# DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems

by

David J. DeWitt

Computer Sciences Technical Report #325

June 1978

# DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems

David J. DeWitt
Computer Sciences Department
University of Wisconsin

## ABSTRACT

The design of DIRECT, a multiprocessor organization for supporting relational database 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, the type and number of relational algebra operations it contains, and the size of the relations referenced. Since DIRECT is a virtual memory machine, the maximum relation size is not limited to that of the associative memory as in some other database machines. Concurrent updates are controlled through the use of locks on relations which are maintained by a controlling processor.

DIRECT is being implemented using LSI-11/03 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 databases 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 database management system on a separate microprogrammed processor with an instruction set tuned to perform database 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 database 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 database. Each of these efforts is based on the idea of a logic per track device which was first proposed by Slotnick[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, Lipovski, and Doty[4], Parhami[5], Minsky[6], Lin, Smith, and Smith[7], and Jino and Liu[8].

Currently being implementated are three back-end database

processors which exploit the logic per track idea. They 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[10,11,12] in 1973, is a cellular processor which is capable of directly supporting all three data models (relational, network, and hierarchical). The second back-end database processor is the Database Computer which has been proposed by Hsiao, Kannan, and Kerr[9]. The Database Computer uses moving head disk technology and is also intended to support the three data models.

RAP, an associative processor for database management, which efficiently supports the relational data model, has been described by Ozkarahan, Schuster, and Smith in [13,14,15]. 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 database management system. Our original goal was to design a parallel processor system which does not waste a large percentage of its processing potential by permitting processors to be idle, particularly 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 database 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 multiprocessor organization for supporting relational database management systems. 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, the size of the relations it references, and the type and number of relational algebra operations included in the query.

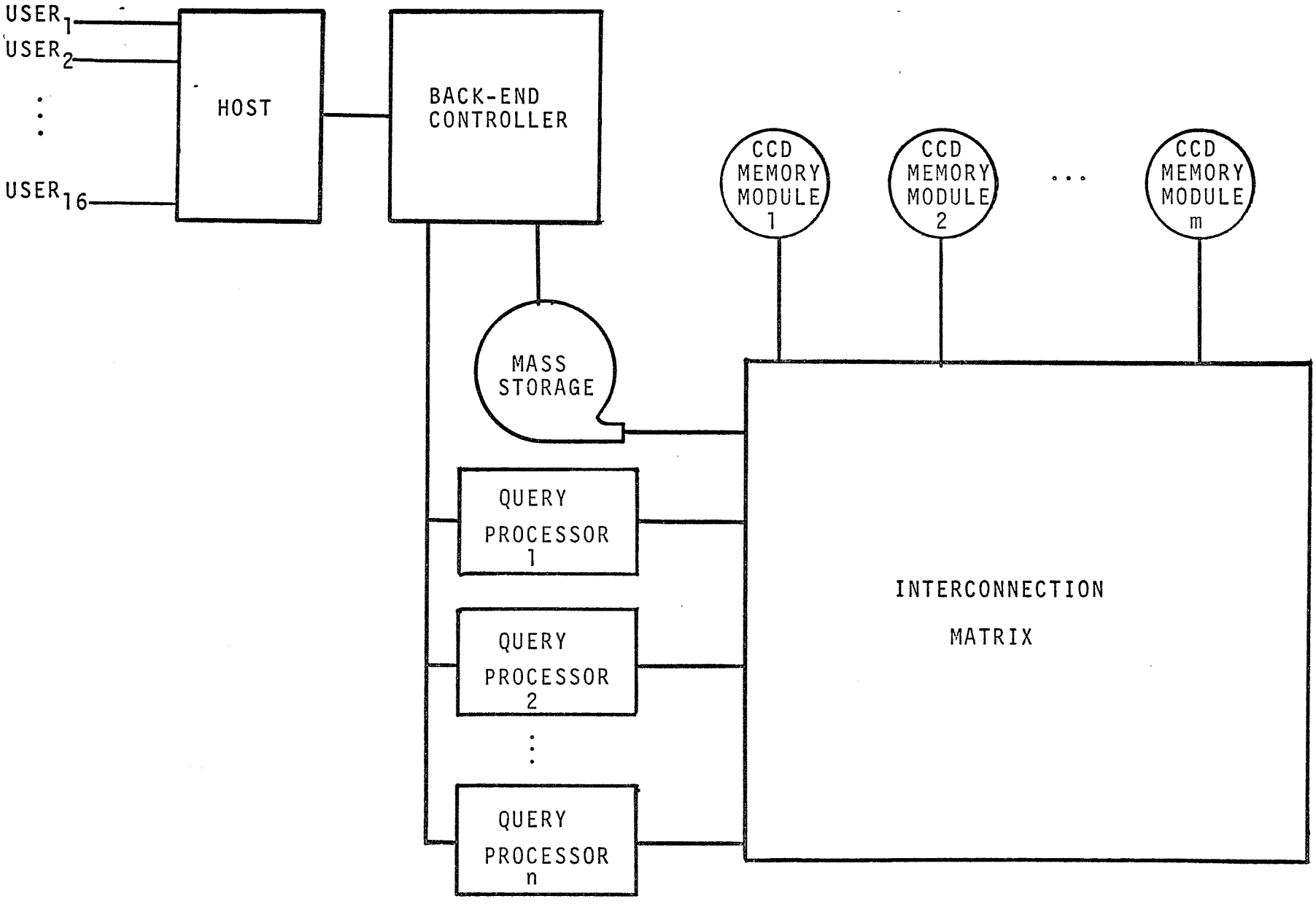
3. Relation size is not limited by the size of the associative memory.
4. Control of concurrent updates through the use of locks on relations.
5. Complete compatibility with INGRES [16,17,18], an existing relational database system.

## 2.0 DIRECT SYSTEMS ARCHITECTURE

### 2.1 Introduction

When operational, the complete DIRECT system will consist of six 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, an interconnection matrix between the set of query processors and the set of CCD memory modules, and a mass storage device. 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 database system will log onto a modified version of INGRES, a relational data base system[16,17,18], and proceed in the normal manner. However, when the user wishes to execute a query INGRES will first compile the user query into a sequence of relational algebra operations which we call a "query packet". After compilation, the query packet is sent by INGRES




DIRECT SYSTEM ARCHITECTURE
Figure 2.1

to the DIRECT back-end controller over a DMA link.

In addition to simplifying the implementation of DIRECT, INGRES will also 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 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 that 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 portions of 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 in-

struction 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 (CCD) chips. 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 to test the design of DIRECT by mimicking a large database 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 in the back-end controller.

## 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 multiport CCD memory can have a bandwidth of 6 megabytes/second while the bandwidth of an LSI-11 DMA interface is .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 [20] the switch introduced an additional delay of 250 ns to the memory cycle time of 250 ns.

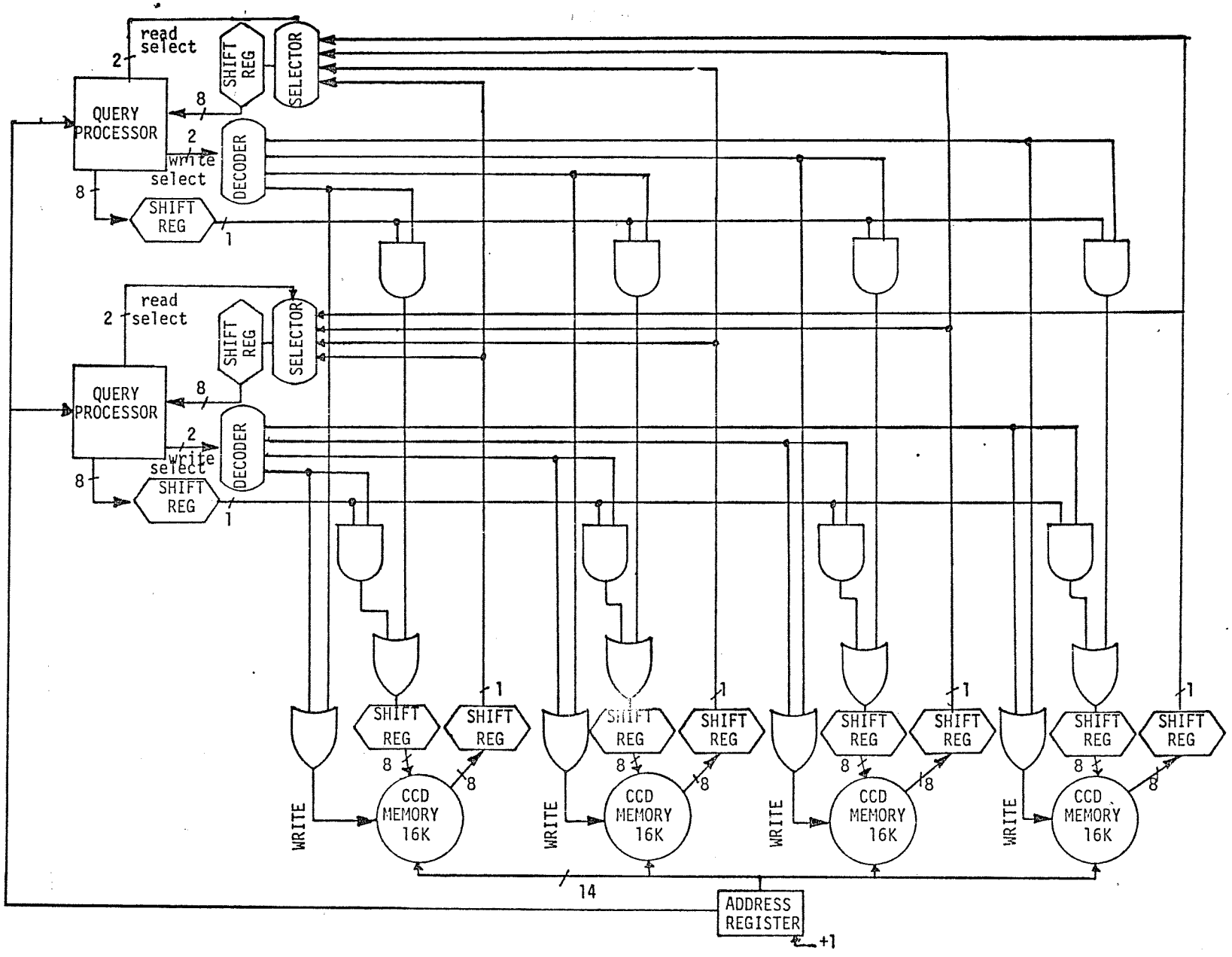
In a traditional cross-point switch, the processors are the active components and the memories are the passive components. In DIRECT, the roles of the processsors and memories have been interchanged. Instead of responding to a request for a word from a processor, each memory element acts as a producer and each processor acts as a consumer. Each CCD memory element continuously "broadcasts" its contents. Whenever a processor wishes to examine the page of a relation which is resident in some CCD memory module, it simply "listens" to what that memory is broadcasting. Hence, unless a processor is updating a page of a relation, any number of processors can "listen" to the same memory element simultaneously.

Using this approach we have designed a cross-point switch which can be used as the interconnection matrix in DIRECT and yet can 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 multiprocessor 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. Since a query processor will always search an entire page of a relation it does not need to wait until the beginning of the page before starting to examine it. The processor can simply start on a tuple boundary and continue until the starting point is reached again. (The starting point can be detected by comparing the Memory Address Register with the address of the first tuple read). Thus, a latency time of 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




A 2x4 DIRECT Configuration

Figure 2.2

another page frame. Since the time to examine a complete page is .012 seconds (16K bytes x 750ns/byte), 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 page request from the query processor to the DIRECT controller 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 attempt to write onto the same page simultaneously are handled in software by the back-end controller. A discussion of the tech-

nique used can be found in Section 4.5.

## 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 eight LSI-11/03 query processors and thirty-two 16K byte CCD page frames.

# 3.0 GENERAL SOFTWARE CONSIDERATIONS

## 3.1 Relational Data Model

The data model chosen for DIRECT is the relational model which was introduced by Codd [19]. 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 relationships between entities.

## 3.2 Page Format

Each relation in the database 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 tuple 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 al-

lowed 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 versus 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 in a page frame of 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 access 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 every relational query is a temporary relation which the user might wish to either add to the database or use in subsequent queries. If this is the case, the back-end controller can simply make the temporary relation permanent.

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 back-end controller will transfer tuples from the temporary relation 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 path between the processors. If mark bits were used, the relation(s) which was (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.

Finally, consider the performance of the two approaches in terms of the number of "revolutions" required to perform a restrict operation (see Example 3.1). (A revolution is the time required to read or write one page of a relation.) In RAP, one revolution is required to mark tuples which satisfy the search criterion and a second revolution is required to read the marked tuples. In DIRECT one "revolution" is required to extract the tuples which satisfy the search criterion. Only when the query processor has filled its output buffer (perhaps after examining

many source pages) will a second revolution be necessary. When the result relation is read and returned to the host, a final access will be required. Hence, the worst case will require three revolutions. In practice, however, the actual number of revolutions required will be two plus a small fraction. While more CCD memory 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 Section 4.4.3).

## 3.4 Query Packet Format

The types of query packets which are received by the back-end controller can be divided into two classes based on information contained in the packet header. Class I contains those INGRES commands which will be executed by back-end controller utility routines (see Section 4.2). Class II contains those queries which will be executed by a collection of query processors. In this section, we will discuss the structure of the second class of query packets.

Each Class II query packet is structured as a tree (see Figure 4.8). The leaf nodes of this tree correspond to relations referenced in the query. Each of the non-leaf nodes contains a relational algebra operator which is to be applied to its children. The set of operators supported include the traditional

operators such as JOIN, PROJECT, RESTRICT, UNION, INTERSECTION, and CROSS-PRODUCT as well as aggregrate operators such as MAX, MIN, COUNT, etc. DIRECT does not need to support the relational algebra operation DIVIDE since the syntax of the INGRES query language does not include the universal quantifier.

While the logical structure of a query packet is a tree, the query packet can be directly executed by one or more query processors without interpretation or further compilation of the packet.

One of the unique features of DIRECT is the general structure which is common to all the relational algebra operators supported. This structure, as we will demonstrate in Section 4.4, permits a query packet to be assigned to any number of query processors without modifying the packet. Furthermore, this operator structure permits additional query processors to be dynamically assigned to execute a packet during execution of the packet by other query processors. We will conclude this section with an example of a simple INGRES query and the corresponding query packet. This example will be used in Section 4.4 to illustrate how our flexible query processor assignment scheme is implemented.

Given the SUPPLIER relation shown in Figure 3.4 and the INGRES query to find the names of all suppliers who do business in N.Y.: RETRIEVE (SUPPLIER.NAME) WHERE SUPPLIER.CITY = "N.Y." Then, the compiled query packet for this query has the structure shown in Example 3.1. Note that the packet never asks for an ex-

plicit page of a relation. Instead, the next page of the relation is always requested. This structure, when combined with a monitor for every relation referenced by a packet, permits us to dynamically assign query processors to executing query packets (see Section 4.4).

SUPPLIER Relation

| NUMBER | NAME | CITY |
|---|---|---|
| 10 | JONES | N.Y. |
| 20 | SMITH | CHICAGO |
| 15 | LeBLANC | ATLANTA |
| 74 | WHITE | DALLAS |
| 101 | RICE | N.Y. |
| 102 | JONES | BOSTON |

Figure 3.4

CITYQPKT

```
LOCK(SUPPLIER,READ)
CREATE NYSUPPLIERS    /* create result relation */
DO FOREVER
BEGIN
     - ASK BACK-END CONTROLLER (BEC) FOR THE NEXT_PAGE OF RELA-
     TION SUPPLIER
     - WAIT FOR THE BEC TO RETURN THE PAGE FRAME NUMBER
     - IF THE BEC RETURNS "END OF RELATION" QUIT AND SIGNAL DONE
     OTHERWISE
          - READ NEXT PAGE OF RELATION SUPPLIER INTO LOCAL MEMORY
          FROM THE PAGE FRAME
          - EXAMINE ALL TUPLES READ IN
               - COPY EACH TUPLE THAT SATISFIES THE RESTRICTION
               SUPPLIER.CITY = "N.Y." INTO A LOCAL PAGE BUFFER
               - WHEN THE BUFFER IS FULL
                    - ASK (BEC) FOR THE NEXT_PAGE OF RELATION
                    NYSUPPLIERS
                    - WAIT FOR BEC TO RETURN A PF#
                    - WRITE BUFFER INTO PF#
END
UNLOCK(SUPPLIER)
```

Example 3.1

# 4.0 FUNCTIONS OF THE BACK-END CONTROLLER

## 4.1 Introduction

The Back-End Controller (BEC) is responsible for receiving queries from the host, distributing the queries to the query processors for execution, and returning the results to the host. Communication with the host to receive queries and return results is straightforward and will not be discussed further. In this section, we will describe those functions of the BEC which deal with query execution in DIRECT.

## 4.2 Database Utilities

The class (Class I) of legal INGRES commands including CREATDB, DESTROYDB, CREATE, DESTROY, PRINT, and COPY is executed in the BEC by the appropriate database utility routines.

A CREATDB/DESTROYDB command to create/destroy a database causes a new entry to be added/deleted from the database catalogue shown in Figure 4.1.

DATABASE CATALOGUE
[One Entry per Database]

| DATABASE NAME | UNIXID OF OWNER |
|---|---|
| | |
| | |
| | |

Figure 4.1

After a user on the host executes a CREATDB command, the next step in the database creation process is for the user to invoke INGRES with the name of the database and create the relations desired. Whenever a user invokes INGRES with the UNIX command "ingres dbname", the query packet "open dbname userid" is sent by the host to the back-end controller. The BEC responds by adding an entry to the OPENDB table (Figure 4.2). Each subsequent query packet sent to the BEC is identified only by the UNIX id of the user. Hence, the proper database can be determined by using the UNIX id to search the OPENDB table. When the user exits INGRES, a "close dbname userid" query packet is sent by the host to the BEC. When the BEC receives such a packet, it will remove the appropriate entry from the OPENDB table.

OPENDB TABLE
[One Entry For Each Active User]

| UNIXID OF USER | DATABASE NAME |
|---|---|
| | |
| | |
| | |

Figure 4.2

The INGRES command "CREATE relation-name", when executed by the BEC, adds entries to three tables. First a general descriptor of the new relation is added to the RELATION DESCRIPTOR table (Figure 4.3).

RELATION DESCRIPTOR TABLE
[One Table Per Database]
[One Entry Per Relation]

| RELATION NAME | ATTRIBUTE TABLE POINTER | PAGE TABLE POINTER | RELATION LOCK | TUPLE WIDTH | NUMBER OF ATTRIBUTES |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |

Figure 4.3

Next a page table (Figure 4.4) is created for this new relation (a relation is essentially equivalent to a segment) and the Page Table Pointer in the Descriptor Table entry for the relation is set to the address of the Page Table for the new relation. The page table for the new relation is initially empty.

Finally, an Attribute Catalogue Table, as shown in Figure 4.5 is created for the new relation. This table describes the format of every attribute in the relation.

When a "DESTROY relation-name" command is executed, the BEC deletes the appropriate tables and table entries after all queries which are currently accessing those tables have terminated (see Section 4.5).

The BEC executes the PRINT and COPY (to the host) commands in an identical fashion. In both cases, a utility program extracts the tuples from the database, places them in a packet with the appropriate header, and then sends the packet to the host.

According to the contents of the header, when the host receives the packet it will either copy the tuples to the user's terminal or to a UNIX file. COPY commands to move tuples from the host to the back-end are handled in an analogous fashion.

PAGE TABLE FOR RELATION J
[One Per Relation]

| PAGE | PRESENCE BIT | DIRTY BIT | FRAME NUMBER | DISK ADDRESS |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |

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

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

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

Disk address:
Disk address where page i of relation j can be found

Figure 4.4

ATTRIBUTE CATALOGUE
[One Per Relation]

| ATTRIBUTE NAME | OFFSET IN BYTES | TYPE | LENGTH |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Figure 4.5

## 4.3 Query Processor Allocation

### 4.3.1 Current Approach

When the BEC receives a query packet which contains a Class II query, the packet is added to the query packet queue (QPKTQ) which is ordered by priority. The function of the query processor allocation (QPA) process is to control the allocation of query processors to query packets.

When the QPA process decides to remove a packet from the QPKTQ for execution it places the identifier of the packet in QPKTX (Figure 4.6), the list of all executing query packets. Next the QPA process examines the packet and attempts to estimate an "optimal" query processor allocation for this packet. By optimal we mean that assignment of more than the optimal number of query processors to the packet will not decrease the execution time for the packet. For example, assume that the query packet joins (a join is a limited cross product) relation A and relation B and that relation A is N pages long and relation B is M pages

long. Then, the optimal query processor allocation for this packet is MAX(M,N). This allocation will require MIN(M,N) time units where a time unit is the time required to join one page of A with one page of B. In this example, the optimal allocation is truly optimal. However, consider the packet which first joins A and B and then joins the resulting relation with C. Since it is impossible to predict the size of A join B, it is impossible to determine exactly how many query processors should be assigned. We plan to investigate several heuristics for determining the "optimal" query processor allocation. The inputs to these heuristic functions might include:

- The size of each relation referenced by the query
- The type and number of relational algebra operations used in the query
- Estimates of intermediate relation sizes

The final entry in the QPKTX table indicates how many query processors are currently assigned to each executing query packet.

QPKTX
[One Entry for Each Executing Query Packet]

| QPKT# | OPTIMAL QP ALLOCATION | CURRENT QP ALLOCATION |
|---|---|---|
| | | |
| | | |
| | | |

Figure 4.6

While there are many viable QPA algorithms, the one we have

chosen attempts to give each query packet its "optimal" allocation of query processors. As shown in Figure 4.7, whenever query processors are available, the algorithm first scans QPKTX for a packet which has less than its optimal allocation. Only if all the packets are at their optimal level is the multiprogramming level increased by removing a packet from the QPKTQ. Therefore, at any instance in time, at most one entry in QPKTX will have less than its optimal allocation.

QUERY PROCESSOR ALLOCATION ALGORITHM

WHEN QUERY PROCESSORS ARE AVAILABLE

- ATTEMPT TO ADD PROCESSORS TO THE QPKT IN QPKTX WHICH HAS LESS THAN ITS OPTIMAL NUMBER OF PROCESSORS
- IF PROCESSORS ARE STILL AVAILABLE
    - REMOVE NEXT QPKT FROM QPKTQ
    - ADD QPKT TO QPKTX
    - ATTEMPT TO GIVE IT ITS OPTIMAL ALLOCATION OF QUERY PROCESSORS

Figure 4.7

Other query processor allocation schemes might attempt to give service to more packets by including in the calculation of the "optimal" QPA, a term for the priority of the query and the length of the QPKTQ when the packet is removed from the queue.

We plan to investigate these alternatives.

4.3.2 Query Processor Allocation using Data Flow Techniques

Under the current approach, during the execution of a query, query processors will be executing nodes in the query packet tree from at most two adjacent levels at any one time. For example, consider the query tree in Figure 4.8. Initially all the query processors assigned to this packet will execute the RESTRICTION on relation A. When a query processor receives a "End of Relation" message from the controller on a NEXT_PAGE of relation A, that query processor will automatically (without intervention of the BEC) begin the join of A' with B. When the join of A' and B is finished, then the join of A'B' with C will begin.

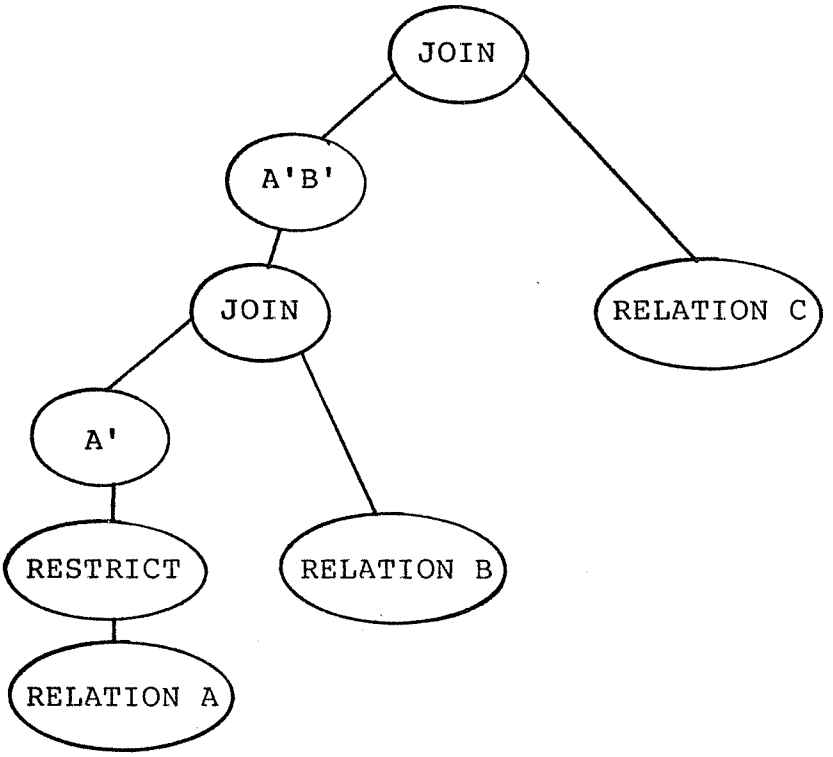



Figure 4.8

While this approach minimizes the requirements placed on the BEC, it potentially wastes query processors if the optimal allocation turns out to be a bad guess. We are currently investigating an alternative approach for query processor allocation in which only simple operations (JOINS, RESTRICTIONS, PROJECTIONS, etc.) are given to the query processors and not complete query packets. Then, instead of estimating the optimal requirements of an entire query, an exact value can be determined for each step in the query. Furthermore, we can now control exactly what step of a packet each query processor is executing. For example, each time the join of A' and B produces a new page of A'B', then an additional query processor can be assigned to join that page with relation C.

While the data flow approach will certainly require a stronger controller, it should increase query processor utilization. In addition, it should also decrease the page traffic in DIRECT. This will occur because as soon as a page of A'B' is produced, another query processor can begin joining it with C. Hence, the likelihood that the page will be paged out is reduced. We are currently comparing this data flow machine approach with the standard QPA scheme to determine the effect of each on system throughput in DIRECT.

## 4.4 CCD Memory Management

The function of the CCD memory management process is three fold:

- Respond to a NEXT_PAGE request from a query processor
- Respond to a GET_PAGE request from a query processor
- Schedule the movement of pages from relations between CCD memory page frames and mass storage as the result of NEXT_PAGE and GET_PAGE operations.

### 4.4.1 The NEXT_PAGE Operation

The form of a NEXT_PAGE request is

NEXT_PAGE(QPKTi,RELj,QPk)

This is a request from query processor k which is executing query packet i for the next page of relation j. The resulting action is for the BEC to send the page frame number which contains the next page of relation j to query processor k. A page fault can occur if the required page is not in some CCD page frame. Handling of page faults is the same for both the NEXT_PAGE and GET_PAGE operation and is discussed in Section 4.4.3.

Since a query packet can be assigned to any number of query processors, there must be a way to prevent two simultaneous NEXT_PAGE requests from different query processors executing the same packet to be resolved correctly. This is handled by the use of the Query Packet Task Table which is shown in Figure 4.9. The Query Packet Task Table has one entry for each relation referenced by each executing query (i.e. each query in QPKTX). Asso-

ciated with each entry is a monitor[21] which controls access to the table entry. Initially, the currency pointer for the relation is set to zero.

QUERY PACKET TASK TABLE

| QPKT # | RELNAME | CURRENCY POINTER | POINTER TO PAGE TABLE FOR RELATION |
|---|---|---|---|
| | | | |
| | | | |
| | | | |

Figure 4.9

As an example, assume that query processor allocation initially assigns the CITYQPKT query in Example 3.1 to query processors QP5 and QP8. Assume that the order in which the monitor for the SUPPLIER relation receives requests is:

NEXT_PAGE(CITYQPKT,SUPPLIER,QP5)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP8)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP5)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP5)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP8)

Then, the pages of the SUPPLIER relation examined by QP5 will be 1,3, and 4 and the pages of the SUPPLIER relation examined by QP8 will be 2 and 5. Assume that after the last NEXT_PAGE request above, the QPA algorithm assigns an additional query processor (QP1) to the CITYQPKT. Now there will be three query processors

requesting pages from relation SUPPLIER. If the request stream is as follows:

NEXT_PAGE(CITYQPKT,SUPPLIER,QP1)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP5)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP1)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP8)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP5)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP1)

NEXT_PAGE(CITYQPKT,SUPPLIER,QP8)

and the SUPPLIER relation consists of nine pages then:

QP1 will examine page 6

QP5 will examine page 7

QP1 will examine page 8

QP8 will examine page 9

QP5 will receive "End of Relation"

QP1 will receive "End of Relation"

QP8 will receive "End of Relation"

By having a monitor associated with each entry in the query packet task table and by using the NEXT_PAGE concept, we can dynamically assign additional query processors to a query packet that is already partially executed. In the case of a complex query such as that in Figure 4.8, it may be the case that the additional query processors are added after the other query processors have finished the restrict of A. Our approach handles this situation correctly. When the newly assigned query processors attempt to restrict A, they will get "End of Relation" immediately

and will therefore proceed to begin the join of A' with a page of B. If that join is also finished, they will proceed to the final join.

### 4.4.2 The GET-PAGE OPERATION

The form of the GET_PAGE Operation is

GET PAGE(QPKTi,RELj,QPk,PAGEm)

It represents a request from a query processor for PAGEm of RELj. Its use in join operations will be demonstrated in Section 5.1. No monitor is needed to coordinate GET_PAGE requests.

### 4.4.3 PAGE FAULTS IN DIRECT

The third task of the CCD memory management process is to handle page faults by scheduling page transfers between CCD memory page frames and mass memory. A page fault occurs when a requested page is not in some CCD memory module. In DIRECT these page faults will be, to a large extent, avoidable by doing anticipatory paging. In the CITYQPKT (Example 3.1), for example, the controller knows that the entire SUPPLIER relation will be examined and hence the reference string of the CITYQPKT is known in advance. By using the currency pointer for the SUPPLIER relation in the query packet task table, the SUPPLIER relation page table, and the current query processor allocation from QPKTX, the controller can determine how far ahead it should attempt to be in order to insure that there will always be a page ready for each query processor which is executing the packet.

## 4.5 DIRECT Concurrency Control

Since DIRECT supports inter-query concurrency, it must provide a mechanism for controlling concurrent updates and retrievals so that each user transaction begins and terminates with a consistent database. The approach we have selected is to lock whole relations. Although locking pages was considered (because of the apparent potential for increased concurrency between transactions), examination of both approaches revealed that they are equivalent. We will now informally demonstrate this fact.

Consider, for example, two transactions each consisting of one query packet, QPKT1 and QPKT2. Assume, without loss of generality, that QPKT1 intends to update relation A and QPKT2 intends to read relation A. If locking is performed at the relation level, and QPKT1 locks relation A, then QPKT2 will be blocked until QPKT1 releases its lock on A. Thus, there is no overlap in the processing of QPKT1 and QPKT2 (as least with regard to relation A). (If QPKT1 and QPKT2 both wanted to read relation A they could have proceeded concurrently.)

Next consider locking at the page level. Because of the sequential manner in which pages are accessed in DIRECT, the first access to relation A by either QPTK1 or QPTK2 will be to page 1 of the relation. If a NEXT_PAGE (A) operation from QPKT1 locks page 1 for writing (exclusive access), NEXT_PAGE (A) from QPKT2 will be blocked. Eswaran et.al. [22] have shown that each transaction must be two phased if database consistency is to be guaranteed. Thus, a transaction cannot request any new locks

after it has released a lock. For our example, this requirement implies that QPKT1 cannot release its lock on page one of A until it has finished locking all pages of A. Hence, QPKT2 will be blocked until QPTK1 releases its locks on A. As with locks on relations, there is no concurrent processing of the two transactions. Therefore, since packets always access relations in a sequential fashion starting with page 1, the lock field for page 1 of the relation has the same effect as a lock for the entire relation.

In the initial DIRECT implementation each transaction will begin with all its lock requests. If the BEC cannot grant all lock requests, the controller will release all locks the transaction had obtained. Then, the controller will wait a random amount of time before again trying to satisfy the transaction's requests. While this approach reduces the potential concurrency, it avoids the problems associated with detecting and resolving deadlock. Later, after rollback of a partially completed transaction is implemented (through the use of before images and duplicate page tables), we intend to modify the locking scheme so that each transaction locks relations as needed. Then when deadlock is detected (through the use of time-outs on locks), one of the transactions will be rolled back.

One important point about concurrency control in DIRECT is that two or more query processors which are executing the same query packet and outputting tuples to a temporary relation will not interfere with each other since the lock associated with the

temporary relation is owned by the query packet and not a query processor executing the packet. Also, when the query processor uses the NEXT_PAGE operator on the temporary relation it is not asking for the next page of an existing relation, but rather, it is informing the controller that it has a new page. Thus, the NEXT_PAGE operation for write operations must also modify the page table in the appropriate fashion.

## 5.0 QUERY PROCESSOR INSTRUCTION SET

The query processor instruction set includes the basic primitives needed to support a relational database 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.

UNION - forms the union of two union compatible relations (relations with identical attributes)

INTERSECTION - forms the intersection of two union compatible relations

CROSS-PRODUCT - performs the cross-product of two relations

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, JOIN, UNION, and CROSS-PRODUCT produce temporary relations which are either used by another operation in the query packet, transmitted to the host as output to be returned to the user, or incorporated as permanent relations in the database. Section 5.1 contains a description of the algorithm used to implement the JOIN operation.

## 5.1 The JOIN Operation

The JOIN of attribute a of relation A with attribute b of relation B, JOIN(A,a,B,b,C), is the set (relation C) of all tuples such that t is a concatentation of tuple tA from A and tB from B where tA.a = tB.b.

```
CREATE C
DO FOREVER
BEGIN
     - ASK BACK-END CONTROLLER (BEC) FOR THE NEXT_PAGE OF RELA-
     TION A
     - WAIT FOR BEC TO RETURN A PAGE FRAME NUMBER (PF#)
     - IF BEC RETURNS "END OF RELATION" (EOR) PROCEED TO NEXT
     OPERATION IN QUERY PACKET
       OTHERWISE
          /* JOIN THIS PAGE OF A WITH EVERY PAGE IN B */
          - READ NEXT PAGE OF A FROM PF#
          - SET I EQUAL TO 1
          - SET END OF_B TO FALSE
          WHILE (END_OF_B = FALSE)
          BEGIN
               - GET PAGE I OF RELATION B FROM BEC
               - WAIT FOR BEC TO RETURN PF#
               - IF BEC RETURNS "EOR" SET END OF_B = TRUE
                 OTHERWISE
                    - READ PAGE I OF B FROM PF#
                    - JOIN CURRENT PAGE OF A WITH PAGE I OF B
                    - WRITE RESULTING TUPLES INTO A BUFFER
                    - WHEN THE BUFFER IS FULL
                         - ASK BEC FOR NEXT_PAGE OF C
                         - WAIT FOR PF# FROM BEC
                         - WRITE OUTPUT BUFFER ONTO PF#
                    - INCREMENT I
               END
END
```

## 6.0 CONCLUSIONS

In conclusion, DIRECT appears to be a promising MIMD architecture for supporting a relational database 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.

We are proceeding on three tasks in parallel. The LSI-11s have arrived and are being assembled. The CCD memory modules and

interconnection matrix have been ordered and should arrive in September 1978. With regard to software development, we have finished our modified version of INGRES and are beginning work on the database utilities of the back-end controller. Finally, we have developed a simulation model of DIRECT and are using it to evaluate different query processor allocation algorithms, alternative techniques for prepaging relations, and a comparison of SIMD and MIMD architectures for database machines.

## 7.0 ACKNOWLEDGEMENTS

I would like to express my appreciation to William Cox for his assistance in designing DIRECT and developing a simulation model of its structure. I would also like to thank Haran Boral and Kevin Wilkinson for their suggestions about query processor allocation and concurrency control and Ken Barry for his work on our modified version of INGRES.

REFERENCES

1. Canaday, R.H., et al. "A back-end computer for database 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 database management application," ACM Transactions on Data Base Systems, Vol. 1,No. 1, March 1976, pp 53-65.

8. Jino,M. and Jane W.S. Liu, "Intelligent Magnetic Bubble Memories," Proceedings of the Fifth Annual symposium on Computer Architecture, April 1978, pp. 166-174.

9. Hsiao, D.K., Kannan, k. and Kerr,D.S., "Structure Memory Designs for a Database Computer," Proceedings of the ACM Annual Conference, 1977, Seattle.

10. 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.

11. 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.

12. 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.

13. Ozkarahan, E.A.,Schuster,S.A., and K.C. Smith,"RAP - An as-

sociative processor for database management," Proceedings of the 1975 NCC,pp.379-386.

14. 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.

15. 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.

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

17. 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.

18. 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.

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

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

21. Hoare, C.A.R., "Monitors: An Operating System Structuring Concept," CACM Vol. 17,10, Oct. 1974, pp. 549-557.

22. Eswaran, K.P., et.al., "The Notions of Consistency and Predicate Locks in a Database System," CACM Vol. 19,11, Nov. 1976, pp. 624-633.