PERFORMANCE ANALYSIS OF ALTERNATIVE DATABASE MACHINE ARCHITECTURES

by

Paula B. Hawthorn

and

David J. DeWitt

Computer Sciences Technical Report #383

March 1980

ABSTRACT

The rapid advances in the development of low-cost computer hardware have led to many proposals for the use of this hardware to improve the performance of database management systems. Usually the design proposals are quite vague about the performance of the system with respect to a given data management application. In this paper we predict the performance of several of the proposed database management machines with respect to several representative INGRES queries. The systems analyzed in this paper include are associative disks, RAP, CASSM, DBC, DIRECT, and CAFS. We demonstrate that no one database machine is best for executing all types of queries. We will also show that for one class of queries the degree of performance improvement achieved does not warrant use of a database machine.

1. Introduction

The rapid advances in the development of low-cost computer hardware have led to many proposals for the use of this hardware to improve the performance of database management systems. Usually the design proposals are quite vague about the performance of the system with respect to a given data management application. In this paper we predict the performance of several of the proposed database management machines with respect to several representative INGRES [STON76] query streams. We will demonstrate that no one database machine is best for executing all types of queries. We will also show that for one class of queries the degree of performance improvement achieved does not warrant use of a database machine. We hope that these results will guide the design of future database machines so that a significant degree of performance improvement can be achieved for all classes of queries.

The term "data management machines" is used here to describe any special-purpose hardware built to enhance the performance of data management systems. The systems analyzed in this paper include both those actually built and those that remain designs on paper. The systems analyzed are associative disks [LANG78, LIN76, SLOT70]; RAP [OZKA75, OZKA77, SCHU78]; CASSM [SU75, LIPO78, SU79]; DBC [BAN78A, BAN78B, BAUM76, HSI76A, HSI76B, KANN78]; DIRECT [DEWI78, DEWI79a, DEWI79b, BORA79a, BORA79b]; and CAFS [BABB79, COUL72].

Since most of the proposed designs are paper machines, we had to make certain assumptions about the characteristics of each

machine in order to make the performance comparisons fair and consequently meaningful. These assumptions are discussed in Section 2. Section 3 predicts and compares the performance of each machine when executing three benchmark retrieval queries. Because several of these systems have continued to evolve since our analysis began, the results presented may not reflect the performance characteristics of the latest versions of each design. The version of each system analyzed in this paper is based on the references cited.

While the analysis techniques we employ could be used for benchmarking the update performance of each machine, we have not done so in this analysis. This is because updates take a relatively short time to execute (compared to projects and joins) and basically consist of a retrieval operation followed by modification (or deletion) of the selected tuples.

The performance of a conventional computer system is also included in the comparison. Therefore, we are able to explore those design differences in the database machines that affect performance as well as the potential improvement of such machines over conventional systems. Our conclusions are presented in Section 4. Appendix I contains a brief description of the architecture of each machine.

2. Specification of database machine characteristics

The performance analysis presented in Section 3 consists of an analytical comparison of the database machines. The comparisons are based on assumptions about the physical characteristics

of the machines. These assumptions are discussed in this section.

2.1. Cell storage media

For the purposes of comparing the systems, it is assumed that each database machine except RAP and DIRECT uses moving head disks as the data cell storage media. Although associative disk machines and CASSM were designed for fixed-head disks, the use of the same media for all the systems helps make the analysis of the design differences independent of the storage media. Since RAP and DIRECT are caching systems, they are assumed to have the faster, but sequential, storage media that CCDs provide.

2.2. Physical specifications for disks

The physical specifications for the disks are assumed to be those of the Ampex 9200 disk drives. The rotation time of the moving head disk is denoted by DROT which is assumed to be .0167 seconds. The average access time, DAVAC, is .030 seconds. The block size, BSIZE, is equal to 512 bytes. The data transfer rate of the disk, through the controller and channel, is DRATE = .0012 seconds per 512 byte block. The time for a cell processor to read a block is not the same as DRATE because the block goes directly to the cell processor, instead of through the bus and the operating system protocols into the computer's main memory. This time, DREAD, for a Ampex 9200 disk would be .0008 seconds per block (.0167 sec. per rotation / 22 blocks per track). DRATE is generally longer than DREAD in most computer systems due to channel and memory interference. DCYL, the number of 512-byte

blocks per cylinder, is equal to 418 blocks (22 blocks per track * 19 tracks per cylinder). Table 2.1 summarizes these parameters.

Table 2.1 Disk parameters and values

parameter	meaning	value
BSIZE	block size	512 bytes
DROT	disk rotation time	.0167 seconds
DAVAC	average access time	.0300 seconds
DRATE	data rate to host	.0012 / block
DREAD	cell read time	.0008 / block
DCYL	# 512-byte blocks/cylinder	418 blocks

2.3. CCD physical specifications

For the CCD pages, we assumed that both RAP and DIRECT use the high-speed, smaller pages specified for DIRECT [DEWI78]. This assumption is made simply to keep the obviously changeable parameter of page size from entering the analysis. The size of the pages (CSIZE) is taken to be 16K bytes. The scan time for an entire page is CSCAN = .012 seconds for the CCD pages. These parameters are summarized in the following table.

Table 2.2 CCD parameters

parameter	meaning	value
CSIZE	size of CCD page	16K bytes
CSCAN	page scan time	.012 seconds

2.4. Cell processor performance

Another assumption that we have made is that the cell processors are fast enough to keep up with the rotational speed of the cell storage media. Independent of the storage media, the cell processors must:

1) Store the tuples in a buffer until the entire block is read and checked for errors.

2) If the query includes qualifications, compare the attribute values of each record in the block to the values in the query qualification.

3) If a tuple is a match, any of the following:

   a) Perform any arithmetic functions specified in the query.

   b) Transmit the tuple from the buffer to the host over a bus.

   c) Transmit the specified attributes to the host.

In order to operate optimally, this work must be performed at the rotational speed of the cell. This rate is .0008 seconds per 512-byte block on most standard disks. Therefore, the processors have 1.5 microseconds per byte to process the block. If the storage media is CCD or RAM pages, the time is .73 microseconds per byte. Assuming that it takes three instructions to examine a byte and that every byte must be examined, then a cell processor must be about a 2 MIP processor for the disk tracks and a 3 MIP processor for the CCD cells.

In practice, however, the performance requirements of the cell processors for both storage media can be significantly reduced through the following techniques:

1) All 512 characters in the block do not have to be manipulated.

By storing the offsets of the attributes to be tested in cell-processor registers, then the only data that has to be examined by the cell processor are those attributes which are referenced in the query. If both the ratio of the number of bytes in the attributes involved in the qualification to the total number of bytes per block and the ratio of the number of successful comparisons to the total number of tuples are small, then the cell processors can be slower and still keep up with the cell cycle time. Furthermore, the data can be stored in the processor's memory through a DMA device, without taking processor time.

2) In [SU75 and BAUM76] it is observed that very fast cell processors can be made inexpensively because they are simple, single function devices and do not need the functionality (e.g. addressing and protection modes, interrupt handling, etc.) of a conventional processor. For example, [OZKA77] reported that the RAP cell processors were bound in speed by the data rate of the CCD pages, not the processor speed.

3) Finally if the cell processor is not fast enough to complete query qualification in the time allotted, the next block will not be read into the cell processor's memory. In this case the cell processor must wait a full revolution of the cell storage media before reading the next block.

2.5. Number of cells

Ampex 9200 moving-head disks contain 19 tracks per cylinder (the 20th track is used as a timing track). Consequently, the number of cells for those database machines that are disk-

resident, NDCELL, is assumed to be 19.

The number of 16K byte data pages in the CCD cache for RAP and DIRECT is assumed to be 16. Since RAP tightly couples the processors to the data cells, the number of cell processors for RAP, NRCELL, is also 16. This seems to be a fair assumption since the RAP processors are faster and slightly more powerful than those cell processors used for the database machines which can employ moving head disks.

The cells in DIRECT are not tightly coupled to the cell processors. To reflect DIRECT's usage of general purpose processors rather than the specialized, less functional processors of the other systems NDRP was chosen to be 8. The results of [BORA79a] indicate that a data cell to cell processor ratio of 2:1 is sufficient for high performance. We have assumed, however, that all cell processors are 1 MIP processors (see Section 3.4). It is important to keep in mind that DIRECT and RAP cannot process data directly on the disk but instead rely on a disk cache constructed from CCD cells which must be loaded from the disk before processing can begin. We feel that reducing the number of cell processors available from 19 for the associative disk systems, to 16 for RAP and 8 for DIRECT sufficiently compensates for the corresponding increase in functionality.

The following table summarizes these assumptions.

Table 2.3 Cell parameters and values

parameter	meaning	value
NDCELL	Number of cells for CASSM, CAFS, DBC	19
NDRP	Number of cell processors in DIRECT	8
NDPAG	Number of data cells in DIRECT	16
NRCELL	Number of cells in RAP	16

2.6. Specification of A Conventional System

It is evident that an analysis based on the current INGRES implementation may not lead to general results, due to the particular constraints of INGRES. Therefore, we have assumed that the conventional system supports a new, high-performance INGRES. which supports precompilation of queries. By inspecting the INGRES code it appears that the CPU time per query would decrease by at least one half if queries were pre-compiled.

2.7. Host time

The host for all the backend machines, and the entire system for the conventional machine, is assumed to be a DEC PDP 11/70. The host system must perform two functions for each of the backend systems: query compilation and communication with the backend. The former is denoted as "overhead", the latter "communication time".

2.7.1. Host overhead

After the query is retrieved from the user's terminal, it is parsed and transformed into the appropriate form for the backend. The time required for this compilation is divided into two parts: OVCPU denotes the overhead CPU time, and OVIO denotes the overhead I/O time. In assigning numerical values to OVIO and OVCPU we assume that the queries are pre-compiled, and that terminal communication time is so small that it can be disregarded.

The usual strategy for pre-compiling queries is that the user defines a query prior to execution time; the data management system parses, validity checks, and sets up a run-time module for the query. When it is executed, information about what the query depends upon (e.g. structure of the relation) must be checked. The minimal I/O time for such checking is one disk access per relation. Therefore, OVIO = DAVAC = .030 seconds per relation. This assumes that validity checking is done entirely in the host and does not involve the backend.

OVCPU is the CPU time required to perform this validity checking. In UNIX it takes .006 seconds of CPU time to request a memory-resident page from the operating system and transfer it into a user's address space [RITC78]. In [HAWT79b] it is shown that INGRES requires approximately .16 seconds of CPU time for validity checking and setting up a minimal query. Clearly the time for the minimal value of OVCPU must fall between .006 seconds and .16 seconds. Where it falls between these two values depends on the functionality of the validity checking: the amount of optimization present or possible, the level of security, etc.

These two values shall be assigned to OVCPU as best case -- worst case assumptions. It must be pointed out that this is an absolute minimum; in general there is much more work to do. If the database has changed, and the query must be re-compiled, the extra work of creating the run-time module actually increases the overhead.

2.7.2. Host - to - Backend Communication Time

BCOM denotes the host CPU time to send a query to the backend system and receive the results. To assign a numerical value to BCOM, the following considerations must be taken into account. In UNIX, the measured inter-process communication time is .12 seconds for the 5 processes of INGRES to send messages to and receive messages from each other in a linear sequence [HAWT79b]. This is .03 seconds for a send-and-receive transmission per process pair. The time, in UNIX, for a process to request a page of data from the operating system, and to receive that page, is .006 seconds if the page is in memory. Whether the time to communicate between the backend system is closer to the inter-process time or the page-transfer time depends on the design and the functionality of the interface. Rather than attempting to further narrow this value, the .006 sec. time is assigned as the "best case" time, and the .03 sec. time as the "worst case" time to perform a single communication with the backend system.

2.7.3. Host data processing time

Each of the machines relies on the host to format the results for printing and move the results to the user's terminal. Some rely on the host to perform arithmetic functions. The amount of time required in the host is query and machine dependent. It is denoted by HDP, the host data processing time. The host parameters are summarized in Table 2.4.

Table 2.4 Host parameters and values

parameter	meaning	value best case --worst case
OVIO	host overhead I/O time	.0300 seconds
OVCPU	host overhead CPU time	(.006--.1600) sec.
BCOM	host CPU time to communicate with backend	(.006--.0300) sec.
HDP	host time to format results for printing, perform math functions, etc.	query and database machine dependent

3. Performance Comparisons

Each of the machines described is a cellular system: data is stored in cells, with a processor (in some cases dynamically assigned) per cell. Operations on the cells take place in parallel. While there are minor implementational differences between the machines, we have ignored them in this analysis since none of the machines is in volume production (see Section 2). The purpose of our evaluation is to determine the effect of the major design differences on machine performance. The following list highlights some of the interesting differences between the machines which we shall examine.

1) Query-processing algorithms implemented in the machines - Each

of the machines uses different algorithms for the join and project operators. For example, several designs depend on repeatedly scanning the database in order to perform a join operation.

2) The performance of caching database machines - RAP and DIRECT cannot process data directly on the disk but instead rely on a CCD cache which is loaded from mass storage.

3) The effect upon performance of the transfer of entire tuples to the host - The simpler database machines always must return the full tuple to the host.

In [HAWT79a] three classes of relational queries are identified: overhead-intensive, data-intensive, and multi-relational queries. In order to compare the performance of each database machine with each other and a "fast" version of INGRES, one query was chosen from each category. Query Q1 is an overhead intensive query; Q2 is a data-intensive multi-relation query (i.e. a join between two relations); and Q3 is a data intensive query on a single relation which includes an aggregate operation.

Each of the three queries are actual queries and were chosen to reflect the "average" query of that type as reported in [HAWT79a]. The database for the three queries is the UC Berkeley EECS Department's course and room scheduling database. This database contains 24704 pages of data in 102 relations. The data is information about courses taught: instructor's name, course name, room number, type of course, etc.

In the following section, the total work and response times are calculated for each query on each machine. The total work is the sum of the time spent in all components of the machine. The

response time is the sum of the component times that cannot be overlapped.

3.1. Q1 - Short queries

The query chosen for this analysis is:

```
query Q1:
        retrieve(QTRCOURSE.day, QTRCOURSE.hour)
           where QTRCOURSE.instructor = "despain, a.m."
```

QTRCOURSE contains 1110 tuples. Each tuple has 24 attributes, and is 127 bytes long. The relation can be stored as a heap in 274 pages. The attribute "day" is a character field, 7 bytes long; "hour" is also a character field, and is 14 bytes long. Three tuples satisfied this query.

The following facts are relevant to the performance comparisons for this query:

1) The relation fits on one cylinder because the number of pages is less than DCYL, the number of disk blocks per cylinder.

2) Since there were only three tuples returned to the host, the time to process the retrieved data and send it to the user terminal is assumed to be 0. Also, since the hit ratio is so low, it is assumed that there is not a problem with bus contention or controller processor speed.

3) There is one relation in the query, so the host overhead cpu time, OVCPU, is a constant factor, (.006--.160) seconds. OVIO, the host overhead I/O time, is .030 seconds.

3.1.1. fast INGRES

The work to execute this query in a "fast INGRES" system is:

FWORK = OVIO + OVCPU + DPIO + DPCPU

When this query is executed by INGRES [HAWT79b] there are eleven I/O references: three to the QTRCOURSE relation and eight to system relations. The references to QTRCOURSE are the data processing references. QTRCOURSE was hashed on instructor name. The worst case page reference time occurs when the three pages reside on separate cylinders. In this case there is one access per reference. The best case condition arises if they are on the same track in which case the reference time is one access and 3 block transfers. Thus DPIO = (.0336 best case -- .090 worst case).

For standard INGRES .12 seconds of CPU time are required to process the query. Since the "fast INGRES" system is compiled (and not interpreted), the data processing CPU time will be half of the measured interpreted query. Therefore DPCPU = .06 seconds.

The total work that a query incurs is a measure of tne impact of the query on system performance. The total work for this query is:

```
FWORK = OVIO + OVCPU + DPIO + DPCPU
      = (.13 --.34) seconds
```

Clearly the overhead CPU and I/O times must be done serially since the one I/O reference contains information necessary for the validation of the query. However, the data processing I/O and CPU times can be overlapped. The response time is therefore:

FRES = OVIO + OVCPU + max [DPIO, DPCPU]
= (.096 -- .280) seconds

The best case is dominated by the CPU time to process the 3 pages which at .06 seconds is almost half of the total best case work and almost two-thirds of the best case response time. The overhead CPU, at .16 seconds worst case, dominates the worst case work and response times.

3.1.2. Associative Disks and CAFS

For this query the associative disks and CAFS have the same functionality. The time required for either system is:

AWORK = OVCPU + OVIO + BCOM + DAVAC + n*DROT

There must be one disk access (DAVAC) to position the machine at the right cylinder. Then, since the data resides on a single cylinder, and since the cell processors operate at the rotational speed of the disk, the data processing will take one revolution of the disk. Thus, n = 1. Then, AWORK = (.089--.267) seconds. The response time is the same.

The major components of the best case time are the two disk access times: one to perform the overhead functions on the host and the other to position the disk arm for data processing. The worst case time is dominated by the host CPU overhead.

3.1.3. CASSM

First the storage requirements for the QTRCOURSE relation on CASSM must be determined. There are 1110 tuples in the relation. Assuming a two-byte encoding of the relation name, 2220 bytes will be required to store the per-tuple relation name. There are

25 attributes per tuple. Assuming each attribute name can be encoded in one 1 byte, then since the name is stored for each attribute of each tuple 27,750 bytes are necessary to identify attributes. Each tuple is 127 bytes long; however, in CASSM no character fields over 4 characters long are directly stored. They are stored once, then pointers to the correct character string are stored in each tuple. Using that method for storing the relation requires 74 bytes per tuple, which includes 9 pointers to character strings. The 9 attributes that are represented as character strings were measured to require 6417 bytes of storage. So the total storage required is 221 blocks, validating Langdon's conjecture that the extra storage required for the delimiters in CASSM is offest by the data encoding algorithm it uses [LANG78]. This data will fit on one cylinder.

The work required to perform this query is:

CWORK = OVCPU + OVIO + BCOM + DAVAC + n*DROT

The number of rotations, n, is:

1 rotation to mark all tuples
for the relation QTRCOURSE
1 rotation to find the pointer to
character string "despain,a.m."
1 rotation to mark all tuples with
the pointer to "despain"
1 rotation to return all "day"
attributes in marked tuples
1 rotation to return all "hour"
attributes in marked tuples

5

It is assumed that CASSM sends the attributes to the host in their coded form, along with the necessary decoding information. Since the query only produces 3 tuples, the host CPU time

required for decoding can be disregarded. The value for n for this query is therefore 5, and the work to process the query is: CWORK = (.156--.334) seconds. The response time, CRES, is the same.

The best case time is dominated by the five rotations. The major component in the worst case is the same as in the previous systems, the host overhead CPU time.

3.1.4. DBC

The work required in the DBC for this query is:

DBWORK = OVCPU + OVIO + BCOM + DCYL + DAVAC + DROT

DBWORK differs from the associative disk case only in the time required to perform the DBC cylinder select, DCYL. Although a significant part of the DBC design effort has been to make index selection and manipulation very fast, a time must be associated with cylinder selection. If it is assumed that the indices reside in RAM, then the time to fetch and manipulate them is probably on the order of the time to fetch a memory-resident page in UNIX, .006 seconds. For this analysis it is assumed that DCYL = .006 seconds. Then, DBWORK = (.095--.273) seconds. Since the cylinder selection must be done before the data processing can begin, it cannot be overlapped. DBRES, the response time, is therefore the same as DBWORK.

3.1.5. DIRECT

The work DIRECT must perform is:

DWORK = OVCPU + OVIO + BCOM + DPIO + n*CSCAN

CSCAN is the CCD page scan time; DPIO is the time to read

QTRCOURSE from the disk.

The QTRCOURSE relation is 274 512-byte blocks long so it will fit in 9 of DIRECT's 16K byte data cells. In this query the cell processors will read from one data cell and write the requested attributes of the qualified tuples to another cell, thus forming a temporary relation. Assuming no other queries are running, all 16 cells are available to hold the relation and thus the entire relation can be loaded from mass storage at once. Therefore, DPIO is one disk seek plus the transfer time. Of course the best case I/O time is when the relation is already in DIRECT's buffers; in that case the time is zero.

```
DPIO = (0 -- (DAVAC + 274*DREAD))
     = (0 -- .249) seconds
```

Since 9 data cells are required to store this relation, and there are only 8 cell processors, two or three scan times are required to process this query. During the first scan each cell processor will read and process one data page. Qualifying tuples are placed in the processor's internal buffer which is written out to a CCD data cell only when it is either full or when all the input data pages have been examined. During the second scan, seven of the processors will each write their internal buffer into a new CCD page if their internal buffer contained any qualifying tuples. Also during this second scan, the eighth processor will read and examine the 9th data page. If after examining this page, the processor has found any qualifying tuples, a third scan will be required to write its internal buffer to a CCD data cell. Since only 3 tuples satisfy this query, at most 3 processors will

produce a new page. Thus in the best case 2 scans are required and in the worst case 3.

The total work is:

DWORK = OVCPU + OVIO + BCOM + (2*CSCAN--3*CSCAN) + DPIO
= (.066 -- .505)

In the best case, the response time is the same as the total work, because no overlapping is possible. If the pages must be read from secondary storage, as in the worst case, the data transfer time for the blocks entering the last data cell can be overlapped with the processing of the first 8. Since there are thirty-two 512-byte blocks per data cell, the response time is:

DRES = OVCPU + OVIO + BCOM + (1*CSCAN--2*CSCAN) +
(CSCAN -- (DAVAC + 242*DREAD + max(CSCAN,32*DREAD)))
= (.066 -- .490)

For both times, the best case is dominated by the scan times for the CCD data cells, and the worst case by the data transfer time from the disk.

3.1.6. RAP

For RAP the execution time of this query is:

RWORK = OVCPU + OVIO + BCOM + DPIO + n*CSCAN

QTRCOURSE, because it is 9 cells long, will completely fit in the sixteen RAP processor cells. If QTRCOURSE is already in the cache when the query begins, then DPIO is equal to 0. Otherwise, DPIO is the same as in DIRECT. Thus DPIO = (0 -- .249) seconds.

The number of cell scans, n, that must be performed for data processing is:

1 to mark qualifying tuples in 9 processor cells
1 to send marked tuples to host
--
2

Thus, RWORK = OVCPU + OVIO + BCOM + 2*CSCAN + DPIO
= (.066 -- .493) seconds

The response time is the same.

3.1.7. Conclusion

While we have attempted to adjust the various system parameters (e.g. number of processors) in order to make the comparisons fair, one expects that a more complex system, where that complexity consists of added buffers, processors, etc., should result in faster response time and less system work. In Figure 3.1, the system work for each of the systems is plotted. (We have not plotted the response times for each system since they are generally the same as the system work.) The systems are ordered along the horizontal axis on the basis of "increasing complexity":

1) INGRES, the "basic" system
2) Associative Disks, CAFS - the cell processors are relatively simple
3) CASSM - more intelligence in the cell and controlling processors
4) RAP - very fast cell processors; CCD cells
5) DBC - highly functional controlling processor: structure memory, index translation unit, etc.
6) DIRECT - fast, general purpose cell processors, cross-point switch

This curve indicates that that the increased complexity of the machines does not result in decreased system work or faster response times. While RAP and DIRECT exhibit the fastest best-case times, they also have the slowest worst-case times - in

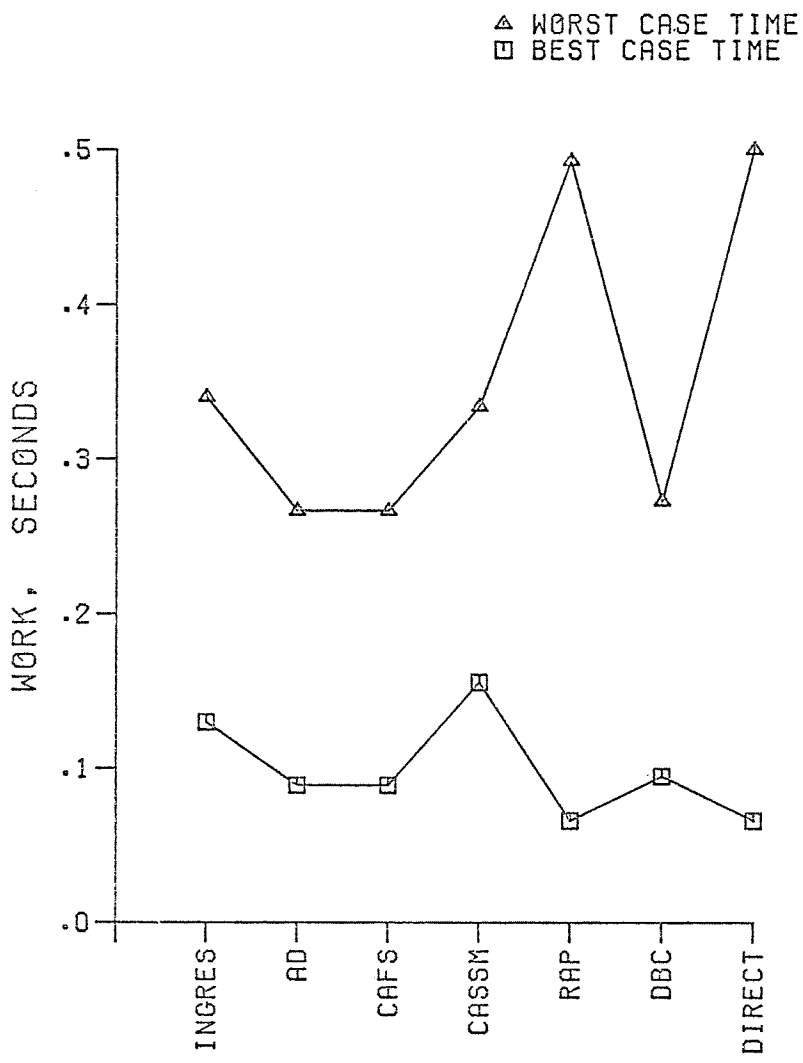

Figure 3.1
Query Q1

which case they are even slower than INGRES. This situation occurs because INGRES maintains QTRCOURSE as a hashed relation, and only reads three pages of it. The caching systems (i.e. RAP and DIRECT) had to serially read the entire relation. This seems to indicate that database machines which rely on caching do not perform satisfactorily on simple overhead-intensive, queries. When compared with the performance of "fast INGRES" it is apparent that, for this query, the increased cost and complexity of the database machines do not result in a significant increase of performance.

3.2. Multi-relation Queries

The second query chosen for our evaluation is the following multi-relation query, Q2.

```
retrieve ( ROOMS.building, ROOMS.roomnum, ROOMS.capacity,
                COURSE.day, COURSE.hour)
        where ROOMS.roomnum = COURSE.roomnum and
        ROOMS.building = COURSE.building and ROOMS.type = "lab"
```

The relation "COURSE" contains information about all the courses taught by the UC Berkeley EECS Department in the last four years. It contains 11436 tuples in 2858 pages, and is stored in an ISAM storage structure, keyed on instructor name and course number. "COURSE" requires 130 tracks (7 cylinders) of disk space. The relation "ROOMS" contains information about every room that the EECS Department can use for teaching courses. It contains 282 tuples in 29 pages, and is hashed on room number. "ROOMS", since it is stored in 29 pages, can be stored on 2 tracks of one cylinder. The result of this query is a list which contains the building, room number, capacity, day, and hour of the use of any

lab for the last four years.

Since the query is a two-relation query, we assumed that the overhead to for this query is double that of the single-relation queries. Thus, OVCPU, the overhead CPU time, is (.Ø12 -- .32) seconds. The overhead I/O is .Ø6 seconds.

3.2.1. fast INGRES

It is assumed that the "fast INGRES" system uses the following algorithm for this query:

1) Form R2, the sorted, restricted projection of "ROOMS", retaining R2 in an in-memory array. R2 contains 22 tuples in 2 pages, and can easily be kept in memory.

2) Scan the "COURSE" relation, and compare the join fields of each tuple in COURSE to each tuple in R2.

3) Output the required attributes of any matching tuples.

Using this algorithm, the total work for fast INGRES is:

FWORK = OVCPU + OVIO + RSORT + CCOMP + CDP + DPIO

RSORT, the time to perform the sorted, restricted projection of ROOMS, will be calculated first. The work that must be performed to create R2 is:

1) Read the entire ROOMS relation, testing each tuple to determine if ROOMS.type = "lab"

2) For those tuples that are the correct type, write the building, roomnum, and capacity attributes to an in-memory array (R2)

3) Sort R2 on "roomnum, building" and remove duplicates.

The CPU time to perform the above is composed of the time to sort R2 (which was measured to be .Ø6 seconds on the 11/7Ø) and the

time to read ROOMS, which is equal to .ØØ6 * 29 = .174 seconds. Therefore, RSORT = .23 seconds.

CCOMP, the time to compare the (roomnum, building) attributes in COURSE to those in R2, will be determined next. There are 22 tuples in R2, and 11436 in COURSE. Since R2 is sorted, each of the COURSE tuples need be compared only until a "greater than" result is found. The (roomnum, building) attribute pair is 2Ø characters long. If for each COURSE tuple, on the average one half of the R2 tuples (11 tuples) are examined then the average number of comparisons that must be made per tuple in COURSE is 2Ø * 11, or 22Ø. Each character comparison requires at least 3 instructions: the comparison, the test for condition, and the loop control. Therefore, there are a minimum of 66Ø instructions for each of the 11436 tuples in COURSE. This is about 7.5 million instructions. Since an 11/7Ø is a 1 MIP machine, 7.5 seconds will be required. This is a minimum time. It does not include the time to split the tuple into attributes, do type conversion, etc. The worst case time, however, should be no more than 15.Ø seconds.

CDP, the time CPU time required to read the data, is calculated next. There are a total of 2887 pages read; the UNIX CPU time required to read a page is .ØØ6 seconds. Therefore CDP = 17.32 seconds.

DPIO, the I/O time to read the data into memory consists of the I/O time to read the ROOMS (RIO) and COURSE (CIO) relations. Since the ROOMS relation is on two tracks of the same cylinder, the time to read it is RIO = DAVAC + DROT/2 + 29 * DRATE where

DROT/2 = the latency required to access the second track. The I/O time to read the COURSE relation is:

CIO = DAVAC + 6*CYAVAC + (13Ø - 7) * DROT/2 + 2858*DRATE

DAVAC is the average access time to begin reading the first cylinder. CYAVAC is the time to move the disk head from one cylinder to an adjacent one. For AMPEX 92ØØ disk drives, that time is .ØlØ seconds. COURSE is stored in 13Ø tracks; the time to access the first track on each cylinder is included in the terms DAVAC and 6*CYAVAC; the time to access the remaining tracks is the average latency, DROT/2. Finally, the time to transfer the data to memory is 2858 pages * DRATE.

Therefore the total DPIO time is:

```
DPIO = DAVAC + ROT/2 + 29 * DRATE
     + DAVAC + 6*CYAVAC + (13Ø - 7) * DROT/2 + 2858*DRATE
     = 4.62 seconds
```

The total work to process this query is:

```
FWORK = OVCPU + OVIO + RSORT + CCOMP + CDP + DPIO
      = (29.74 -- 37.55) seconds
```

Since the data processing I/O can be overlapped with CDP, the response time is: FRES = (25.12 -- 32.93) seconds. Both times are dominated by the operating system overhead associated with reading the pages (CDP).

3.2.2. Associative Disks

It shall assumed that the following algorithm is used in processing query Q2 in an associative disk system:

1) The sorted, restricted projection of ROOMS (R2) is formed in the host

2) The host issues 22 queries, one for each tuple in R2, to

retrieve from the associative disk the tuples in "COURSE" with matching roomnum and building attributes as those in the query. These queries shall be denoted as the 22 sub-queries, and are of the form

```
retrieve (COURSE.day, COURSE.hour)
where COURSE.roomnum = value1 and COURSE.building = value2
```

As matching tuples are returned, they are printed on the users's terminal.

The time to perform this query using the above algorithm is:

AWORK = OVCPU + OVIO + RSORT + (n'+1)*BCOM + RPROC + n' * CPROC

RSORT, the CPU time to perform the sorted, restricted projection of the ROOMS relation in the host, is the same as for fast INGRES. The number of communications required between the host and the associative disk (BCOM) is the number of subqueries (n' = 22) plus the one query to form the sorted, restricted projection.

RPROC, the I/O time to process the restricted projection of ROOMS, is: RPROC = DAVAC + DROT. This is one average access time, plus one disk rotation time to perform the query (since the ROOMS relation resides on one cylinder).

The time to process each of the 22 subqueries, CPROC, is: CPROC = DAVAC + 6*CYAVAC + 7*DROT which is the time to access each of the seven cylinders on which the COURSE relation resides and to perform the subquery on each.

Therefore the total time is:

```
AWORK = OVCPU + OVIO + RSORT + (n'+1)*BCOM + RPROC + n'*CPROC
      = (5.Ø4 -- 5.9Ø) seconds
```

The response time is the same. The execution time of the 22

subqueries (4.55 seconds) dominates both the best and worst case times.

3.2.3. DBC

The key question in determining the performance of the DBC on this query is whether the search space in the 7 cylinders used to store the "COURSE" relation can be narrowed to one or two cylinders by the use of attribute indices. The "COURSE" relation was clustered on course number, which is not mentioned in this query. Therefore, even if indices existed for each of the "COURSE" attributes in this query, they would not narrow the search space because, in the absence of any strong correlation between course number and the other attributes, it is to be expected that qualifying tuples will occur on each of the 7 cylinders.

DBC handles single-relation queries only, and in [BAN78b] it was shown how joins are split into single-relation queries. Using the queries in [BAN78b] as a guide, we find that DBC will process query Q2 in exactly the same manner as the associative disk. Therefore, its execution time is the same as the associative disk.

3.2.4. CASSM

The algorithm which CASSM uses to implement joins is based on two bit-maps, one for each of the relations to be joined. First, a pass is made over the smaller relation (ROOMS), and the join attributes (roomnum,building) of qualifying tuples are used to hash to a bit map to mark the presence of a value. Next a

pass is made over the second relation (COURSE) and a second bit map is marked if the (roomnum,building) attribute pair hash to a location that is marked in the first bit map. Also if both bit-maps are marked, the tuple is itself marked for collection. Finally a second pass is made over the first relation, and those tuples whose (roomnum,building) pairs hash to marked locations in both bit-maps are marked for output. The marked attributes from both relations are sent to the host processor to perform the actual join. This final step is required since the hash functions are most likely not perfect.

The time to execute this query in CASSM is:

CWORK = OVCPU + OVIO + BCOM + HJOIN + n*RPROC + m*CPROC

HJOIN is the amount of CPU time required in the host to perform the actual join. RPROC is the time required to scan the "ROOMS" relation and CPROC is the time to scan the COURSE relation. HJOIN will be computed first.

Using the above algorithm, 22 tuples of "ROOMS" and 422 tuples of "COURSE" are marked in both bit maps. Only the necessary attributes from those tuples are sent to the host. Those attributes require a total of 47 pages of storage. Because there are a small number of pages the host can complete the processing of this query by first sorting in memory the qualified tuples of the two relations and then merging the two sorted lists to match the correct tuples. The CPU time to sort the 47 pages is 14.1 seconds on the 11/70; therefore HJOIN = 14.1 seconds.

The number of scans of the rooms relation (n) is:

```
1       scan to mark bit map
1       scan to check COURSE's bit map and mark tuple
4       scans to return 4 attributes
--
6       n, total scans required of the ROOMS relation
```

The number of scans of the COURSE relation (m) is:

```
1       scan to mark map and mark qualifying tuples
4       scans to send attributes to host
--
5       m, the required number of scans of COURSE
```

The total time to do this query is, therefore:

```
CWORK = OVCPU + OVIO + BCOM + HJOIN + n*RPROC + m*CPROC
      = (15.50 -- 15.83) seconds
```

The best and worst case times are dominated by the time to perform the join in the host. The response time is the same.

3.2.5. DIRECT

The DIRECT join algorithm works by joining each of n units in one relation (the outer relation) with all of the units in the other relation (the inner relation). If we consider a unit to be a page and there are n processors available, then each processor can join one page of the outer relation with the entire inner relation. (The granularity of a unit could be as fine as a tuple).

For this query the COURSE relation will be the outer relation (in general, the outer relation is the larger of the two relations) and the ROOMS relation will be the inner relation. Each of the eight cell processors will first read the ROOMS relation, apply the restriction ROOMS.type = "lab", project over the desired attributes, and then sort the resulting 22 tuples. Each

cell processor will keep these 22 tuples in an internal buffer for the duration of the query. Then each cell processor will join a subset of the COURSE relation with its own copy of the ROOMS relation using the same technique employed by "fast INGRES".

Using this algorithm, the work required is:

```
DWORK = OVCPU + OVIO + BCOM + r*CSCAN + DSORT
        + c*CSCAN + DCOMP + DPIO
```

The parameter r is the number of CCD page scans required by the cell processors to read the ROOMS relation. The time required by the cell processors to form the sorted, restricted projection of ROOMS is DSORT. The parameter c is the number of scans of the COURSE relation required to execute the join. DCOMP is the CPU time required by the cell processors to compare the (roomnum, building) attributes in their subset of COURSE with the sorted, restricted projection of ROOMS. DPIO is the I/O time required to transfer the ROOMS and COURSE relations from mass storage to the CCD cells.

Since the entire "ROOMS" relation will fit on one CCD cell, only one scan is required by the cell processors to read the relation. Therefore, r equals 1. Forming the sorted, restricted projection of the ROOMS relation takes .06 seconds on an 11/70. Since the DIRECT cell processors are assumed to also be 1 MIP processors, DSORT = .06 seconds.

Next, c will be calculated. The 2858 pages of "COURSE" correspond to ninety 16K byte pages. Eight processors, each working in parallel, will require 12 scans of the COURSE relation to

complete the join. Six cell processors will join their copy of tne 22 tuples from the ROOMS relation with eleven 16K pages of the COURSE relation; two cell processors will examine twelve pages of the COURSE relation. Each cell processor will process a distinct subset of the COURSE relation. One additional scan is required for each cell processor to write its output buffer to a CCD cell. Therefore, c = 13.

DCOMP is the CPU time required by the cell processors to compare the (roomnum, building) attributes in their subset of COURSE with the sorted, restricted projection of ROOMS. Since two cell processors must examine twelve pages of the COURSE relation, their execution time will be the limiting factor. For "fast INGRES" it was determined that, on the average, 220 comparisons must be made for each COURSE tuple examined and that each comparison requires 3 instructions. The two cell processors which examine twelve pages of COURSE will each examine 1525 tuples. This will require 1 million instructions at approximately 1 microsecond/instruction. Thus DCOMP equals 1 second.

DPIO is composed of the time to read the ROOMS (RIO) and COURSE (CIO) relations into CCD memory. If the ROOMS relation is in CCD memory then RIO = 0; otherwise, RIO = DAVAC + 29 * DREAD. Thus, RIO = (0 -- DAVAC + 29 * DREAD).

Because the entire COURSE relation will not fit within the CCD cache the minimum value of CIO will occur when the first fifteen 16K byte pages are already loaded (along with the ROOMS relation): CIO = DAVAC + 5 * CYAVAC + 2378 * DREAD. If none of the COURSE relation is initially loaded, then CIO = DAVAC +

6*CYAVAC + 2858*DREAD. Thus:

CIO = (DAVAC+5*CYAVAC+2378*DREAD -- DAVAC+6*CYAVAC+2858*DREAD)

This formula assumes that the average access time must be used for only the initial access to the relation; the remaining scans will be for adjacent cylinders.

The total time is therefore:

DWORK = OVCPU + OVIO + BCOM + r*CSCANS + DSORT
+ c*CSCANS + DCOMP + DPIO
= (3.29 -- 4.07) seconds

For the best case 11/12th of the DCOMP time can be overlapped with the I/O time. For the worst case only 5/6 of it can be overlapped. Thus the response time is:

DRES = (2.37 -- 3.24) seconds

It should be noted that DIRECT could achieve approximately the same level of performance on this query if it were constructed from sixteen 1/2 MIP processors instead of eight 1 MIP processors. Furthermore, the number of cell processors which could be effectively utilized could be increased to 90. However, doing so would decrease both the best and worst case execution times by at most one second (DCOMP equals 0) because the execution time for DIRECT is dominated by the time to load the CCD cache.

3.2.6. RAP

It is assumed that the algorithm used by RAP for this query is the same as the one which is used by the associative disk system. That is:

1) Form sorted, restricted projection of "ROOMS", holding it in

memory.

2) Perform 22 sub-queries

Since the RAP backend system does not include a general-purpose computer, the sorted, restricted projection of ROOMS must be performed in the host, which subsequently issues the 22 sub-queries. The time required by RAP is:

```
RWORK = OVCPU + OVIO + RSORT + (1+n*c)*BCOM +
                r*CSCAN + n*(c * CSCAN) + DPIO
```

RSORT, the time to perform the sorted, restricted projection, was calculated for "fast INGRES" to be .23 seconds.

The number of cell scans to produce the restricted projection of ROOMS (r) is:

```
  1   scan to mark tuple which satisfy the ROOMS restriction
  1   scan to return marked tuples to the host
 ---
  2   number of scans for ROOMS
```

The number of scans of the COURSE relation (n*c) is calculated next. Because the CCD cache is not large enough to hold the entire COURSE relation (90 pages), this query will have to be processed as six separate phases (n=6). Each phase will begin by first reading the next 16 pages into the cache and then applying the 22 subqueries. For each subquery one scan is required to mark the qualifying tuples and one is needed to return the marked tuples to the host. Thus each phase will require c = 2 * 22 = 44 scans.

DPIO is the same is for DIRECT. The total work is therefore:


```
RWORK = OVCPU + OVIO + RSORT + (1+n*c)*BCOM +
                r*CSCAN + n*(c * SCAN) + DPIO
      = (7.06 -- 14.18) seconds
```

Because the scanning of a cell in RAP cannot be overlapped with the loading of the data cells the response time is the same.

3.2.7. CAFS

CAFS handles joins in the same way as CASSM except that it does not mark qualified tuples, but passes them on to the host. To review the algorithm: There are two bit-maps, one for each of the relations to be joined.

1) First, a pass is made over the smaller relation (ROOMS), and the join attributes (roomnum,building) of qualifying tuples used to hash to a bit map to mark the presence of a value.

2) A pass is made over the second relation (COURSE) and a second bit map is marked if the (roomnum,building) attribute pair hash to a location that is marked in the first bit map. Also, if both bit-maps are marked the tuple is itself sent to the host.

3) A second pass is made over the first relation, and those tuples whose (roomnum,building) pairs hash to marked locations in both bit-maps are sent to the host.

The host must perform the actual join on the smaller relations passed to it.

The time to process query Q2 in CAFS is therefore:

CAWORK = OVCPU + OVIO + BCOM + HJOIN + 2*RPROC + CPROC

The time to perform the join in the host, HJOIN, was discussed in the section on CASSM, and found to be 14.1 seconds.

RPROC, the time to scan the ROOMS relation and mark the

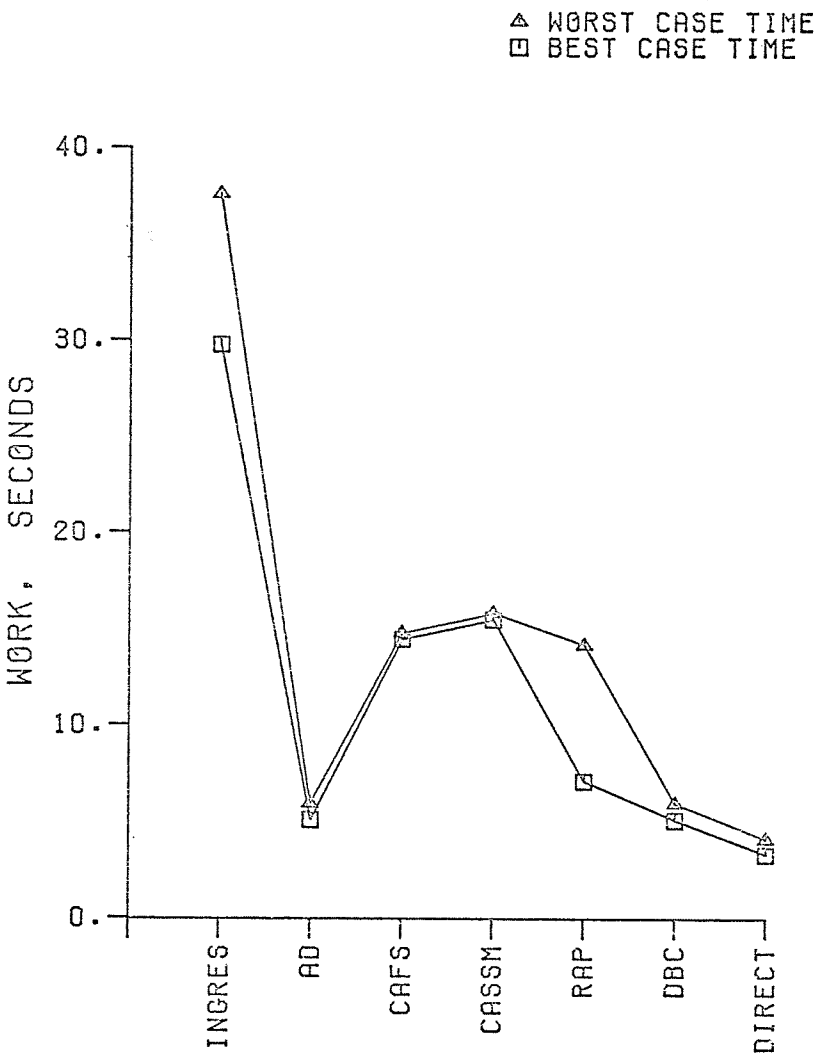

Figure 3.2
Query Q2

first bit map, is RPROC = DAVAC + DROT. CRPOC, the time to scan the COURSE relation, is:

CPROC = DAVAC + 6*CYAVAC + 7*DROT

The total time is therefore: CAWORK = (14.48 -- 14.81) seconds. The response time is the same.

3.2.8. Conclusion

The results of this query are plotted in Figure 3.2. The best performance is demonstrated by DIRECT. The associative disk and DBC systems also perform well because they do not rely significantly on the host. The CAFS and CASSM systems are handicapped by the fact that they must perform the join in the host.

The performance of RAP is very sensitive to whether the COURSE relation can be completely stored in the ccd data cells. In order to execute the join of the ROOMS and COURSE relations, RAP, like the associative disk and DBC systems, depends on rescanning the COURSE relation for each tuple of the ROOMS relation. While this approach limits the performance of the associative disk and DBC systems on join operations, it has a very significant impact on the performance of RAP when the size of the COURSE relation requires that the query be repeated for each section of the COURSE relation.

3.3. Aggregate functions

An aggregate function is applied to a relation by first partitioning the relation based on the value of an attribute. Then a function is applied to the tuples in each partition. The result is one value for each partition. The following query was chosen

for the aggregate function query:

```
Q3: retrieve (GMASTER.acct, GMASTER.fund,
    encumb = sum (GMASTER.encumb by GMASTER.acct, GMASTER.fund))
```

Relation GMASTER contains 194 tuples, 2 tuples per page. and resides on a single cylinder. There are 17 unique values for the (acct,fund) pair. The query returns to the user the 17 unique (acct,fund) pairs along with their associated sums.

The algorithm employed by INGRES for performing this query is to read the GMASTER relation once and retain in memory only the (encumb,acct,fund) attributes from each tuple. These are first stored in a temporary storage area and then sorted on (acct, fund). The sum is computed for each partition, and the results are printed.

Since DIRECT utilizes simple, but general purpose processors as cell processors, each cell processor can execute an algorithm similar to the one employed by INGRES. The entire query will be given to each cell processor which will perform the query on a subset of the GMASTER relation and report its partial results to the back-end controller. After receiving the partial results from each participating cell processor, the controller will compute the final values for the 17 unique (acct, fund) pairs and return the results to the host for printing.

Since associative disks, CAFS, DBC, and RAP do not directly support aggregate functions, either a host processor or back-end controller must be used. The algorithm used by each of these machines is to first retrieve all the (acct,fund) pairs from the GMASTER relation and sort them to remove duplicates. This

requires one back-end communication and one pass over the data. Then the host (or back-end controller) will issue the following query for each of the 17 unique values found:

Retrieve (GMASTER.encumb) where acct=value1 and fund=value2

The host processor must then accumulate the sum and print it with the values of the acct and fund attributes on the user's terminal. This algorithm requires one back-end communication and one pass over the GMASTER relation for each of the 17 uniques values.

Since CASSM's cell processors are capable of summing attribute values, the host processor does not need to be used. The algorithm used is to find the first value of acct and set one of its mark bits to show it is participating in this sum and then mark every tuple that contains a pointer to that value of acct. This requires one pass over the data. During the next pass over the data, the first value of fund for the marked tuples is found and another mark bit is set. Then the CASSM cell processors will make another pass over the data in which the encumb attribute from each doubly marked tuple is summed. This algorithm continues by repeatedly finding the next unmarked value of the fund field for this value of acct. When all qualifying values of fund have been found, the algorithm is repeated until there are no more values of acct. In this particular query, the results are skewed because there was only one value for acct and 17 values for fund. Therefore, the algorithm takes 1 pass to mark all tuples for the one acct and then 34 passes for the 17 unique values of the fund field (2 passes for each fund value).

Using the algorithms described above and similar techniques

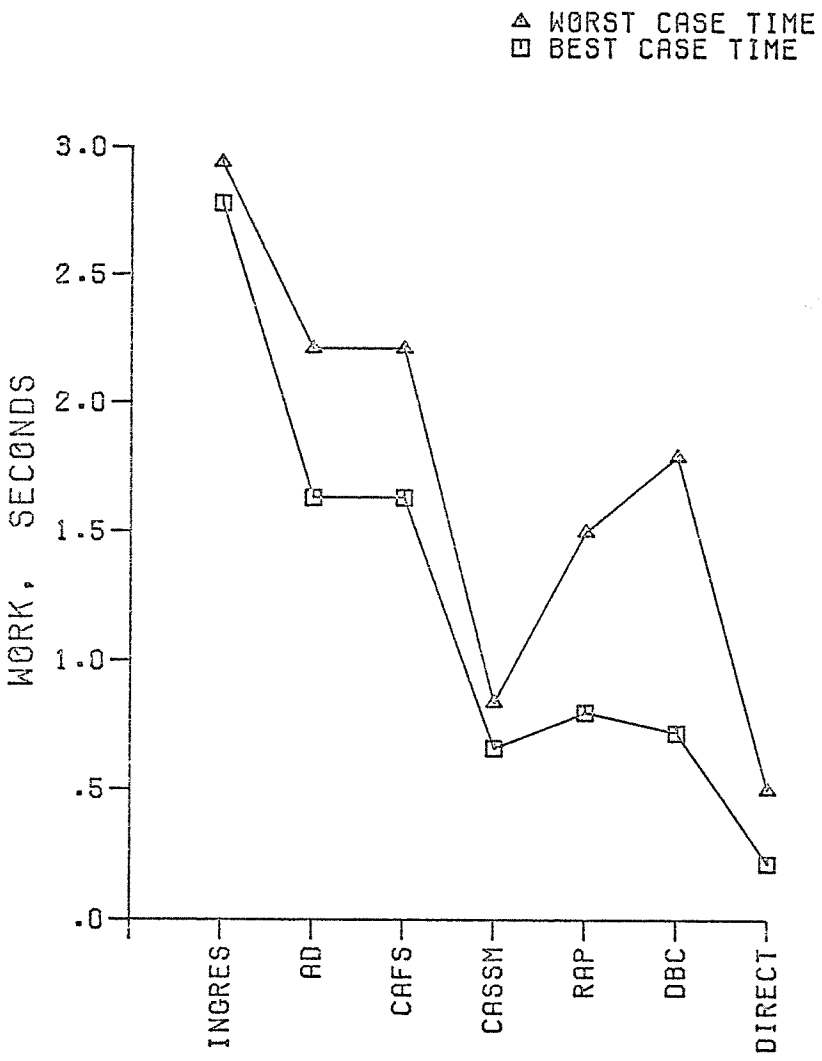

Figure 3.3
Query Q3

to those employed in Sections 3.1 and 3.2, the performance of each database machine was evaluated for this query. The results are displayed in Figure 3.3. From examination of this graph it is apparent that the CASSM and DIRECT machines give the best performance enhancement over a standard "fast INGRES" system. Even in the worst-case (when the entire GMASTER relation must first be loaded into the cache) DIRECT is the fastest on this query. Notice, however that RAP does not show the same degree of performance enchancement for this query. This seems to indicate that caching, when coupled with general purpose cell processors, works satisfactorily for this type of query. While DIRECT shows almost an order of magnitude improvement over INGRES on this query, that may not be too surprising since DIRECT has eight 1 MIP processors working on the query instead of just one. The results do indicate that parallelism can be successfully exploited to enhance query performance. The DBC also shows a significant degree of improvement. The associate disks, however, show almost no improvement for this query. This seems to be a consequence of the fact that the cell processors of these systems are not capable of summing attribute values and thus the host must handle each tuple twice.

4. Conclusion

The purpose of this paper has been to gain insight into the possibility of the use of database machines and attempt a meaningful comparison between the different database machines which have been proposed.

Two query types previously defined are data-intensive and overhead-intensive queries. It has been shown [HAWT79a] that database machines are not cost-effective if the application supported is mainly overhead-intensive queries.

In this paper it was shown that data-intensive queries can be performed very efficiently on database machines if the function performed on the data is a function the database machine provides. For instance, if the function is a simple test for equality, the database machine can perform the query entirely in the backend system, thus causing a gain in the system's performance. However, if the queries are such that the function on the data is one that the database machine does not provide, as in the function of printing the data in query Q1, the host processor is heavily impacted and the database machine causes little gain in the system's performance.

The performance of DIRECT on queries Q2 and Q3 indicates that a database machine which employs a fast disk cache and general purpose query processors performs very well on non-trivial queries. The relatively poor performance of RAP on all three queries indicates that a caching database machine which relies on simple cell processors which must repeatedly scan the cache to perform the query is not a good design.

The poor performance of both DIRECT and RAP on query Q1 indicates that caching database machines are not a good design if the majority of transactions expected are simple retrievals from a single relation. For this case the associative disk and DBC systems are the best design. However, none of the proposed

designs showed a significant performance improvement over the "fast INGRES" system. This seems to imply that if a database is structured so that the majority of simple queries can be processed quickly through the use of secondary indices, then the benefits of a database machine may not be significant. However, for multi-relation queries such as Q2 (for which INGRES employed a sophisticated join algorithm), the benefits of a database machine are very clear. One conclusion of this investigation is that more research is needed to marry the "on-the-disk-processing" of the associative disk, CASSM, and DBC systems for processing simple queries with an organization such as DIRECT for processing complex and multi-relation queries.

One point which has not been covered in this paper is the effect of the SIMD (Single instruction stream, multiple data stream) nature of all the database machines other than DIRECT. In [BORA79a], it is demonstrated that when a SIMD version of DIRECT is compared against a MIMD (Multiple instruction stream, multiple data stream) version of DIRECT, the MIMD version of DIRECT outperforms the SIMD version by approximately a factor of four. The MIMD version of DIRECT permits several user queries to execute simultaneously. The significance of this is illustrated by query Q2 in which only three of DIRECT's cell processors were used to execute the query. In DIRECT the other five cell processors would at the same time be executing another query thus improving overall system throughput.

5. Acknowledgements

The authors would like to thank Michael Stonebraker for his contributions to this paper.

Appendix A. REFERENCES

[BABB79] Babb,E. "Implementing a Relational Database by Means of Specialized Hardware," ACM Transactions on Database Systems, Vol. 4, No. 1, March 1979, pp. 1-29.

[BAN78a] Banerjee, J., and D.K. Hsiao, "Performance Study of a Database Machine in Supporting Relational Databases," Proc. Fourth International Conf. on VLDB.

[BAN78b] Banerjee, J., and D.K. Hsiao, "The Use of a 'Non-Relational' Database Machine in Supporting Relational Databases," Proc. Fourth Workshop on Computer Architecture for Non-numeric Processing, Syracuse, Aug. 1978.

[BAUM76] Baum, Richard I., David K. Hsiao, and Krishnamurthi Kannan, "The Architecture of a Database Computer - Part I: Concepts and Capabilities," Technical Report OSU--CISRC-TR-76-1, Computer and Information Science Research Center, The Ohio State University, Columbus, Ohio (National Technical Information Service Number AD-A034 154)

[BORA79a] Boral, H., and D.J. DeWitt, "Processor Allocation Strategies for Multiprocessor Database Machines," To appear: ACM Transactions on Database Systems. Also Computer Sciences Technical Report No. 368, University of Wisconsin, October 1979.

[BORAL79b] Boral, H. and D.J. DeWitt, "Design Considerations for Data-flow Database Machines," To appear: ACM-SIGMOD 1980 International Conference of Management of Data, also Computer Sciences Technical Report No. 369, University of Wisconsin, September 1979.

[COUL72] Coulouris, G.F., Evans, J.M., and R.W. Mitchell, "Towards content addressing in databases," Computer Journal, Vol. 15, No. 2, 1972, pp 95-98.

[DEWI78] Dewitt, D. J., "DIRECT - A Multiprocessor Organization for Supporting Relational Data Base Management Systems," Proc. Fifth Annual Symposium on Computer Architecture, 1978.

[DEWI79a] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Transactions on Computers, June 1979, pp. 395-406.

[DEWI79b] DeWitt, D.J., "Query Execution in DIRECT", Proceedings of the ACM-SIGMOD 1979 International Conference of Management of Data, May 1979, pp 13-22.

[HAWT79a] Hawthorn, Paula and M. Stonebraker, "Performance Analysis of a Relational Database Management System," Proceedings of the ACM-SIGMOD 1979 International Conference

of Management of Data, May 1979, pp 1-12.

[HAWT79b] Hawthorn, Paula, "Evaluation and Enhancement of the Performance of Relational Database Management Systems," Electronics Research Laboratory, University of California at Berkeley, Memo No. M79-70.

[HSIA79a] Hsiao, David K., "Data Base Machines Are Coming!", IEEE Computer, March 1979, pages 7-10.

[HSIA79b] Hsiao, D.K., and J. Memon, "The Post Processing Functions of a Database Computer," Computer and Information Science Technical Report OSU-CISRC-TR-79-6, Ohio State University, July 1979.

[HSI76a] Hsiao, David K. and Krishnamurti Kannan, "The Architecture of a Database Computer - Part II: The Design of Structure Memory and itsd Related Processors," Technical Report OSU--CISRC-TR-76-2, Computer and Information Science Research Center, The Ohio State University, Columbus, Ohio (National Technical Information Service Number AD/A-035 178)

[HSI76b] Hsiao, David K. and Krishnamurti Kannan, "The Architecture of a Database Computer - Part III: The Design of the Mass Memory and its Related Components," Technical Report OSU--CISRC-TR-76-3, Computer and Information Science Research Center, The Ohio State University, Columbus, Ohio (National Technical Information Service Number ADA-036 217)

[KANN78] Kannan, Krishnamurthi, "The Design of a Mass Memory for a Database Computer," Proc. Fifth Annual Symposium on Computer Architecture, Palo Alto, CA. April 1978.

[LANG78] Langdon, Glen G., "A Note on Associative Processors for Data Management," TODS, Vol. 3, NO. 2, June 1978, Pages 148 - 158.

[LIN 76] Lin, S.C., D.C.P. Smith, and J.M. Smith, "The Design of a Rotating Associative Memory for Relational Database Applications," TODS vol. 1, No. 1, pages 53 - 75, Mar. 1976.

[LIPO78] Lipovski, G. J., "Architectural Features of CASSM: a Context Addressed Segement Sequential Memory", Proceedings, Fifth Annual IEEE Symposium on Computer Architecture, April, 1978.

[OZKA75] Ozkarahan, E.A., S.A. Schuster, and K.C. Smith, "RAP - Associative Processor for Database Management," AFIPS Conference Proceedings, vol. 44, 1975, pp. 379 - 388.

[OZKA77] Ozkarahan, E.A., Schuster, S.A. and Sevcik, K.C., "Performance Evaluation of a Relational Associative Processor," ACM Transactions on Database Systems, Vol. 2, No.2, June

1977. Commmunications ACM 17, 7 , July,1974.

[RITC78] Ritchie, D.M. "A Retrospective," The Bell System Technical Journal, July-Aug. 1978, Vol. 57, No. 6, part 2, pp. 1947-1969.

[SCHU78] Schuster, S. A. et al, "RAP.2 - An Associative Processor for Data Bases", Proceedings, Fifth Annual IEEE Symposium on Computer Architecture, April, 1978.

[SLOT70] Slotnik, D.L. "Logic per Track Devices" in "Advances in Computers", Vol. 10., Frantz Alt, Ed., Academic Press, New York, 1970, pp 291 - 296.

[STON76] Stonebraker, M. et. al., "The Design and Implementation of INGRES," TODS, Vol 1, No. 3, September 1976.

[SU75] Su, Stanley Y. W., and G. Jack Lipovski, "CASSM: A Cellular System for Very Large Data Bases", Proceedings of the VLDB, 1975, pages 456 - 472.

[SU79] Su, Stanley Y. W., "Cellular-Logic Devices: Concepts and Applications", IEEE Computer, March 1979, pages 11-28.

6. Appendix I

Overview of proposed database machine architectures

In this section each database machine is briefly described (for more details on each machine the reader is encouraged to examine the appropriate references) and its operation is illustrated with an example. That example is the following query, QE:

retrieve (EMP.name, EMP.salary) where EMP.dept = 10

Each machine is also illustrated with a figure. For comparison, Figure 6.1 shows a standard computer system (i.e., one that does not include a back-end machine). In that system, the data blocks which are read are serially processed by the disk controller and the channel.

6.1. Associative disks

Most earlier designs for hardware to enhance the performance of data management systems were associative disk designs. First proposed by Slotnik [SLOT70], the design is to attach a processor to each of the heads of a head-per-track device (disk or drum). Figure 6.2 shows an associative disk system. The per-track processors are denoted as cell processors, and the processor that co-ordinates their activities is the controlling processor.

The cell processors can be loaded by the controlling processor with the value or values to search on, the search can take place in parallel, and the only data returned to the main computer are the records with the required values. As originally designed [SLOT70], the cell processors performed no arithmetic functions (e.g. sum, max, min, etc.). They were only search

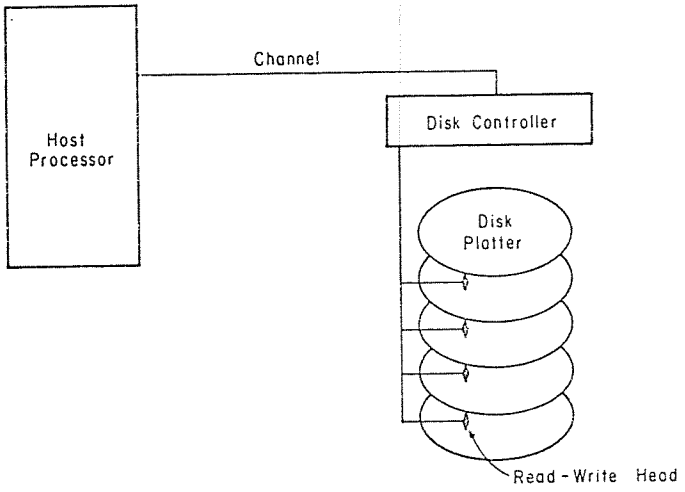

Figure 6.1

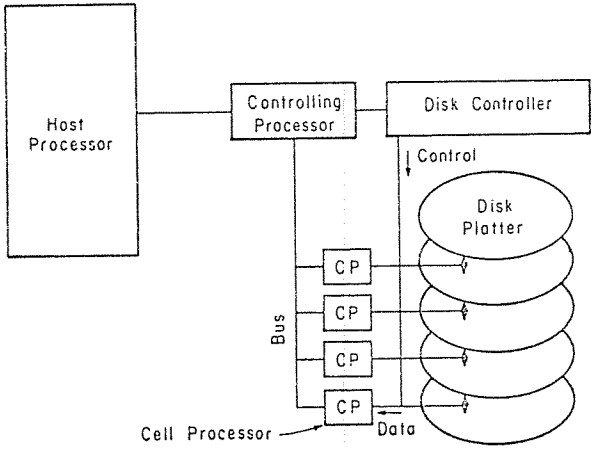

Figure 6.2

engines.

In Slotnik's associative disk system, query QE above would first be processed in the host machine and then a command would be sent to the associative disk to return all records in which the dept field = 1Ø. The host processor would then format the tuples for printing, taking only the salary and name field from each tuple.

If all the cell processors attempt to return a value to the controlling processor at the same time serious performance problems can occur. There may be bus contention problems on the data bus from the cell processors to the controlling processor. If the system does not bottleneck at the bus, the controlling processor may have problems keeping up with the data rate of all of the cell processors transferring at once.

6.2. CASSM

CASSM (Context Addressed Segment Sequential Memory) is the data management machine developed at the University of Florida. It was developed after associative disks, but before RAP, and represents a middle ground between them. CASSM is essentially a processor-per-head device, like an associative disk, but the processors have added capability in that they can perform a few arithmetic functions (integer sum, min, max).

The data is laid out in segments, where each segment is operated on by a single processor. The segments are not independent modules, however; data can freely migrate across segment boundaries. The processors operate under the direction of a

separate controlling processor. On the fixed head disk, each segment is one disk track.

For our analysis, we have assumed that CASSM is implemented on a moving-head disk. In this case its architecture closely resembles that of the associative disk, and is also represented by Figure 6.2. The differences in the systems lie in the power and function of the cell and controlling processors.

In CASSM, the processors all execute the same function at the same time, where the functions are data-processing directives (search for, delete, add, etc.) The processors have a minimum of buffer space, and will only test for a single qualification on one attribute at a time. The system includes special-purpose functions for string searches, updates, inserts, and garbage collection. The processors also have the capability of following pointers within the records written on the disk, so the hierarchical and network data models can be supported as easily as the relational model. The capability exists to write instructions on the disk itself, to be read later by the processors to change their actions. This capability is used, for instance, to change the function performed after one pass over the data.

A significant feature of CASSM is that data is stored by attribute rather than by tuple or record. Each attribute is flagged with its name and attributes are grouped together in records. Each record is defined by delimiter fields and a record number. Groups of mark-bits are associated with both individual attributes and records. One of the mark-bits is a collection bit, which if set signifies that the CASSM processors should send

the data to the host processor.

A data encoding algorithm is implemented in the hardware: each character-string value is stored only once, in a table of values. This table is stored as one of the segments. In the actual record, a pointer to the value is kept.

CASSM is limited in that only one attribute at a time may be tested or output. However, the testing of one attribute may be overlapped with the output of another.

The following explanation of the processing of a query in CASSM is based on a narrative of the execution of a similar query in [SU75]. To process QE, CASSM will, in the first cycle, mark all records that have the attribute-value pair (relation, EMP). Then, in the next cycle, the attributes within the marked records are inspected and marked for collection if the record contains the attribute-value pair (dept,10). The marked attributes "salary" are then returned to the host on the next cycle, while CASSM follows the name pointers in the marked attributes "name". The final cycle returns the name fields. The host machine must assemble the tuples to be printed by matching record numbers.

In the case that many values must be sent simultaneously to the controlling processor, the same performance problems occur as in the associative disk. CASSM deals with these problems by having the cell processors request the bus when they have a data item to transmit; if it is unavailable, the processor waits until it can transmit the data item it has before reading the next block. Therefore bus contention problems may result in the query requiring several extra disk revolutions.

6.3. RAP

The relational associative processor (RAP) is similar to CASSM in that it was at first implemented by attaching multiple processors, one per head, to a fixed-head disk. However, RAP is very different from CASSM in that RAP supports only the relational data model.

There are three functional parts to the RAP design: the controller, the statistical unit, and the cells. On a fixed-head disk each cell is a disk track. There is one processor per cell. The controller communicates with the front-end computer and directs the actions of the cell processors and the statistical unit.

Figure 6.3 shows the RAP system. The "controlling processor" box includes the statistical unit, and directs the functions of the cell processors. RAP is more powerful in function than CASSM because of the existence of the statistical unit, which performs more functions than the CASSM processors, and because each cell processor contains several comparator units, so several attributes can be tested simultaneously. RAP organizes data in relations which are stored in the data cells.

RAP, unlike CASSM and the associative disks, is a currently active research project. Subsequent implementations of RAP have used CCDs [OZKA77] and may eventually use random access memory [SCHU78] in place of the fixed-head disk. The version of RAP used in this discussion is that reported in [SCHU78]. It allows an individual cell to contain parts of several relations.

QE would be processed by RAP by passing the query to the RAP

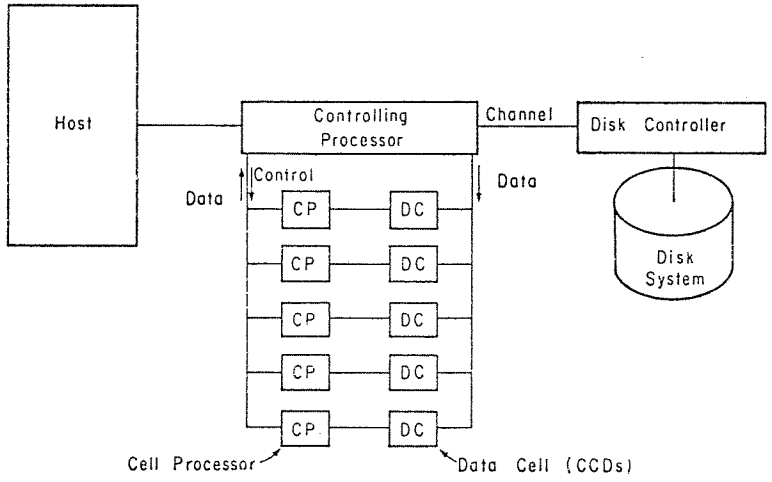

Figure 6.3

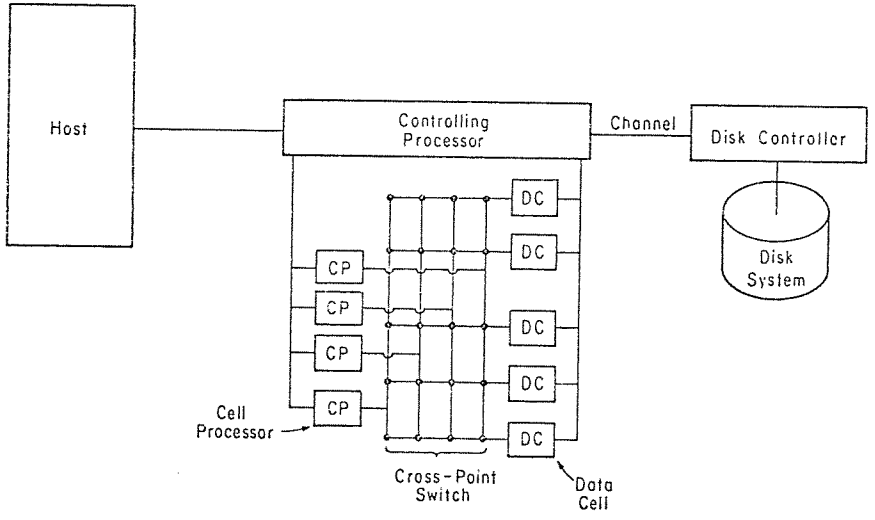

Figure 6.4

controller, which would determine which cells contain the "emp" relation, and direct the appropriate processors to return all tuples for which the dept = 10. The controlling processor would then pass these tuples to the host machine. In the case of bus contention the cell processors hold the data until the bus is free, thus potentially losing revolutions of the cells.

6.4. DBC

The data management machine being developed at Ohio State is called the Data Base Computer (DBC). Like CASSM, it is an attribute based system. It is a backend machine with seven major components, which are:

1) The data base command and control processor (DBCCP), which fields queries, communicates with the host machine and controls the functioning of the other components.

2) The keyword transformation unit (KXU), which forms an encoded version of the keywords to send to the next unit.

3) The structure memory (SM), which takes the encoded keyword and looks it up in a directory to determine the positions (indices) of the matching attributes in secondary storage.

4) The structure memory information processor (SMIP), which performs set operations on the information from the structure memory.

5) The index translation unit (IXU) which decodes the information about the location of the attributes required in order to produce a physical address on the secondary memory.

6) The mass memory (MM), which is composed of moving head disks.

Each disk has a processor per track (the Track Information Processors) that will perform basic functions.

7) The security filter processor (SFP), which contains a capability list of who has permission to access what data.

In the block diagram Figure 6.2, all of the above components except the MM are grouped together in the "controlling processor" box. The following explanation of the processing of query QE is derived from [BAN78A] and [BAN78B].

The DBCCP receives the command:

Retrieve (name, salary) ((relation = 'emp') & (dept = 10))

from the host processor. The predicate ((relation = 'emp') & (dept = 10)) is analyzed by the KXU, which determines if dept or relation are clustering keywords. In the case that they both are, the KXU transforms them into a coded, internal representation for look-up within the SM. The SM produces a series of index terms for relation = emp and for dept = 10 which are a coded representation of the cylinder numbers for the data. The SMIP performs a logical AND on the coded cylinder numbers so that only the coded cylinders for the data satisfying the predicate are passed to the next unit, the IXU. It transforms the internal, coded representations into actual disk cylinder numbers, and passes the information to the DBCCP. The DBCCP then directs the MM to perform the given task on the proper cylinders (find those records with dept = 10 and relation = emp, and output the name and salary fields). The data goes from the MM to the SFP which checks to determine if the access is proper. If so, the results are sent to the DBCCP, which sends them on to the host computer.

6.5. DIRECT

DIRECT is the backend data management machine under development at the University of Wisconsin-Madison (refer to Figure 6.4). It is composed of a controller, an arbitrary number of query processors, a CCD cache consisting of a set of page frames, and mass storage. The page frames are connected through a special cross-point switch [DEWI79a] to the processors. This cross-point switch permits each processor to read/write a different page simultaneously and two or more processors to read the same CCD page concurrently. The controller receives the query from the host computer and directs the processors to act on the proper pages. If the pages are not in the cache, the controller initiates a disk transfer and allocates page frames for them. Each processor can execute its instructions independently, so that it is possible that each processor can be executing a different query. The controller estimates the optimal number of processors to execute a query so that total system throughput is maximized. In [BORA79a] four alternative processor allocation strategies are analyzed.

To execute QE, DIRECT receives a compiled representation of the query from the host computer. After examining the query, the controller assigns a set of the available processors to work on the query. Through the use of the "next_page" operator [DEWI79a], each processor examines a distinct subset of the EMP relation. After a processor makes a request to the controller for the "next_page" of the EMP relation, the controller returns the address of the appropriate page frame. The processor reads the

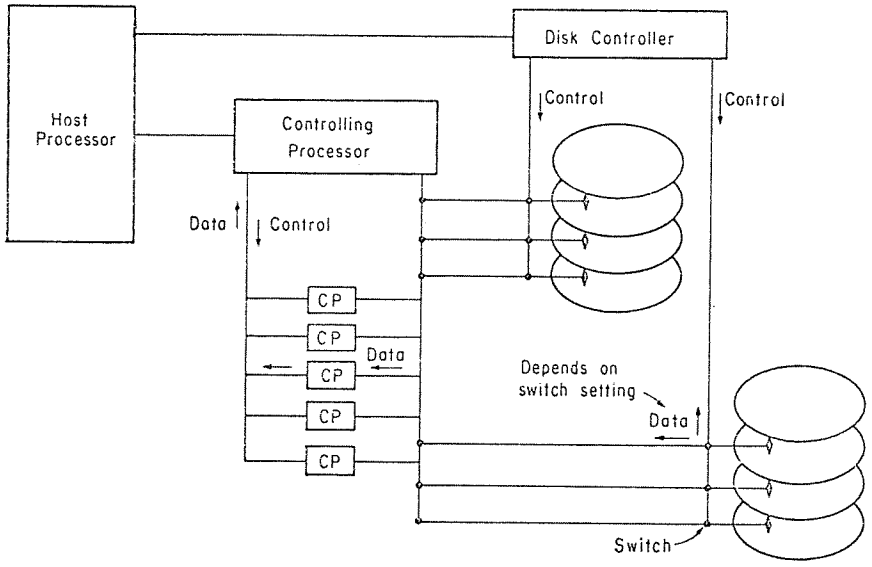

Figure 6.5

page into its local buffer and scans the page for tuples which satisfy the query. Qualifying tuples are placed in an internal buffer. After the query processor finishes scanning a page of the source relation it requests the next page from the controlling processor. When the query processor either fills its internal result buffer or receives a "no next page" notice it writes the temporary relation page into a new CCD cell.

6.6. CAFS

The Content Addressable File Store (CAFS) consists of a filter box between the disks and the host computer. This box can simultaneously operate on twelve different data streams to allow only qualified tuples to pass through to the host. The system reported in [BABB79] involves using disks whose track heads can be multiplexed to the CAFS box. They can also transfer data directly to the host, bypassing the CAFS system. Figure 6.5 shows CAFS.

The first action by the host processor to run query QE with CAFS is to load the CAFS box with the search key (dept = 10) and to connect each of the disk tracks that contain the EMP relation to the box. Qualifying tuples will be passed to the host. Collisions that occur in CAFS will result in extra disk revolutions.