DATABASE MACHINES: AN IDEA WHOSE TIME HAS PASSED? A CRITIQUE OF THE FUTURE OF DATABASE MACHINES

by

Haran Boral
David J. DeWitt

Computer Sciences Technical Report #504
July 1983

Database Machines: An Idea Whose Time has Passed?
A Critique of the Future of Database Machines

Haran Boral
Computer Science Department
Technion - Israel Institute of Technology

David J. DeWitt
Computer Sciences Department
University of Wisconsin - Madison

This research was partially supported by the National Science Foundation under grant MCS82-01870 and the Department of Energy under contract #DE-AC02-81ER10920.

ABSTRACT

In this paper we take a critical look at the future of the field of database machines. We hypothesize that trends in mass storage technology are making database machines that attempt to exploit a high degree of parallelism to enhance performance an idea whose time has passed.

1. Introduction

In 1979 our colleague David Hsiao wrote an introductory paper [HSIA79] for a special issue of the IEEE Computer Journal on database machines. He titled that article "Database Machines are Coming, Database Machines are Coming". Indeed at that time the future of database machines looked rosy. A large number of research projects were active and several commercial products were on the horizon. Although the designs and assumptions of the projects varied dramatically, the fundamental design objective of each project was to improve access to very large databases.

In this paper we will explore what has happened to database machines in the past four years and will attempt to predict the future of the area. Although we were tempted to title this paper "Database Machines are Dead, Database Machines are Dead", a statement this strong is probably premature. However, we assert that unless some fundamental changes occur in commercially available mass storage technology, the use of massive parallelism or associative memory for reducing the execution time of a single query is definitely an idea whose time has passed.

We will begin by describing in Section 2 the changes in technology that started the database machine revolution. We will then trace the evolution of the field by briefly describing a number of key projects. In Section 3, we will discuss the trends in mass storage and processor technology since 1970 and their effects on the future of database machines. Our assertion is that highly parallel high-performance database machines (of the type described in Section 2) are predicated on the availability of mass storage technologies that have not and probably will not emerge as commercially viable products. In Section 4, we suggest three avenues of research that seem promising for resolving the lack of sufficient mass storage bandwidth. Our conclusions are presented in Section 5.

2. The Desert Blooms

In this section we chronicle the development of database machines during the past twelve years. We have divided the numerous machines that have been proposed into three general classes: processor-per-track designs, processor-per-head designs, and off-the-disk designs. In the following sections we present an overview of each classification and describe several machines belonging to each.

2.1. Processor-per-Track (PPT) Architectures

Database machines were pioneered by Slotnick [SLOT70] who in 1970 formulated the idea of associating processing logic with the read/write heads of a rotating storage device such as a fixed head disk, as shown in Figure 1, in order to process certain search operations "on the fly". The processors are connected to a global data bus onto which they place selected records for transmission to the host processor. Coordination of the operation of the cells is performed by a controlling processor. The basic motivation of Slotnick's design was to be able to search a database directly on the mass storage unit and thus limit the amount of data to be transferred to the host for

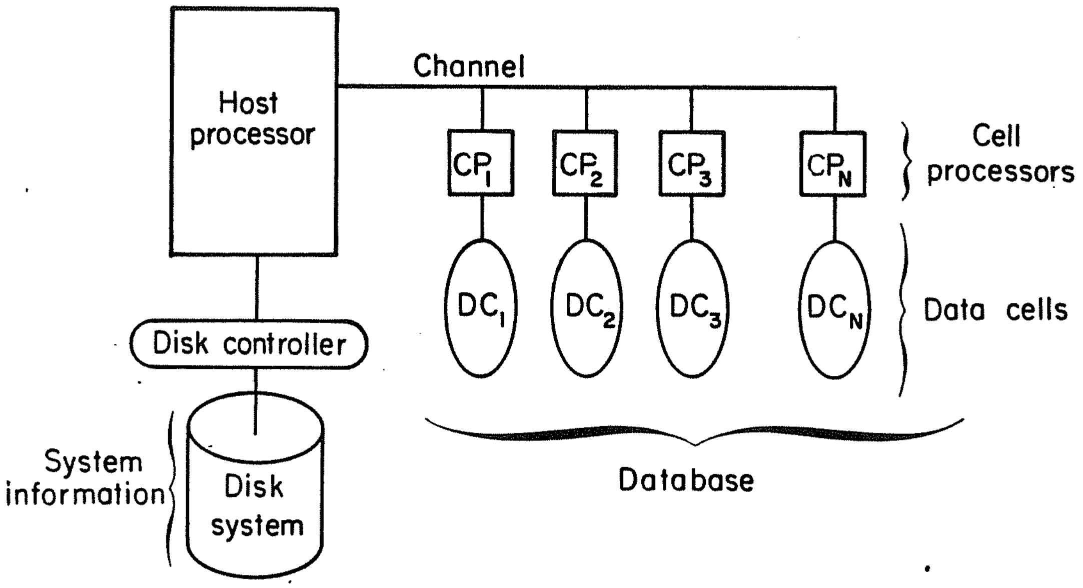


Figure 1

additional processing. In addition, since an entire database could be searched in one revolution of the mass storage device, the need for auxiliary storage structures such as indices to enhance performance was eliminated simplifying, considerably, the DBMS software.

Slotnick's ideas were further developed by Parker [PARK71], Minsky [MINS72], and Parhami [PARH72]. Although none of these efforts resulted in a comprehensive proposal for the implementation of a database machine, they served as a source of ideas for subsequent efforts; in particular CASSM, RAP, and RARES which are described briefly below.

2.1.1. CASSM

CASSM was the first complete database machine design. It was designed to support the network, hierarchical, and relational data models. A fixed head disk is used as the storage medium and a simple processing element is associated with each read/write head. The processing elements are controlled by a single processor which is responsible for communication with the host computer(s), distributing instructions to the processors, and collating and processing both intermediate and result relations.

Data items in CASSM are stored as ordered pairs: <attribute name, value>. All data items belonging to a record are stored in a physically contiguous block preceded by record and relation identifiers. Associated with each attribute and each record are a set of mark bits. These bits are used to indicate the results of an operation. Strings are stored only once in the database, separately from the records in which they are values. In these cases the value field of the ordered pair is a pointer to the string

When executing a selection query, a processing element marks tuples belonging to the relation being processed during the first disk revolution. A second revolution is used to search the marked records for the desired attribute and check its value; qualifying attributes are marked. A third revolution is used to output the marked attributes. In the event that the marked attribute is a string, the third revolution is used to chase the pointer in the value field of the marked ordered pair and an additional, fourth, revolution is required to output the marked string. Joins were implemented using a hashing scheme and an auxiliary memory similar to algorithms proposed for the CAFS [BABB79] and LEECH [MGCR76] database machines.

2.1.2. RAP

The RAP [OZKA75, OZKA77] database machine project is also based on a PPT approach in which tuples from a relation are stored bitwise along a track. Only tuples from one relation are allowed on a track, although numerous tracks can be used to store a relation. As in CASSM, a tuple is augmented with a fixed number of mark bits (attributes are not) that serve the same purpose. Processing of a selection operation is similar to CASSM, although it is faster because of the simpler data structure. Also, the processing elements have the capability of scanning for a number of different values in parallel. Joins are processed as a series of selection sub-queries on the larger relation, using the values of the joining attribute in the smaller relation as the selection criteria (a parallel nested loops algorithm).

2.1.3. RARES

RARES [LIN76], like RAP and CASSM, employs a PPT design utilizing a third storage format. Tuples are stored across tracks in byte parallel fashion. That is, byte 0 of a tuple is stored on track 0; byte 1 of the same tuple is stored in the same position on track 1; and so on. In this orthogonal approach a tuple that is "waiting" to be sent will be distributed over several buffers (a byte in each processing element) whereas in RAP the tuple will reside in a buffer in a single processing element. Thus, RAP would be more likely to be blocked than RARES due to bus contention.

2.1.4. Discussion

Although the PPT designs initially looked attractive, researchers quickly realized that they could not be used for the storage of very large databases (which was the original motivation for database machines). Their fundamental flaw is that a track of data on a magnetic medium was limited in 1970 to about 15,000 bytes [GORS80]. Thus to hold a 150 million byte database (which is not very big), a device with 10,000 tracks and processors would be required. Researchers attacked this problem in three ways. The PPT advocates turned to solid state devices such as bubble memories and charge coupled devices to provide longer tracks. We will address the problems associated with this solution in Section 3. Another group of researchers turned to an approach in which a processor is associated with each read/write head of a moving head disk. We term this approach "processor-per-head". It is discussed in the following section. A third solution investigated was to separate the processors from the storage medium with a large disk cache. The goal of this "off-the-disk" approach was to be able to continue to exploit parallelism to enhance performance while using conventional (and hence cheap) mass storage devices for holding the database. We discuss this approach in more detail below.

2.2. Processor-per-Head Designs

A second class of database machines are those that associate processing logic with each head of a moving-head disk as illustrated in Figure 2. We term this class of machines "processor-per-head" (PPH) machines. In a PPH database machine, data is transferred, in parallel, from the heads to a set of processors. Each processor applies the selection criteria to its incoming data stream and places selected tuples in its output buffer. In such an organization an entire cylinder of a moving head disk is examined in a single revolution (assuming no output bus contention).

2.2.1. Ohio State Data Base Computer (DBC)

The DBC [KANN78, BANE78] project chose the PPH approach as the basis for the design of their "Mass Memory Unit" as PPT devices were not deemed to be cost-effective for the storage of large databases (say more than 10^{10} bytes) [KANN78]. DBC consists of seven functionally different components. Of particular interest are the Mass

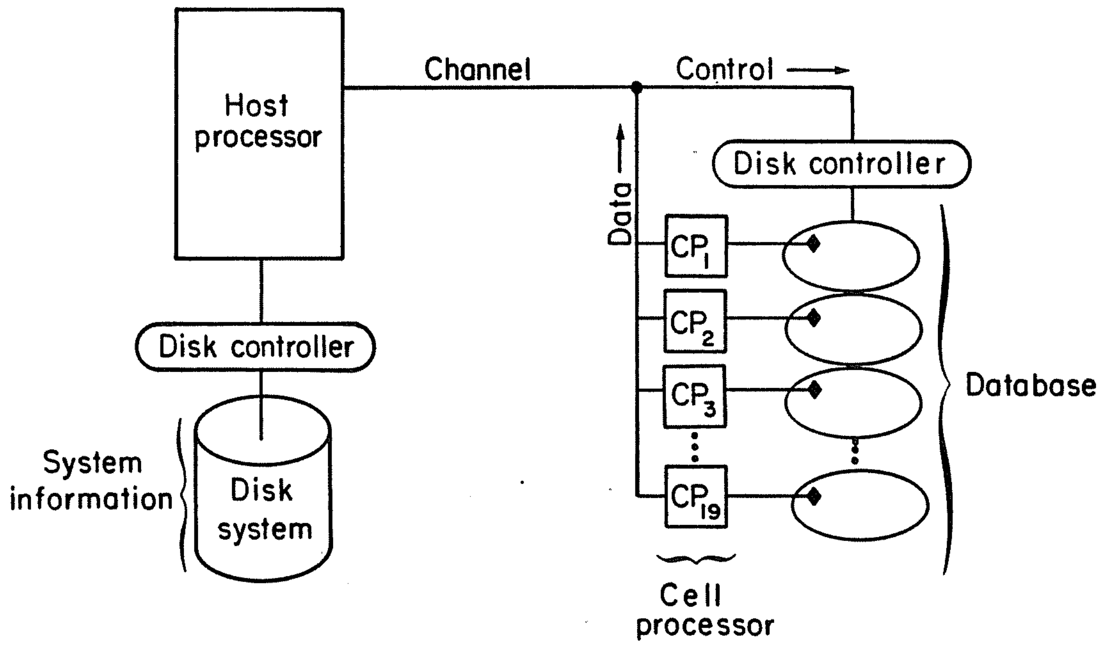


Figure 2

Memory and Structure Memory components. The mass memory uses several moving head disks, with parallel readout capabilities, to store the database. The heads of the disks are connected, via a switch, to a number of processors which perform search operations.

While an entire cylinder can be searched in one revolution, executing search queries which access data that spans many cylinders in a brute force fashion is not effective [DEWI81]. The DBC designers recognized this fact and incorporated indices into their design. The indices are kept in the Structure Memory. These indices are used to limit the number of cylinders that must be searched when executing a query.

2.2.2. SURE

SURE [LEIL78] uses a moving head disk modified to enable parallel read out from all of the recording surfaces simultaneously. The output is collected into a single high-speed broadcast channel from which it is read by a number of processors. Each processor is a very high-speed pipelined unit with a simple instruction set geared towards searching. A selection query is broken down into as many simple components as possible, each of which is assigned to one of the processors for execution. The actual number of processors used for the execution of a selection query depends on its complexity. Like RARES, SURE is intended to function only as a search processor, perhaps in the context of a database machine or serving a DBMS on a general-purpose computer.

2.2.3. Discussion

In [DEWI81] the performance of a number of alternative database machine architectures were evaluated for a range of different query types. The result of this evaluation indicates that the PPH design provides very good performance when searching a file for those records that satisfy a search condition. When a suitable index was available, a PPH organization with 19 processors (and read/write heads) was never more than a factor of 1.6 slower than a PPT organization with 7676 processors and tracks. Without a suitable index, the PPH organization was approximately a factor of 4 slower.

When, however, this same design was used to process complex queries, such as the relational algebra join operation by repeatedly processing one relation using values from the second relation (the RAP join algorithm), both it and a "generic" PPT design performed significantly slower than a conventional processor. This indicates that the PPH design alone does not provide a suitable level of performance for all relational algebra operations. To solve this problem, the designers of the DBC augmented their original design to include a postprocessing unit for processing complex database operations such as the relational join operator.

2.3. Off-the-Disk Machines

Another solution to the database size problem is to separate the processors from the storage medium with a large disk cache. We term this approach the "off-the-disk" approach. The goal of this approach is to be able to continue to exploit parallelism to enhance performance while using conventional mass storage devices. While all data to be processed must first be moved from mass storage to the disk cache, once the data is there it can be accessed by the processors in parallel (assuming the existence of suitable hardware). Furthermore, intermediate results in processing a query will be placed by the processors in the disk cache and hence be quickly accessible for the subsequent operations in the query.

There is a large number of database machines in this class including RAP.2 [SCHU79], DIRECT [DEWI79], INFOPLEX [MADN79], RDBM [HELL81], and DBMAC [MISS81]. In addition, some database machine designers have designed machines that combine the characteristics of the PPH and "off-the-disk" designs (REPT [SHUL81], HYPERTREE [GOOD81]). As discussed in [DEWI79], such designs look promising if parallel readout disks can be constructed (a point we will return to in Section 3). We will briefly describe DIRECT and RAP.2 in the following sections.

2.3.1. RAP.2

In [SCHU79], Schuster et al describe a virtual RAP machine. In this organization the database resides on some number of conventional mass storage devices. The RAP system consists of a number of cells, each with a pair of tracks. The controller assumes the additional responsibilities of loading the tracks with data to be examined. Each processor can examine only one track at a time. However, while one track is being examined, the second can be loaded.

2.3.2. DIRECT

In this section we describe the architecture of DIRECT. The DIRECT project began after a critical evaluation of several database machine research projects (in particular RAP which in the mid 70's was the most advanced and best known database machine design) revealed two major shortcomings with each of the proposed designs: the inability to process complex database operations (e.g. joins) efficiently [OZKA77,DEWI81], and the SIMD mode of operation in which only a single instruction from a single query can be executed simultaneously. As shown in [BORA81] a database machine that processes queries in a SIMD mode has a lower transaction throughput than one which processes multiple operations from multiple queries simultaneously. DIRECT was designed to overcome these two shortcomings.

In DIRECT there are some number of processors whose function is to execute selection, projection, join, update, and aggregate operations on the database. These processors are controlled by a processor (termed the back-end controller) which is responsible for distributing instructions and overseeing data transfers to the processors. The database resides on some number of mass storage devices (moving head disks). Each relation is organized as a set of fixed size pages. The disk cache is divided into page frames of the same size. The query processors and disk cache page frames are connected by a cross point switch that has two important capabilities: any number of processors can read the same page simultaneously; and, any two processors can read/write from any two different devices concurrently. This organization permits processors to work in parallel on the same operation or on different operations.

2.3.3. Discussion

Although both the RAP.2 and DIRECT designs appeared promising initially, the research presented in [DEWI81] indicates that when the performance of a DIRECT-like design is compared to that of a conventional computer, increases in parallelism do not result in corresponding decreases in query execution time.

3. The Flowers Wilt

In the previous section we described three classes of database machine architectures: processor-per-track designs, processor-per-head designs, and "off-the-track" designs. In this section we will discuss how recent changes in mass storage technology have either made each of these designs infeasible or of reduced value with present day or near future technology.

Although we realize that all database machines may not fit directly into our classification, we contend that most database designs that *utilize parallelism* can be represented as a combination of these architectures. Thus, since we have serious doubts about the near term viability of each approach, just patching different approaches together does not resolve the fundamental problem we are facing: *current mass storage devices do not provide sufficient I/O bandwidth to justify highly parallel database machines.*

The reader should not, however, interpret this statement to mean that we feel that database machines that use limited parallelism (such as the IDM 500 [IDM500] with its database accelerator co-processor) are not viable. On the contrary such database machine designs with 1-3 processors per disk drive look very attractive (see the conclusions of [DEWI81]) and a reasonable design might a collection of such clusters interconnected by a high-speed local network [STON79].

In this section we first discuss the impact of trends in mass storage technology on the processor-per-track, processor-per-head, and off-the-disk database machine designs. Then we briefly discuss trends in processor technology. Finally, we present our interpretation of the compounded effect of these two trends on the future of database machines.

3.1. Trends in Mass Storage Technology

3.1.1. Impact on Processor-per-Track Database Machines

PPT designs were predicated on: 1) associating a processor with every storage cell, and 2) the availability of a sufficient number of storage cells to hold the entire database. Thus, conventional database techniques such as indexing would be never by needed by these designs. Unfortunately, the technologies on which these designs were predicated have either become obsolete or not commercially viable.

Fixed head disks on which the original designs were based are no longer manufactured[1]. Furthermore, storage of entire databases on fixed head disks never has been and never will be cost effective.

[1] We are aware that certain models of winchester type disk drives such as the IBM 3380 have a limited number of tracks with fixed read/write heads. These tracks do not, however, store enough data to consider using them as the basis for a database machine.

The other technologies considered as the basis for the storage cells of the processor-per-track designs were bubble memory and charge-couple-device (CCD) components. Although constructing mass storage devices from either CCD or bubble devices may be academically interesting, at this point in time it does not appear that either technology is likely to become a suitable replacement for moving head disk technology. Since their introduction, the cost/bit of CCD and bubble memory devices has not dropped as rapidly as that of moving head disks or random-access memory components. Although bubble memories are being used in a small niche of the market (mainly non-volatile memory for portable terminals), the state of the art is that Intel sells a board with bubbles that is slower than hard disk and more expensive than RAM.

CCDs have failed to generate a big enough market for parts where the access time did not matter. After their introduction, CCDs never were more than a half generation ahead of RAMs. This meant that if a customer was willing to wait an additional year, he would be able to purchase a RAM chip with the same density for the same price but with an access time of about two orders of magnitude faster. Consequently, most semi-conductor manufacturers have stopped manufacturing both types of components. It is likely that neither of these technologies will ever become cost effective for storing even moderate-sized databases.

In conclusion, it appears that processor-per-track database machines have only historical value.

3.1.2. Impact on Processor-per-Head Database Machines

When a parallel-readout disk drive is combined with a simple processor for each read/write head and an indexing mechanism to limit the number of cylinders that must be searched, the result is a database machine that can process *selection* queries at a very high rate [DEWI81]. In addition, when such a design is combined with an off-the-disk design, the performance for processing complex queries is also excellent. Why then is such an architecture not the wave of the future? The problem is that the same changes in magnetic disk technology that have produced constantly increasing disk capacities have made constructing parallel-readout disk drives increasingly more difficult.

To understand why it is difficult to construct a parallel-readout disk, one must reflect on the mechanism used to read/write conventional moving head disks. To minimize costs, all read/write heads are mounted on the same disk arm assembly which is then activated by some form of a linear motor. When the head assembly is moved to a cylinder (i.e. a seek operation), the read/write heads are not properly aligned on their respective tracks and fine adjustment is generally necessary to align the head selected for reading/writing. Due to the effects of thermal expansion of the arm, alignment must take place continuously (even while the track is being read/written). In parallel-readout disks all (or most) read/write heads must be aligned after each seek. If more than one head is to be aligned then the arm's position at any given time instance will most likely not be the optimal position for any one head and the signal received by each head will not be maximal; possibly leading to errors. As disk densities increase, the magnetic spots get smaller, further aggravating the effects discussed above.

Another problem with parallel readout disks is that the error detection and correction logic must be replicated for each head. Although we were not able to obtain precise figures on what part of the cost of a disk the error correction logic represents, we have been lead to understand that it is substantial. Associating error correction logic with each read/write head would contribute significantly to the cost of a disk. As an example, consider the winchester drive in which four of the 16 heads can be read/written at once made by CDC. The storage capacity of the drive is 675 megabytes, transfer rate of 4.84 megabytes/second, cost of $80,000. A drive of comparable storage capacity (484 megabytes) and a transfer rate of about 1.8 megabytes/second costs around $15,000 (Fujistu Eagle).

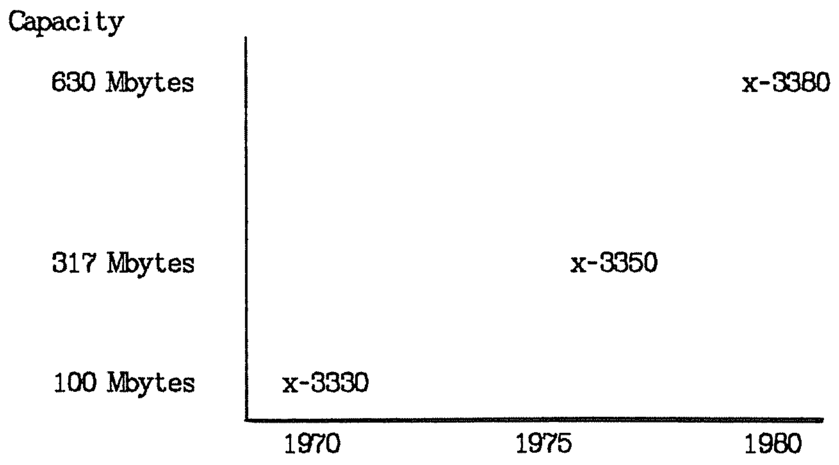
Thus, while parallel-readout drives were perhaps feasible 5-10 years ago, they are only marginally feasible today (from both technological and cost points of view). As we mentioned above, CDC does have a parallel readout drive with four heads operating in parallel. However, it is not clear whether their new generation of IBM 3380 compatible disks will include a parallel readout version.

Actuating each head with a different mechanical assembly would provide a solution to this problem. But since the linear motor and disk arm assembly with its associated positional sensing circuitry constitutes the major portion of the cost of a conventional disk drive, such a drive would be prohibitively expensive (and extremely bulky!). In addition to the problems of construction, such drives will have higher cost per bit of storage than conventional drives. For some applications (such as feeding data to a Cray-1) the performance benefits probably outweigh the cost disadvantages. For storage of large databases, their cost disadvantage will, in our opinion, make the use of such drives impractical.

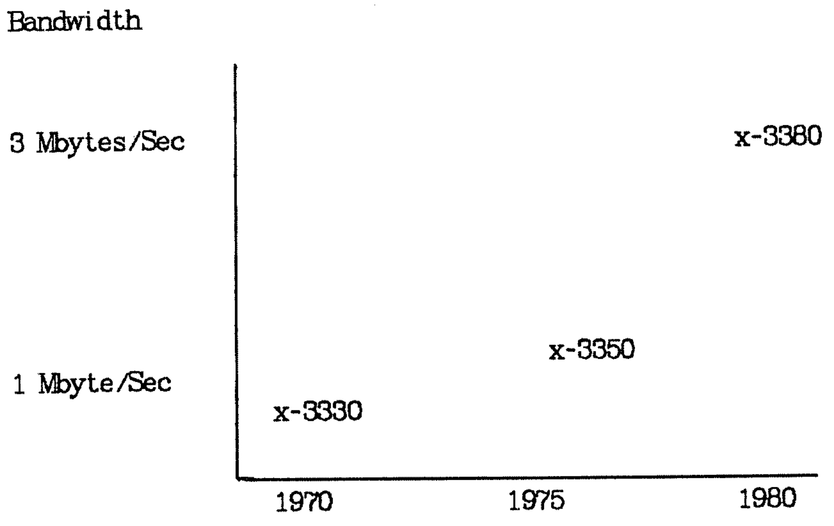
In conclusion, although parallel-readout disk drives could form the basis of high performance, highly parallel database machine, changes in disk technology have rendered this approach questionable and other approaches must be developed to provide high bandwidth mass storage devices.

3.1.3. Impact on Off-the-Disk Database Machines

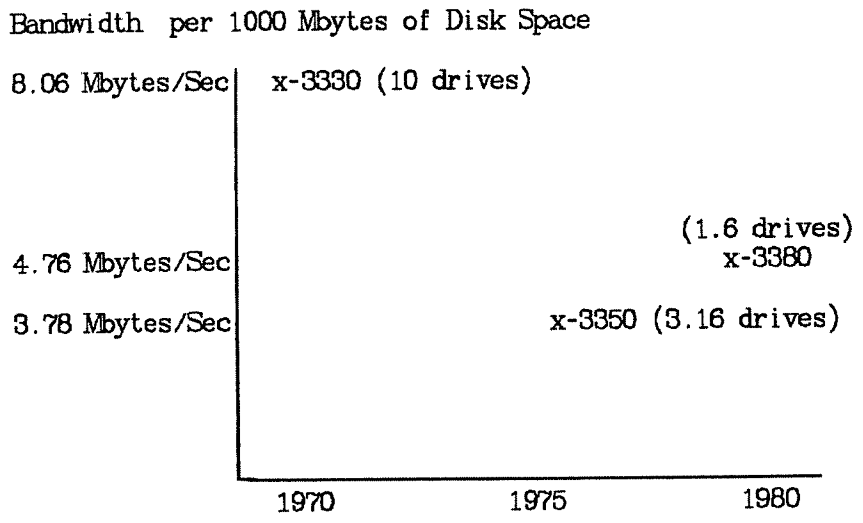
In addition to their effect on PPH designs, increasing disk densities also have had an adverse impact on the performance of off-the-disk database machine designs that utilize parallelism. Consider, for example, the three trends that moving head disk technology has displayed during the last ten years illustrated below in Figures 3, 4, and 5. Figures 3 and 4 illustrate encouraging trends for all computer applications: increased disk capacities (therefore cheaper cost per bit) and increased bandwidth per drive. When we combine the figures to obtain Figure 5 which shows the trend in bandwidth per gigabyte we find that the total bandwidth for a gigabyte of storage has actually decreased with improvements in disk technology from 1970 to 1975. The reason for this phenomenon is that almost every increase in disk capacly was the result of decreasing the distance between adjacent tracks on a surface andnot the result of shrinking the distance between bits along a track. Thus, increases instorage capacity per drive were largely due to improvements in the mechanical headpositioning technology.



Disk Capacity
Figure 3



Bandwidth Per Disk Drive
Figure 4



Total Available Bandwidth for 1 Billion Bytes of Storage
Figure 5

The IBM 3380 caused this trend to change. In the IBM 3380, the capacity of a single track was increased from a typical capacity of about 15,000 bytes to about 45,000 bytes. This increase is due to improvements in head technology that resulted in the shrinking of the area occupied by each bit. Unfortunately, the gains in bandwidth illustrated by the IBM 3380 in Figure 4 (and the corresponding change in direction of the curve in Figure 5) due to increased track capacities are not likely to continue. The 3 megabyte/second bandwidth of the IBM 3380, is difficult for most machines to handle (even on large IBM machines, a special channel is needed to handle these drives). Thus, it is not likely that track capacities will increase further (or at least the maximum bandwidth per spindle is likely to remain at the level of the 3380).

What we are saying then is that although the trends illustrated by all three figures seems encouraging, in fact they are not. The improvement in disk storage capacity is due to improvements in mechanical head positioning technology and in head technology itself. Improvements in head technology cause, in addition to increases in disk capacity, a corresponding increase in disk bandwidth. Improvements in mechanical head positioning technology do not affect disk bandwidth. Existing I/O processors cannot handle transfer rates higher than 3 megabytes/second.[2] Undoubtedly this will change over time. However, we believe that several years will pass until this change takes place. In the meantime additional increases in disk capacity will be made through further improvements in head positioning technology. Thus, for a database with a given size, the bandwidth available for access to that database is likely to decrease.

3.2. Trends in Processor Technology

While the capacity of mass storage devices has been growing, "slower" and "slower" conventional microprocessors have been getting faster and faster. In this section we will explore the tasks performed by the processing elements of a database machine and will demonstrate that advances in microprocessor technology have eliminated the need for customized database machine processors implemented either through microcode or VLSI.

[2] The Cray-1 and several of the higher-end CDC machines use the parallel readout CDC drive. To handle the high data transfer rate they have had to resort to the use of two I/O processors to control the disk and to ping pong control over the disk between them!

3.2.1. Description of Major Tasks Performed by the Processing Elements

All the database machine organizations described in the previous section are organized in a two tiered manner: at the top level a single processor that is responsible for communications with the outside world and management of the machine resources; at the bottom level several processing elements are responsible for executing tasks. Practically all of the overhead functions, such as disk arm scheduling and buffer memory management (if there is one in the machine), associated with executing queries, are handled by the controller. The processing elements spend almost none of their time executing any of these overhead tasks. The database machines discussed previously attempt to decrease the execution time of queries through the use of parallelism in the query execution itself not by speeding up the controller's function (an exception is the DBC which uses a form of pipelining to distribute the controller's functions among four or five specialized processors). Since the emphasis in most designs is on the use of parallelism in the lower level we now carefully examine the processing requirements of that level and ignore those of the top level.

The operations supported by most relational database systems, can be divided into two classes according to the time complexity of the algorithms used on a uniprocessor system. The first class includes those operations that reference a single relation and require linear time (i.e. they can be processed in a single pass over the relation). The most familiar example is the selection operation that selects those tuples from a relation that satisfy a predicate (e.g. suppliers in "New York").

The second class contains operations that have either one or two input relations and require non-linear time for their execution. An example of an operation in this class that references one relation is the projection operation. Projecting a relation involves first eliminating one or more attributes (columns) of the relation and then eliminating any duplicate tuples that may have been introduced by the first step. Sorting (which requires O(nlogn) time) is the generally accepted way of eliminating the duplicate tuples. The join operation is the most frequently used operation from this class that references two relations. This operation can be viewed as a restricted cross-product of two relations.

For both classes of operations, once a block (page) of data has been retrieved from mass storage to the memory of a database machine, there are three basic operations that must be performed efficiently while executing a query: search operations, moving data from one memory location to another, and indexing operations (pointing at the next tuple/record to be processed).

The operation of searching for a record (tuple) which satisfies certain conditions is used in a variety of relational operations including:

(1) selection - in which the search condition may be arbitrarily complex: name = "Jones" and salary > $10,000 or name = "Smith" and salary = $95,000

(2) join - in which an attribute of one tuple is compared against the attribute of another tuple

(3) projection - duplicate tuples are detected by comparing all attributes of one tuple with all attributes of another.

Moving records from one memory location to another also occurs in a number of database operations including sorting and placing tuples in the result relation of a query. Finally, while searching a block of tuples the task of locating the proper field in the next tuple to be examined is performed frequently and thus should be executed as quickly as possible.

3.2.2. Processing Requirements of these Operations

In a separate paper [BITT83], we report the results of running benchmarks on several database systems. We had hoped to utilize the results of these benchmarks to gather statistics on cpu time per disk access for the different operations. In particular, we were planning to use the Commercial Ingres numbers. However, the numbers we obtained from these benchmarks do not permit separation of cpu time into overhead and processing components. Instead we ran a small subset of the benchmark on the Wisconsin Storage System (WiSS) [CHOU83] running on a VAX 11/750.

A description of WiSS is beyond the scope of this paper. However, a few words are due. WiSS is an extent based file system implemented on top of Unix although bypassing the Unix file system. WiSS consists of several layers that do buffer management, index maintenance, recovery, concurrency control etc. From its inception, in 1981, it was intended to provide faculty and students at Wisconsin with a tool for database research. It has been used as a basis for student projects in database management systems and database machines.

All the major relational operators have been implemented on top of WiSS. Algorithms that rely on sorting, indexing, and other data structures have been implemented. For this paper we ran a small number of experiments trying to measure the amount of cpu time spent per access to disk. We looked at selection and join operations without indexes. The join was implemented using a sort merge algorithm. Selection was implemented using a sequential scan of the relation.

The main result of this effort is that a considerable amount of additional work is required if one is to obtain the kind of numbers we desire. There are numerous factors that contribute to the cpu time measured. Some examples are doing the measurements while other users were using the system, factoring out various overhead costs such as buffer management (not done by the processing elements of a database machine), and disk scheduling.

All tests were run on a 10,000 tuple relation with 182 bytes/tuple. The relation was organized as 4096 byte pages. Tuples were not stored across page boundaries. Pages were full.

Selection operations were evaluated on both 2-byte integer attributes and 52-byte string attributes. Selectivity factors of 1% and 10% were used for each attribute. We found that the average cpu time spent per disk access ranged between a little less than a millisecond to about 2.5 milliseconds.

A 10,000 tuple relation was joined with a 1,000 tuple relation to obtain a 1,000 tuple result relation. We ran the tests with different main memory buffer sizes. The buffer sizes evaluated were 3, 5, 10, 20, 50, and 100 (each buffer could hold a single 4096-byte page). We found that the cpu time time per disk access ranged between 13 and 22 milliseconds. In general, the execution time per disk access increased as the number of buffers available to the sort-merge routine increased (of course, the total execution time decreased).

What is the period of time required to access a page on disk? In the best of cases no seek will be required. Thus total time consists of latency and transfer (about 10 milliseconds on a Fujitsu Eagle disk). On the average the seek time of the Eagle is about 20 milliseconds thus we obtain a figure of 30 milliseconds for the random access case.

The conclusions that can be drawn from these figures are:

(1) A single processor with performance characteristics similar to the VAX 11/750 can process selection queries at the rate the disk delivers data.

(2) Two or three processors would be required to process joins.

3.2.3. Implications

Based on the above arguments, we claim that "off-the-shelf" processing components provide sufficient processing power and that there is no need to implement customized database processors either through microcode or VLSI as has been argued in [KUNG80] and [YAO81] for example.

Although microprogramming is frequently touted as providing orders of magnitude of performance improvement for time critical database operations such as searching and moving strings, we claim that the performance improvements are minimal at best [ATKI73, DEWI76]. For machines without string handling instructions the best improvement one is likely to see is a factor of 2-3. For processors with machine language instructions for processing strings, there is, in general, little or no improvement in performance as such operations are inherently limited by memory bandwidth.

Furthermore, development of customized microcode is a very time consuming process and has the tendency to lock a project into obsolete technology. As an example consider the Braunschweig database machine project RDBM. This project began in 1980. By the end of 1981, the design and initial implementation of the AMD 2901 processor modules was completed and microprogramming of the customized functions was initiated. A year later (December 1982), the microcoding still had not been completed. In fact, for a number of operations, microcoding was abandoned because it was just too time consuming. Although the RDBM may indeed turn out to be relatively fast, we claim that by using "off the shelf" microprocessors and a high level language implementation a machine with equivalent performance could have been constructed. In addition, by utilizing a component such as the Motorola 68000, the performance a database machine can relatively closely track improvements in processor technology. This cannot be accomplished in such a straight-forward fashion if microprogramming is used as an implementation technique.

We also feel that utilizing *custom* VLSI components as the basis of a database machine will not produce a superior database machine. We have two arguments to support this almost heretical claim. First, since many database operations involve manipulating and moving strings (which are memory intensive operations requiring a high percentage of off-chip accesses), customized VLSI components are not likely to outperform state of the art microprocessors. Second, microprocessors have not only been getting faster but they have also become more sophisticated -- having richer instruction sets and handling a variety of data types, such as floating point numbers and strings, in addition to integers. Examples of such advanced VLSI microprocessors include the HP 9000 product with a 18 megahertz cpu chip containing 450,000 gates, the 16 megahertz Motorola 68020, and the Intel 287.

3.3. Combining the Trends

When one considers the trend of decreasing I/O bandwidth per gigabyte of storage and the increasing performance of off-the-shelf microprocessors, one is forced to conclude that the limiting factor is I/O bandwidth. We are convinced that one needs at most 2 or 3 conventional processors and *not* tens or hundreds of processors or customized VLSI components to process data at the rate that can be produced by current mass storage technology. Thus, an architecture that blindly uses custom VLSI components or a lot of processors (no matter how they are interconnected) to process data that resides on few standard disk drives will inevitably be I/O bound. Even with IBM 3380 disk drives only several of today's microprocessors are needed (and perhaps only one with the power of a Motorola 68020). Researchers should concentrate their efforts at increasing the I/O bandwidth. In the next section we briefly outline three possible approaches.

4. Making it Rain

We contend that before continuing to explore highly parallel architectures, database machine designers must turn their attention to exploring mechanisms for increasing I/O bandwidth. In addition, it appears that the use of indexing is necessary with ever increasing disk capacities. However, while improving I/O bandwidth is critical, researchers must face reality and figure out how to do it with conventional disk technology. The purpose of a database machine is to improve access to very large databases and users are almost certainly not going to be willing to accept database machines if their performance depends on the utilization of expensive mass storage devices.

One possible approach for improving raw I/O bandwidth is to use unmodified disk drives with a customized disk controller. The basic idea is that instead of trying to read or write all the heads of a single drive in parallel, a number (e.g. 12) of standard drives are attached to a customized controller that has been designed to permit the simultaneous data transfer between the controller and a single head of all drives. This will permit all tracks of a logical cylinder (a logical cylinder is formed by the set of tracks at the same physical location on all the attached drives) to be accessed in parallel. Such a design should provide the same I/O bandwidth as the parallel readout disk while using standard disk drives. Furthermore, by using off-the-shelf drives, one will be able to take advantage of improvements in disk storage technology by simply replacing the set of old drives with new ones.

There are several technical issues to be addressed. Chief is the design of the controller. The controller must coordinate several disks, some of which may be rotating at slightly different speeds. Error handling by the controller also becomes quite tricky. Consider, for example, what happens when one of the drives has a read error. Should all the drives retry the read or just one that had the error? While it might be simplest to have all drives repeat the read, during the retry a different drive may have a read error. Another question warranting attention is if such an organization is utilized in a processor-per-head database machine where should the filtering processors be placed? As part of the controller or between the controller and the other components of the database machine? Finally, how should indexing be incorporated into this mass storage organization? We understand that this approach is being explored by Cray Research as an alternative to using customized parallel-readout disks.

A second approach that appears to be worth exploring is to front end a number of conventional disks with a very large (100-200 megabyte), very fast solid state (RAM) memory.[3] The memory could be used as a disk cache (much in the same way that the CCD memory was used in DIRECT). Data pages could be prefetched from disks in anticipation of their usage, and/or kept in the cache in anticipation of their future usage (as in the case of a sort-merge join). There are a number of open research problems with this approach. The foremost is whether "real" database applications runing in a multi-

[3] Certain large IBM machines have main memory bandwidths in the 100 megabytes/second range (achieved through a combination of fast memory components, a high degree of interleaving (8 - 16), and a wide path into memory (e.g. 8 bytes at time).

user environment demonstrate any form of temporal locality. That is, do application programs access different portions of the database in a random fashion or does the working set change gradually over time? The addition of virtual memory to RAP was justified on the notion of temporal locality [SCHU79]. To our knowledge almost no effort has been made to instrument a running database system to determine the extent to which real systems demonstrate locality in their references to the database.[4] If there is a fair degree of locality, then this approach might solve the I/O bottleneck. An experiment to determine locality certainly seems to be worthwhile. Other unsolved research problems associated with this approach include: development of replacement strategies that satisfy the needs of a variety of different applications, volatility and the notion of stable storage, and evaluation to see whether this approach will really work for a variety of applications.

Note that IBM has recently announced a disk controller that includes a disk cache that serves as a backup to main memory. The memory management algorithms used for managing the disk cache are different from those used for main memory. For example, data is prefetched from disk. IBM claims that the average access time for a disk with the disk cache is reduced to about 8 or 9 milliseconds from the usual 30-50. Our proposal in the previous paragraph is essentially similar, although specialized for database applications.

Finally, since increased disk densities seem to necessitate the use of indices to reduce the number of cylinders that need to be searched, more research is needed on effective index strategies in a database machine environment. For example, do we need a mechanism as complicated as the Structure Memory of the DBC or will a single fast microprocessor with its own dedicated disk for storage of indices be sufficient?

[4]One exception is [RODR]. However, the results of this effort are inconclusive.

5. Conclusions

In this paper we have examined the impact in changes of mass storage technology on three different classes of database machines organizations: processor-per-track designs, processor-per-head designs, and off-the-disk designs. We have shown that processor-per-track designs no longer look technically feasible and that increasing magnetic disk capacities have had very a negative impact on the potential of highly parallel database machine designs. We conclude that unless mechanisms for increasing the bandwidth of mass storage devices are found, highly parallel database machine architectures are doomed to extinction.

6. Acknowledgements

We gratefully acknowledge the support of the National Science Foundation under grant MCS82-01870 and the Department of Energy under contract #DE-AC02-81ER10920.

7. References

[ATKI73] Atkins, D., DeWitt, D., and M. Schlansker, "Studiesin the Applications of Microprogramming," ISDOS Working Paper No. 97, Dept. olIndustrial Engineering and Operations Engineering, University of Michigan, Decemler 1973.

[BABB79] Babb, E. "Implementing a Relational Database b) Means of Specialized Hardware," ACM TODS, Vol. 4, No. 1, March 1979.

[BANE78] Banerjee J., R.I. Baum, and D.K. Hsiao, "Concepts anl Capabilities of a Database Computer," ACM TODS, Vol. 3, No. 4, Dec. 1978.

[BITT83] Bitton-Friedland D., D.J. DeWitt, and C. Turbyfil, "CanDatabase Machines Do Better? A Comparative Performance Evaluation," Proceedigs of the VLDB Conference, 1983.

[BORA81] Boral H. and D. J. DeWitt, "Processor Allocation Stratgies for Multiprocessor Database Machines," ACM Transactions on Database System, Vol. 6, No. 2, pp. 227-254, June 1981.

[CHOU83] Chou H. T., D. J. DeWitt, R. H. Katz, and A. Klug "Dsign and Performance Evaluation of the Wisconsin Storage System," In preparation

[DEWI76] DeWitt, D.J., "A Machine Independent Approach to the Production of Optimized Horizontal Microcode," Ph.D. Dissertation, University of Miclgan, 1976.

[DEWI79] DeWitt, D.J., "DIRECT - A Multiprocessor Organization fr Supporting Relational Database Management Systems," IEEE Transactions on Coiputers, June 1979, pp. 395-406.

[DEWI81] DeWitt, D. J., and P. Hawthorn, "Performance Evaluatin of Database Machine Architectures," Invited Paper, 1981 Very Large Database Conference, September, 1981.

[GOOD81] Goodman, J. R., "An Investigation of Multiprocessa Structures and Algorithms for Data Base Management," Electronics Research Lboratory Memorandum No. UCB/ERL M81/33, University of California, Berkeley, Mq 1981.

[GORS80] Gorsline, G.W., *Computer Organization*, Prentice Hall,1980.

[HELL81] Hell, W., "RDBM - A Relational Data Base Machine: Arditecture and Hardware Design," Proceedings of the 6th Workshop on ComputerArchitecture for Non-Numeric Processing, June 1981.

[HSIA79] Hsiao, D.K., "Database Machines are Coming, Databse Machines are Coming!," Computer, Vol. 12, No. 3, March 1979.

[IDM500] IDM 500 Reference Manual, Britton-Lee Inc., Los Gatos California.

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

[KUNG80] Kung, H.T. and P.L. Lehman, "Systolic (VLSI) Arraysfor Relational Database Operations," Proceedings of the 1980 International Confereice on the Management of Data, May 1980, pp. 105-116.

[LEIL78] Leilich H.O., G. Stiege, and H.Ch. Zeidler, "A Search Processor for Data Base Management Systems," Proc. 4th Conference on Very Large Databases, 1978.

[LIN76] 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.

[MADN79] Madnick, S.E. "The Infoplex Database Computer: Concepts and Directions," Proceedings of the IEEE Computer Conference, Feb. 1979.

[MCGR76] McGregor, D.R., Thomson, R.G., and W.N. Dawson, "High Performance Hardware for Database Systems," in *Systems for Large Databases*, North Holland, 1976.

[MINS72] Minsky N., "Rotating Storage Devices as Partially Associative Memories," Proc. 1972 FJCC.

[MISS81] Missikoff, M., "An Overview of the project DBMAC for a relational machine," Proceedings of the 6th Workshop on Computer Architecture for Non-Numeric Processing, June 1981.

[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.

[PARH72] Parhami, B., "A Highly Parallel Computing System for Information Retrieval," Proceedings of the Fall Joint Computer Conference, 1972.

[PARK71] Parker, J.L., "A Logic per Track Retrieval System," IFIP Congress, 1971.

[RODR76] Rodriguez-Rosell J., "Empirical Data Reference Behavior in Data Base Systems," Computer Vol. 9, No. 11, November 1976.

[SCHU79] Schuster, S.A., Nguyen, H.B., Ozkarahan, E.A., and K.C. Smith, "RAP.2 - An Associative Processor for Databases and its Applications," IEEE Transactions on Computers, C-28, No. 6, June 1979.

[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. TODS, Vol 1, No. 3, September 1976.

[STON79] Stonebraker, M. R., "MUFFIN: A Distributed Database Machine," University of California, Electronics Research Laboratory, Memo UCB/ERL m79/28, May, 1979.

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

[SHUL81] Shultz, R., "A Multiprocessor Computer Architecture for Database Support," Ph.D. Dissertation, Computer Sciences Department, Iowa State University, August, 1981.

[YAO81] Yao, B. and M.M. Tong, "Design of a Two-Dimensional Join Processor Array," Proceedings of the 6th Workshop on Computer Architecture for Non-Numeric Processing, June 1981.