
3. Relation size is not limited by the size of the associative memory.
4. Control of concurrent updates through address translation tables.
5. Compatibility with INGRES [14,15], an existing relational data base system.

2.0 DIRECT SYSTEMS ARCHITECTURE

2.1 Introduction

When operational, the complete DIRECT system will consist of five main components: a host processor, the back-end controller, a set of query processors, a set of CCD memory modules which are used as pseudo-associative memories, and an interconnection matrix between the set of query processors and the set of CCD memory modules. A diagram of these components and their interconnections can be found in Figure 2.1.

The host processor, a PDP 11/45 running the UNIX operating system, will handle all communications with the users. A user who wishes to use the data base system will log onto a modified version of INGRES, a relational data base system[14,15], and proceed in the normal manner. However, when the user wishes to execute a query, a modified INGRES, after parsing and decomposition, will send the query to the DIRECT back-end controller for execution. The INGRES decomposition strategy [16] transforms a user query into a set of one and two variable queries which we will call a "query packet". In addition to simplifying the implementation of DIRECT, INGRES will be useful for evaluating the performance of DIRECT. By running benchmark scripts on both standard INGRES and INGRES with query execution on DIRECT, we will be able to determine the cost effectiveness of the DIRECT architecture compared to a relational DBMS on a conventional processor. Furthermore, by instructing the controller to assign



DIRECT SYSTEM ARCHITECTURE
Figure 2.1

each query packet to every query processor, we can transform DIRECT into a SIMD machine. This will permit us to compare DIRECT's performance to that of a RAP-like architecture.

The back-end controller is a microprogrammable PDP 11/40. It is responsible for interacting with the host processor and controlling the query processors. After the back-end controller receives a query packet from the host, it will determine the number of query processors which should be assigned to execute the packet. If the relations which are referenced by the query packet are not currently in the associative memory, the back-end controller will page them in before distributing the query packet to each query processor selected for its execution. A detailed discussion of the operations performed by the back-end controller is found in Section 4.0.

Each query processor is a PDP 11/03 with 28K words memory. The function of each query processor is to execute query packets assigned by the back-end controller and transmitted from the controller over a DMA interface to the query processor. The instruction set of a query processor is described in Section 5.0.

Since DIRECT has a MIMD architecture, it is capable of supporting both intra and inter-query concurrency. To facilitate the support of intra-query concurrency, relations are divided into fixed size pages. Each query processor, assigned by the controller to execute a query packet, will associatively search a subset of each relation referenced in the packet. When a query processor finishes examining one page of a relation, it will make

a request to the back-end controller for the address of the next page it should examine. Since several query processors, each executing the same query, can request the "next page" of the same relation simultaneously, the controller operations must be indivisible. This will insure that each of the query processors will be given a different page to examine. After receiving the address of the page from the controller, the query processor must be able to rapidly switch to that page. The interconnection matrix, as described in Section 2.3, will permit this.

To facilitate support of inter-query concurrency, the associative memory and interconnection matrix must permit two query processors, each executing different queries, to search the same page of a common relation simultaneously. By eliminating duplicate copies of a relation, we not only reduce memory requirements but, more importantly, the problem of updating multiple copies of a relation is eliminated without sacrificing performance.

2.2 A Shared Associative Memory

After consideration of the requirements of both intra and inter-query concurrency, we divided each relation and the associative memory into fixed size pages of 16K bytes.

Each page frame of the associative memory contains 16K bytes and is constructed from eight charge coupled device chips (CCDs). Bytes are stored across chips which are kept synchronized with a common clock. Furthermore, one address register is used to indicate the address of the current byte which is available from all

CCD page frames.

This page size, which may seem small (compared to that of RAP) to the reader, was chosen for several reasons. One important reason is financial. By choosing a small page size we can construct more page frames for a fixed amount of money. Our initial configuration will have thirty-two page frames. Also, more page frames will have a higher potential for concurrency and the smaller page size will enable us (in an academic environment with our small data bases) to mimic a large data base with small relations. If the page size was too large then each relation might fit on just one page. This would limit the potential concurrency to just inter-query concurrency instead of a mix of intra and inter-query concurrency.

Another important reason for choosing a small page size is to minimize the amount of internal fragmentation which occurs when a relation does not fill all of the pages it occupies. While a small page size does minimize this wasted space, it does so at the expense of a larger page table size in the back-end controller (external fragmentation).

2.3 The Interconnection Matrix

To support inter and intra-query concurrency, the interconnection matrix must permit:

- a query processor to rapidly switch between page frames containing pages of the same or different relations.
- two or more query processors to simultaneously search the same page of a relation.
- all query processors to simultaneously access some page frame.

The bandwidth of the interconnection scheme which is selected must also be very high. Consider for example a DIRECT configuration of 100 query processors and 100 page frames. If each page frame produces one byte every 750 ns and bytes are consumed by the query processors at the same rate, then the data rate between memory and query processors would be 10**9 bits/second.

Some possible interconnection schemes are the time-shared bus, pipelined loop, multiport memory, and cross-point switch. The high bandwidth requirement eliminates the time-shared bus and pipelined loop as potential contenders. A multiport memory with a RAM buffer for each query processor might be suitable for a small implementation of DIRECT since a suitably designed CCD memory can have a bandwidth of 6 megabytes/second while the bandwidth of the LSI-11 Q bus is 1.5 megabytes/second.

For large DIRECT configurations, the cross-point switch seems to be the only feasible interconnection scheme. Traditionally, the use of cross-point switches has been limited because of their high cost and complexity due to the following requirements:

1. High bandwidth between processors and memories for addresses and data.
2. Contention detection and resolution hardware to handle simultaneous access of two or more processors to the same memory bank.

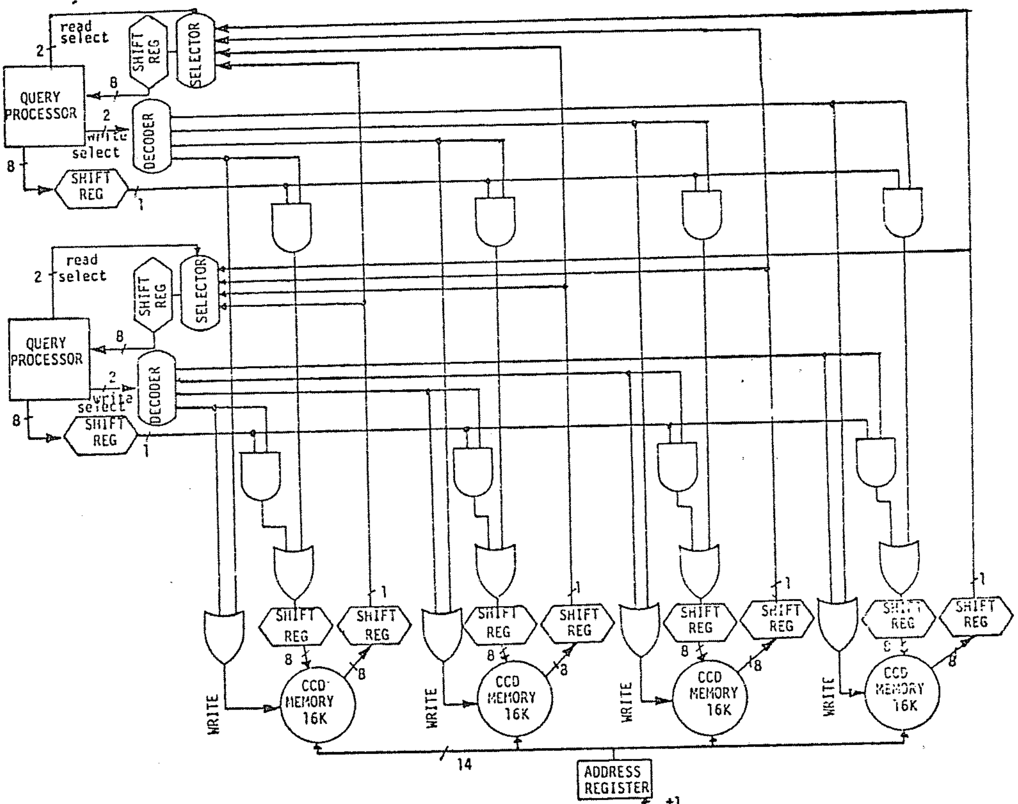
3. Extremely fast switches to minimize the delay time introduced by the switch in each memory access. In C.MMP [18] the switch introduced an additional delay of 250 ns to the memory cycle time of 250 ns.

We have designed a cross-point switch which can be used as the interconnection matrix in DIRECT and yet be constructed without incurring the high cost of the traditional cross-point switch. A diagram of this switch for a DIRECT configuration with two query processors and four page frames can be found in Figure 2.2. The notable features of this switch are:

1. NO address lines.
2. 1 bit wide data paths.
3. Elimination of conflict resolution hardware.
4. Minimization of the effect of the switch delay on memory performance.

While this cross-point switch is not suitable for a general purpose computer it is well suited for an associative processor such as DIRECT.

Address lines were eliminated by stepping the CCD page frames with a common clock and using one address register for all page frames. The DMA interface in each query processor need only compare its starting address with that of the address register to determine when to start the transfer to or from the query processor. Furthermore, since the query processor will associatively search the entire page of the relation, it does not have to start reading at the beginning of the page. Thus, a latency time of



A 2x4 DIRECT Configuration
Figure 2.2

essentially zero can be achieved. The performance of the switches is also not very important when one considers that each query processor will generally examine an entire page before switching to another page frame. Since the time to examine a complete page is .012 seconds, the effect of a switching time on the order of one microsecond is insignificant.

The data paths through the cross-point switch have been reduced to one bit wide paths by using a pair of serial-in/parallel-out and parallel-in/serial-out shift registers at each query processor and page frame interface. (See Figure 2.2). Because shift registers (such as the AM25LS164 and 299) can be shifted at the rate of 1 bit/20 ns, the memory rate of 750 ns/byte can be maintained by using (for reading) a 1 bit data path with parallel to serial conversion at the memory interface and serial to parallel conversion at the query processor interface.

Finally, conflict resolution hardware was eliminated by not permitting query processors to address individual bytes on a page. Instead, each page frame produces bytes without a request from any query processor (except for the request which caused the page to be loaded into a frame). A query processor which wishes to examine a page of a relation simply switches to the data line of the proper page frame. This approach permits several query processors, executing the same or different queries, to simultaneously examine the same page of a relation.

Conflicts which arise when two or more query processors at-

tempt to write onto the same page simultaneously are handled in software by the back-end controller. A discussion of the technique used can be found in Section 3.0.

2.4 Conclusions

In conclusion, the architecture of DIRECT appears to solve the problems associated with the need to support inter- and intra-query concurrency. The original configuration, whose implementation is supported by NSF equipment grant #MCS77-08968, will have five LSI-11/03 query processors and thirty-two 16K byte CCD page frames.

3.0 SOFTWARE CONSIDERATIONS

3.1 Relational Data Model

The data model chosen for DIRECT is the relational model which was introduced by Codd [17]. In a relational database both entities and relationships between different entities are described in terms of normalized relations. For example, a database containing information about suppliers and parts might contain one relation describing all the suppliers (e.g. name, address,status), one describing all the parts (e.g. part#, part name, weight, color), and a third relation describing which suppliers supply which parts. One can view a normalized relation as a table. Each row in the table is called a tuple and describes an entity (e.g. one part, one association).

In addition to the high degree of data independence afforded by the relational data model, its regularity makes it very suitable for hardware implementation using associative memories and parallel processors. In INGRES, for example, relations are used to describe the data model, integrity constraints, and views, in addition to entities and associations.

3.2 Page Format

Each relation in the data base is divided into a number of fixed size pages. Each page contains tuples in sequential order from only one relation. When each relation is created, the maximum length of each attribute is specified by the user. Each tu-

ple in a relation is allocated a fixed number of bytes (ie. the sum of the maximum length of each attribute in the relation). By choosing a fixed length format, we can eliminate the need for special characters to delimit tuples within the page and attributes within the tuple.

While the fixed format will indeed waste space it reduces the need for garbage collection and simplifies modification and insertion of tuples. The need for garbage collection is reduced by marking a deleted tuple in an appropriate fashion. Later, when a tuple is to be inserted the query processor can search the relation until it finds a vacant tuple location. Alternatively, it can place the tuple on the last page of the relation if it does not find a spot after examining a predetermined number of pages. Modification of a tuple will never require that the tuple be moved since it was allocated the maximum number of bytes allowed in the first place.

The need for garbage collection has not been completely eliminated however. If, over time, many tuples are deleted from a relation it may be desirable to compactify the relation. The COMPRESS operation of the query processor performs this function.

Finally, we assume the existence of two special characters which are used to mark the beginning of the page (BOP) and the end of the tuples on the page (EOP).

3.3 Mark Bits and Temporary Relations

Unlike RAP, DIRECT does not use mark bits to indicate which tuples in a relation have satisfied the search criterion of a query. Rather, as the query is executed, tuples which satisfy the search criterion are written into a temporary relation on a page frame in the associative memory. This approach was chosen for several reasons. First, mark bits reduce the potential performance of the processor by forcing a query processor to lock each relation it is evaluating. This is not a problem for RAP which is a SIMD processor. It would, however, introduce problems in DIRECT where two or more query processors, each executing a different query, can potentially be accessing the same page of the same relation simultaneously. An alternative approach would be to provide duplicate sets of mark bits. However, unless one set was provided for each query processor, then conflicts could arise over sharing the mark bits.

Furthermore, the result of any relational query is a temporary relation which the user might wish to add to the data base. If this is the case, the query processor simply passes a description of the temporary relation to the back-end controller which will add the necessary information to the address translation tables it maintains.

Output performance is also enhanced by this approach. The result of a query which is not to be saved as a new relation will reside in a temporary relation. When the channel between the host and the back-end controller is free, the query processor

will transfer tuples from the temporary relation, via the controller, to a waiting process in the host which was spawned when the query was sent from the host to the back-end. The temporary relation acts as a buffer and should permit maximum utilization of the data paths in the process. If mark bits were used, the relation(s) which were used in the query could not be freed until all the qualifying tuples were transferred to the host. This clearly is not the case in our approach.

In addition, the performance of the "temporary relation approach" is essentially the same as with mark bits. Output of qualifying tuples to the temporary relation can be overlapped with the input of tuples to be examined. While more space is used during query evaluation than in the RAP approach, the increase in potential performance of the entire system is significant and appears to justify the increased page traffic which will result. However, since relations will be referenced in a predictable fashion, page faults should be avoidable by doing anticipatory paging. (See the NEXTPAGE operation in Section 4.2).

3.4 Description of Tables Maintained by the Back-End Controller

The back-end controller maintains several sets of tables which are used for describing data bases, establishing concurrency control for updates, and coordinating page requests from query processors executing the same query packet.

The Relation Address Translation Table (RATT), as shown in Figure 3.1, contains one entry for every relation of every data-

base. The relation lock can have one of three values: UNLOCKED, IN-USE, LOCKED. A relation is UNLOCKED if no query processor is currently accessing it. If a query processor is searching a relation, the lock has a value of IN-USE. If a query is to update or modify a relation, it must first LOCK the relation after all queries currently using it have terminated. The owner entry in the RATT table indicates the name of the owner of the relation. This is used to prevent an unauthorized person from deleting a relation.

RELATION ADDRESS TRANSLATION TABLES
[One Entry Per Relation]

RELATION NAME	DBNAME	PAGE TABLE POINTER	RELATION LOCK	OWNER	TUPLE WIDTH	NUMBER OF ATTRIBUTES

Figure 3.1

Each entry in the RATT table contains a pointer to a Page Table (PT) for the relation. The format of PT is shown in Figure 3.2 along with a description of each entry in the table.

The last table which is used by the controller for maintaining information on the data base is the Attribute Catalogue Table (ACT) as shown in Figure 3.3. There is one table for each data

base and within each table an entry describing the format of each attribute of every relation in the data base. When a query packet is assigned to a query processor this formatting information is passed along with the query packet.

PAGE TABLE FOR RELATION J
[One Per Relation]

PAGE	PRESENCE BIT	DIRTY BIT	LOCK BIT	FRAME NUMBER	DISK ADDRESS

Presence bit:
- Ø if page i is on disk
- 1 if page is in some CCD page frame

Dirty bit:
- Ø is page i is clean
- 1 if page i has been updated and thus must be paged out

Lock bit:
- Ø is page i is unlocked
- 1 if page i is locked

Frame number:
If presence bit=1 frame contains the CCD frame number in which page i of the relation can resides

Disk address:
If presence bit=Ø disk address contains the disk address where page i of relation j can be found

Figure 3.2

The Query Packet Task Table (QPTT), as shown in Figure 3.4, has one entry for each relation referenced by each query packet. The currency pointer entry points to the current page of the relation (initially page 0) for that query packet. When a query processor executing query packet i page faults and requests the next page of relation j to continue processing the query packet, the currency indicator will indicate which is the proper next page. Operations on the QPTT must be indivisible as several query processors, executing the same query packet, may simultaneously request the "next" page of the same relation.

ATTRIBUTE CATALOGUE
(One Per Relation)

RELNAME	ATTRIBUTE NAME	OFFSET IN BYTES	TYPE	LENGTH

Figure 3.3

QUERY PACKET TASK TABLE

QPKT #	RELNAME	CURRENCY POINTER	POINTER TO PAGE TABLE FOR RELATION

Figure 3.4

4.0 BACK-END CONTROLLER PRIMITIVES

4.1 Introduction

The back-end controller is responsible for receiving queries from the host, returning results from a query to the host, and doing memory and processor management. The first two tasks are self-explanatory and will not be discussed further. Memory and query processor management includes:

1. Determining how many query processors are to be assigned to execute a query. The number chosen will depend on:
   - The size of each relation referenced by the query.
   - The priority of the query. Thus, low priority queries can be assigned to just one query processor.
   - The number of query processors currently available. If a high priority query packet which references large relations cannot be assigned to enough query processors initially, then as other query packets terminate execution, the query processors which are freed can be assigned to execute the high priority query which may already be partially executed. This is possible by using the Query Packet Task Table to synchronize query processors through their requests for pages.
2. Assigning CCD page frames to query processors. A query processor which requests the next page of a relation will busy-wait until the controller sends it the number of the page frame which contains the next page it is to work on.

3. Scheduling page transfers between CCD page frames and mass storage. If the page requested by a query processor is not resident in some page frame, then the controller must, if necessary, free a page frame, and schedule a read operation of the desired page.

4. Handle updates properly. When a query requests a page frame of a relation, it must specify what it intends to do - retrieve or modify. If it is a retrieve request then access will be granted if the relation lock field of the entry in the RATT is UNLOCKED or IN-USE. If the field is LOCKED then the request for access will be queued. A request to modify a page frame will be granted access only if the relation lock field is UNLOCKED. If the field has a value of IN-USE or LOCKED, then the request for update rights will be queued since some query packet is currently doing a retrieve or an update. Requests by processors executing query packets with identifiers less than the updating query packet will continue to be granted. All retrieve and modify requests from query packets with identifiers greater than the updating packet will be queued after it. When all query packets using the requested relation have terminated (via a counting semaphore) the relation lock field will be set from IN-USE to UNLOCKED. Then the first element from the request queue for that relaion will be removed and executed.

4.2 Controller Primitives

The primitives of the back-end controller are:

ASSIGN (QPKTi, {QP}) - assign query packet i to the set {QP} of query processors. The back-end controller must first decide on a value for n, the number of query processors to which the query packet is to be assigned. The value of n will be determined dynamically by considering:

a) The number of free query processors.

b) The channel load between mass storage and CCD page frames.

c) The size of the relations referenced by the query packet.

d) The length of the queue of query packets in the back-end controller.

e) The priority of the query packet.

After deciding how many (and which) query processors are to execute the query packet, the back-end controller must send the query packet to each processor along with format information about the relations it references. The controller must also create a task which waits for a done signal from each query processor (analagous to the join in the fork-join construct). When all query processors have signalled done, the waiting task will transmit the results of the query packet back to the host.

NEXTPAGE (QPKTi,RELj,QPk,LOCK-VALUE) - request from query processor k which is executing query packet QPKTi for the next

page of relation j. The resulting action is for the back-end controller to send the page frame number which contains the next page of relation j to query processor k using the SEND instruction. If a page fault occurs, that is, the next page of relation j is not in some page frame, then the back-end controller must choose an available page frame (which may require that another page be paged out) and then page in the requested page. Page faults (except the request for the first page) will be avoidable by doing anticipatory paging. That is, the back-end controller can always make sure that the next n pages are in memory. The value of n will depend on the number of query processors which are assigned to execute a query packet. If m query processors are assigned to execute the same query packet then n should be equal to m. This will insure that there will always be a page ready for each query processor. The LOCK-VALUE can be either retrieve-only or update.

If NEXTPAGE or GETPAGE requests a non-existent page from a temporary relation, a page frame is assigned and an entry is added to the page table for that relation. Finally, NEXTPAGE and GETPAGE must be indivisible operations.

GETPAGE(QPKTi,RELATIONj,QPk,PAGEm,LOCK-VALUE) - request from query processor k which is executing query packet QPKTi for PAGEm of relation j. The back-end controller will use the SEND instruction to send the page frame number which contains PAGEm to the requesting query processor. Handling of

page faults is described above.

SEND(QPk,PFi) - send page frame number PFi to query processor k.

DESTROY(DBNAME) - destroy data base DBNAME. After an authorization check is performed and after all queries accessing relations in the data base have terminated, (i.e. relation lock field is UNLOCKED), the back-end controller will delete all relations in the data base in addition to all information about the data base which is contained in the system catalogues.

DELETE(RELATIONi,DBNAME) - delete relation i of data base DBNAME. After an authorization check is performed and all queries accessing relation i have terminated, relation i will be removed from the data base.

CREATEDB(DBNAME,OWNER) - create a new data base which is owned by user OWNER and has name DBNAME.

CREATE(RELATIONi,DBNAME,n,[ATT1=FMT1,..., ATTn=FMTn]) - create a new relation in data base DBNAME with n attributes to be stored in the specified formats.

5.0 QUERY PROCESSOR INSTRUCTION SET

The query processor instruction set includes the basic primitives needed to support a relational data base system. A query packet, after parsing and decomposition by INGRES on the host processor, will be composed of query processor instructions. Thus, query packets can be executed directly without further compilation.

The basic primitives include:

RESTRICT	- select tuples from a relation based on a boolean search condition.
PROJECT	- eliminate the specified attributes (columns) of a relation.
JOIN	- combine two relations to form a third relation based on the equality (for equi-join) between an attribute in each relation.
MODIFY	- Modify all tuples of a relation which satisfy a specified boolean condition. Can also be used to delete a tuple.
INSERT	- insert a tuple into a relation.
COMPRESS	- compactify a relation by removing tuples marked for deletion.
AGGREGATE OPERATORS	- such as MAX, MIN, COUNT, and AVERAGE for collecting information about the data in the relation.

RESTRICT, PROJECT, and JOIN produce temporary relations which are either used by another operation in the query packet, or transmitted to the host as output to be returned to the user, or incorporated as permanent relations in the data base. Section 5.1 contains a description of the algorithm used to implement the RESTRICT operation. This description indicates how the back-end controller primitives are used to permit the simultaneous execution of a query by more than one query processor.

MODIFY and INSERT do not generate temporary relations. Both use the lock parameter of the NEXTPAGE and GETPAGE controller primitives to request exclusive use of the relation. Section 5.2 contains a description of the INSERT algorithm. The COMPRESS operator does garbage collection on a relation by removing deleted tuples.

5.1 The RESTRICT Operation

The RESTRICT operation, RESTRICT(RELf,C), applies a Boolean condition C to each tuple in RELf. Those tuples which satisfy the search criterion are placed in the temporary relation TEMPi. The RESTRICT is part of query packet QPKTi and is being executed on query processor QPj.

```
NEXTPAGE(QPKTi,TEMPi,QPj,UNLOCKED) /* request the next page */
                                   /* of relaton TEMPi*/
RECEIVE(T1PF#) /* wait for page frame number */
NEXTPAGE(QPTKi,RELf,QPj,UNLOCKED) /*request next page  */
     /* of relaton RELf which this query processor */
     /* should examine  */
WHILE (RECEIVE(PF#) ~= EOR)) /*until end of relation is indicated*/
BEGIN
    WHILE (NEXTCHAR(PF#) ~= EOP) /*until end of page */
    BEGIN
         READ NEXT TUPLE FROM PF#
         IF C(NEXT TUPLE) = T /* tuple satisfies search criterion*/
         BEGIN
              IF (T1PF# is FULL)
              BEGIN
                   NEXTPAGE(QPKTi,TEMPi,QPj,UNLOCKED)
                   RECEIVE (T1PF#)
              END
              WRITE TUPLE ON T1PF# /* output tuple onto*/
                             /* temporary relation*/
         END
    END
    NEXTPAGE(QPKTi,RELf,QPj,UNLOCKED) /*request next page of RELf*/
END
SIGNAL(DONE,QPKTi,QPj,TEMPi) /*signal  controller */
    /* that query processor QPj is done executing */
    /* query packet QPTKi and that temporary relation */
    /* TEMPi holds result of restriction */
```

5.2 The INSERT Operation

The INSERT operation, INSERT(RELf,TUPLEi), inserts TUPLEi into relation RELf. (This a simplified version that always inserts tuples on the last page of the relation.)

```
GETPAGE(QPKTi,RELf,QPj,$,LOCKED) /* request exclusive use */
    /* of relation RELf and last page */
RECEIVE(PF#) /* wait for page frame number of last page */
IF (PF# is FULL)
BEGIN
    NEXTPAGE(QPKTi,RELf,QPj,LOCKED)
         /* request next page of RELf ie. a empty page*/
    RECEIVE(PF#)
END
WRITE TUPLEi onto P
SIGNAL(DONE,QPKTi,QPj) /*signal finished with update */
```


6.0 CONCLUSIONS

In conclusion, DIRECT appears to be a promising MIMD architecture for supporting a relational data base management system through parallel processing and the use of associative memories. It can support both inter and intra-query concurrency and thus eliminates many of the cost/performance limitations of the previous SIMD architectures such as RAP. In many respects DIRECT can be viewed as a limited distributed data base system.

Before finalizing the design of the DIRECT software, we are analyzing the DIRECT architecture with a trace driven simulation model. This should enable us to estimate the potential performance of the initial DIRECT configuration and spot any potential bottlenecks.

7.0 ACKNOWLEDGEMENTS

I would like to thank William Cox and Paul Pierce for their constructive suggestions that helped shape the structure of the hardware and software of DIRECT.

REFERENCES

1. Canaday, R.H., et al. "A back-end computer for data base management," CACM 17,10,October 1974,pp.575-582.

2. Slotnick, D.L., "Logic per track devices" In Advances in Computers, Vol. 10, New York, Academic Press(1970), pp. 291-296.

3. Parker, J.L., "A logic per track retrieval system," IFIP Congress (1971), pp. TA-4-146 to TA-4-150.

4. Healy,L.D., Lipovski, G.J., and K.L. Doty, "The architecture of a context addressed segment-sequential storage," Proc. of 1972 FJCC , pp. 691-701.

5. Parhami, B. " A highly parallel computing system for information retrieval," Proc. of 1972 FJCC, pp. 681-690.

6. Minsky, N., "Rotating storage devices as partially associative memories," Proc. of 1972 FJCC, pp.587-596.

7. Lin,C.S., Smith,D., and J. Smith, "The design of a rotating associative array memory for a relational data base management application," ACM Transactions on Data Base Systems, Vol. 1,No. 1, March 1976, pp 53-65.

8. Su,S.Y.W.,Copeland,G.P., and G.J. Lipovski, "Retrieval Operations and Data Representations in a Context Addressed Disk System," Proceedings of the ACM Programming Languages and Information Retrieval Interface Meeting, 1973.

9. Copeland,G.P., Lipovski,G.J., and S.Y.W. Su, "The architecture of CASSM: A Cellular System for Non-numeric Processing," Proceedings of the First Annual Workshop on Computer Architecture, 1973.

10. Copeland,G.P. and S.Y.W. Su,"A high level data sublanguage for context addressed segment-sequential memory," Proceedings of the ACM SIGFIDET Workshop on Data Description, Access, and Control, 1974.

11. Ozkarahan, E.A.,Schuster,S.A., and K.C. Smith,"RAP - An associative processor for data base management," Proceedings of the 1975 NCC,pp.379-386.

12. Schuster,S.A., Ozkarahan,E.A., and K.C. Smith, "A virtual memory system for a relational associative processor," Proceedings of the 1976 NCC, pp. 855-862.

13. Ozkarahan, E.A., Schuster, S.A., and Sevcik, "Performance of

a Relational Associative Processor," ACM Transactions on Data Base Systems, Vol. 2, No. 2, June 1977, pp 175-195.

14. Held, G.D., Stonebraker. M.R., and E. Wong, "INGRES - a relational data base system," Proc. of 1975 NCC, Vol 44, May 1975,pp. 409-416.

15. Stonebraker, M.R., Wong, E., and P. Kreps, "The design and implementation of INGRES," ACM Transactions on Data Base Systems, Vol. 1, No. 3, Sept 1976, pp.189-222.

16. Wong,E. and K. Youssefi, "Decomposition - a strategy for query processing," ACM Transactions on Data Base Systems, Vol. 1,No. 3, Sept. 1976, pp. 223-241.

17. Codd,E.F., "A relational model of data for large shared data banks," CACM. 13,6, 1970.

18. Wulf,W.A. and G.C. Bell, "C.MMP - a multi-mini processor," Proceedings of the 1972 FJCC, Vol. 41, pp. 765-777.