

PERFORMANCE of TWO-DIMENSIONAL DATA MODELS for I/O LIMITED NON-NUMERIC APPLICATIONS

George G. Gorbatenko
Data Machine International
106 Wildwood Bay Drive
St. Paul, MN 55115 USA
gorby@ece.umn.edu

David J. Lilja
Dept. of Electrical and Computer Engineering
Minnesota Super Computing Institute
200 Union Street SE, Univ. of Minnesota
Minneapolis, MN 55455 USA
lilja@ece.umn.edu

Abstract

It was not too long ago that designers were looking to dedicated hardware to solve the “I/O problem”. However, despite the claimed improvements of such back-end processors, with few exceptions, they never reached commercial viability. With today’s exponential growth in data, there is growing concern that we are once again reaching a “memory wall” and there is talk of intelligent processing by I/O. While the basic I/O performance characteristics are governed by physics, we can adjust how we view, store, and process data so as to be more in consonance with the I/O devices themselves. Key factors are data granularity and achieving synchronous operation. This paper examines the performance issues of non-numeric applications that are primarily I/O bound. By using a two dimensional data model, logical topology of data is physically preserved on disk. This enables the disk to operate more in a synchronous fashion with granular data access. Using several benchmarks, we contrast the performance of the 2D model with a lineal model and find that improvements are on the order of 10-100X and roughly in proportion to the granularity of the data. By using intelligent controllers, each controlling a set of spindles, a high degree of parallelism can be achieved without the overhead usually associated with parallel architectures.

1.0 Introduction

Today, applications are being developed with little regard to the needs and capabilities of I/O. For cases where the same data is frequently accessed, the use of a cache mitigates relatively poor I/O performance. For the other cases, we have an “I/O problem” and look for creative ways to solve that problem. Indeed, the root cause of poor performance is not due to I/O per se, but rather due to a virtual disregard

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 28th VLDB Conference
Hong Kong, China 2002

of I/O by the applications. By adjusting how we view, store, and process data so as to be more in consonance with I/O devices, significant performance gains can be achieved.

In this paper we propose a new two-dimensional (2D) mapping scheme that preserves the logical topology of data. We introduce a figure-of-merit metric that facilitates comparing the I/O performance between the old way (lineal map) and that of the 2D case. Using a combination of analytical models and measurements on an actual hardware prototype, we show that the relative improvement is on the order of 10-100X and reasonably predicted by this new metric. While we focus primarily on non-numeric applications that are I/O constrained, many of the principles discussed here can be applied to more general applications.

The approach we take is to first look at a single disk and examine specifically the characteristics that effect performance. We introduce two metrics: access efficiency, ξ , and the utility of the data, ψ . The product of these two terms form a figure of merit (FOM), a convenient metric for measuring the “goodness” of I/O for the specific class of systems under study.

In an attempt to improve the FOM, we introduce a 2D mapping methodology that preserves the logical topology of data as it is mapped to disk. We examine several benchmarks and contrast the lineal mapping with the proposed 2D approach. We show that as an overall metric, the FOM is a reasonable predictor of performance.

Finally, the consequence of such a design reduces the traditional I/O bottleneck and offloads the host system. For maximum gain, we see the I/O subsystem as a computational partner. Subject to the even distribution of data, we assert that I/O performance will scale linearly with spindles and such architectures will become good candidates for our future high-performance systems.

2.0 I/O Coherence

It is a well known fact that maximum power is transferred when the load is matched to the source. We assert that the same phenomenon applies to architecture, namely that application (requestor) should be matched to the I/O (provider). It is this coherence, then, that holds the key to achieving high I/O performance.

2.1 Historical Perspective

Early computer systems typically use channel controllers to match the performance of higher speed processors with lower speed I/O such as punched cards, tapes, etc. Consider, for example, the IBM 3741 System, a key-to-diskette system driven by a modest (measured by today's standards) microprocessor [IBM73].

In a typical application, fixed length records are read in logical order and processed as shown in Figure 1. Note that the processing time exceeds the disk burst rate by a factor of 3. In the non-interlaced instance (case a), by the time the system wants to read the second record, it has passed the head. Thus, we incur a latency delay of a complete revolution, 167ms.

In the second instance, the disk was reformatted so as to interleave the logical records. In this case, the logical order of records (logical burst rate) exactly matched the processor's capability to consume the data, eliminating the effect of latency. In our example (and for a 26 sector disk) we realized nearly an order of magnitude gain by interleaving.

Note that this gain is predicated on the ability to anticipate the host's requests, a good understanding of the various times involved, and a consistent or coherent pattern of usage. Unfortunately, today's applications are characterized by exactly the opposite: very powerful hosts, many users, and often ad hoc operation. The operation is very random and certainly not coherent.

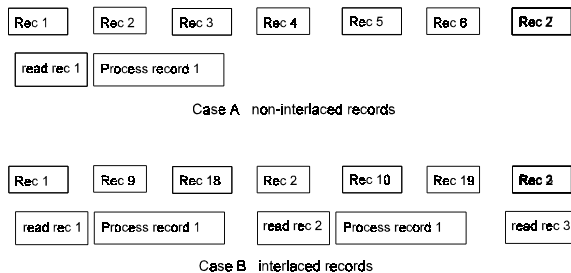


Figure 1 Interlacing records to match processor requirements eliminated the negative effect of latency.

2.2 Random Access

Compared to tapes, random access disks were a great improvement. As processors became more powerful, however, the penalty associated with random access became significant to the point that I/O now is traditionally viewed as the bottleneck of contemporary systems.

Because the magnetic disk is a physical medium, it is governed by physical laws. The seek time is the time required to position the head on a particular track. Some times the sought-after record will have just passed the head. Other times, it will just be coming up. On average, we will wait a half revolution to read the desired record. In a disk context, this time is defined as *latency*.

The sum of seek time and latency define the random access time. For contemporary disks this may be on the order of 10 ms. Once the head is positioned before the record, the data read time is accomplished at the burst rate

of the disk. The ratio of random access time to read time is large since we are dealing with mechanical time as opposed to electronic time, perhaps 2 orders of magnitude.

2.3 Challenge

The challenge before us is to define an I/O architecture that will exhibit the coherence of our simple example in an environment that is basically chaotic. In the example, the only random access was to position the head before the start of the record string. Since we were dealing with a long record string, the inefficient access is effectively amortized over the large data block. Finally, the processor was able to keep up with the logical burst rate so that the next data sought for processing was just coming underneath the head.

3.0 Systematic Applications & Locality

Exploiting locality is a key factor in achieving high performance I/O. Recalling the key-to-diskette system studied earlier, each record predictably followed the previous record. After the initial positioning of the disk to the start of the record sequence, there were no more random accesses since the next data was always underneath the head.

We will define a set of applications where some form of locality of reference applies. This could be reading one record after the next, processing an index, anticipatory reads, etc. We will refer to such applications as *systematic*. For systematic applications, there exists a logical neighborhood where the next data sought is a member. Thus, the logical neighborhood becomes the predictor of the next data we want.

3.1 Logical vs Physical Neighborhood

Consider the example in Fig 2a. Here we have shown a two dimensional table where our point of reference is C3. Its neighbors are shown shaded. Assuming we read from top to bottom and left to right, the next data element we want would be either D3, if we were moving from left to right, or C4 if we were moving in the orthogonal direction. In other words, the next data we want is in a logical neighborhood..

Given that we treat the disk as lineal storage (block 0 to block n-1) we have a choice as to how we map this logical data: row or column order. If we map it by row, we maintain our horizontal contiguity but loose in vertical direction (case b). Alternatively, mapping by column gives us good locality in the vertical direction at the expense of horizontal contiguity (case c). In either case, we have failed to retain the neighbors of C3, thus the topology is not preserved .

The significance of preserving topology is that, to the extent that locality of reference applies, the next data the system will want will be close by. However, it is axiomatic that to preserve the neighborhood when mapping from logical to physical space, the physical space must be at least the same cardinality as the logical data space. Thus, for a two dimensional table, we need a two dimensional disk.

A1	B1	C1	D1	E1
A2	B2	C2	D2	E2
A3	B3	C3	D3	E3
A4	B4	C4	D4	E4
A5	B5	C5	D5	E5

(a) Two dimensional table. The cells adjacent to C3 (shown shaded) belong in a logical neighborhood of C3

A1 B1 C1 D1 E1 A2 B2 C2 D2 E2 A3 B3 C3 D3

(b) Row ordered mapping. Only the horizontally adjacent cells remain close to C3.

A1 A2 A3 A4 A5 B1 B2 B3 B4 B5 C1 C2 C3 C5

(c) Column ordered mapping. Only the vertically adjacent cells remain close to C3

Fig 2 Given a two dimensional table (a), the data can be mapped to disk either by rows (b) or by columns (c). In either case, logical topology is not preserved.

3.2 Disk as a Two Dimensional Medium

While disk is currently treated as a large lineal medium (block 0 to block n-1), it is in fact three dimensional. It has many concentric cylinders, each of which has as many tracks as there are data heads. Within each track there are sectors (or blocks in SCSI terminology). Thus, any sector can be physically identified by the triplet: cylinder-head-sector.

3.2.1 Cylinder map

While in the abstract, it makes no difference which two of the three dimensions we choose to act as the basic 2D space, there is more temporal affinity between tracks on a cylinder and sectors within a track than between adjacent cylinders. Thus, we view the disk cylinder as the basic unit of 2D space. Within that space, then, we have heads (tracks) in one direction and sectors in the orthogonal direction.

In Figure 3, we have taken a single cylinder and unwrapped it, laying it flat in the plane of the paper. As indicated, this cylinder has m heads and n sectors (blocks) per track. We also indicate a skew of 2 sectors.¹

3.2.2 Navigation

Per the SCSI standard, each physical block is assigned a logical block number (LBN). Starting with LBN 0, there are n logical blocks on the first track. From the last block on the first track (LBN n-1), the first block we can read on the

adjacent track is the third physical sector. This block is assigned LBN n. Note that as we move down from head to head, there is a skew of 2. This skew is inherent in the drive and is required to accommodate embedded servo information. Each track will be skewed from the previous track by a like amount.

Starting from LBN 0, we can move in a horizontal direction by simply reading along the track (LBN 0, LBN 1, ..., LBN n-1). This is a track read. To move in the orthogonal direction, we would read diagonally across the heads (LBN 0, LBN n+1, LBN 2n+2, ..., LBN (m-1)(n+1)). We refer to such a read as a sector block read.

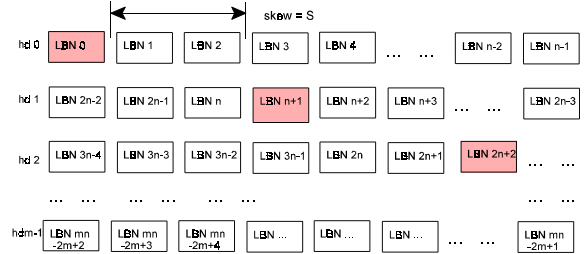


Fig 3 Physical block layout for a cylinder having m heads and n sectors per track for a total of mn blocks. For illustrative purposes, a skew of two is shown. We assume also that the starting block number of the cylinder is LBN 0.

To summarize, we have two modes of navigation in our 2D space. Track reads and sector block reads. There are as many sector blocks in a cylinder as there are sectors in a track. Since we are reading a block from each track, we refer to the diagonal set as a sector block. In our example the cylinder consists of n sector blocks. The first three sectors that comprise Sector Block 0 are shown shaded.

4.0 Figure of Merit (FOM)

We define the FOM as the product of two terms. The first term deals with how we amortize the inefficient random access by reading large blocks. The second measures how much of the data we read is pertinent to the computation.

4.1 Access Efficiency

We define the efficiency of the data access, ξ , as the ratio of time we reading data over the sum of data and access time.

$$\xi = \text{data time} / (\text{data time} + \text{rand access time}) \quad (1)$$

Since random access is unavoidable, the best we can do is to amortize that fixed time over a large data read. Reading an entire cylinder of data would preempt the I/O for perhaps 100-200 ms which isn't permissible for a multi-user system. On the other hand, choosing to read a single block takes much less time but is very inefficient. The throughput vs block size relationship is shown in Fig 4.

¹ Skew is defined as the number of sectors that slip by before the first sector on an adjacent track can be read.

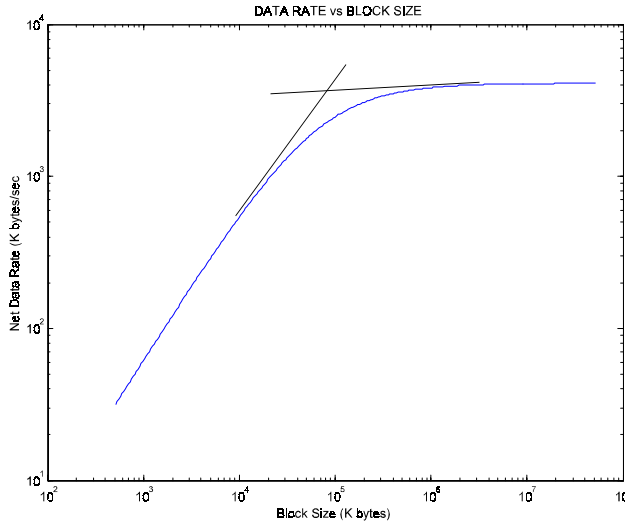


Fig 4 Average rate of record access vs block size. Note that the "half power point" occurs at a block size that corresponds to a track.

As indicated in the figure, a reasonable compromise is realized between the two extremes at the intersection of the two slopes as shown in the figure. This "half power" point roughly coincides with a track's worth of data, a conclusion that escaped neither disk manufacturers who offer anticipatory track reads [Seag95] nor researchers who align data along track boundaries [Schi02].

4.2 Read Utility

While large data reads make the access efficiency look good, there is merit to such a read only if there is utility to the data. For example, if from the track read we only wanted a single record, then the utility of the read would be low. To measure the quality of the read, we define the metric, data utility, Ψ , to be

$$\Psi = \text{min data needed} / \text{amount of data read.} \quad (2)$$

As an illustrative example, consider an SQL query of the form

```
SELECT name, address, salary FROM employee
WHERE salary > $20K
```

operating on the small database described in Figure 5. In this case, only three fields are requested (as opposed to the entire employee record). In SQL terms, this operation is referred to as a *projection* [Date82] and is an important notion that we will return to later.

In a conventional system where records are mapped one record after the next in a head-to-tail fashion, we would have to read all of the data in order to complete the query. Yet, only the shaded portion was needed. The percentage that the shaded fields represent compared to the whole, then, constitutes the data utility, Ψ .

name	addr	city	state	zip	emp	sal	doh
gorby	106 wild	St Paul	MN	55115	636	10K	1961
chas	1218 first	Hudson	WI	54016	123	8K	1980
timbo	434 Polar	Plymoth	MN	55123	987	11K	1982
maria	208 Mull	Pleasant	CA	93010	997	20K	1960
susan	280 Mead	Reno	NV	89509	654	15K	1965
galina	336 Brod	San Fran	CA	94012	2341	24K	1975

Fig 5 An example database for an SQL query. Data space consists of rows and columns. The shaded area represents the minimum amount of data required to satisfy the query.

Finally, we define the overall "goodness" of the I/O subsystem as the product of the two. We refer to this term as the Figure of Merit (FOM).

$$\text{FOM} = \xi * \Psi \quad (3)$$

In the earlier key-to-disk example, the FOM could be simply calculated and found to be close to unity, a near perfect value. In conventional systems, this value could be less by perhaps several orders of magnitude. As we will show later, the FOM is a reasonable predictor of I/O performance and that improvements in the FOM in an I/O constrained application are nearly directly proportional to the performance gain.

5.0 Mapping of Records

In Fig 6, we show a "straightened out" version of the shaded diagonal earlier shown in Fig 2. Since we are mapping a two dimensional space (rows and columns) to a sector block, we have a choice of whether to map rows across the heads, leaving fields to be mapped along the tracks, or vice-versa.

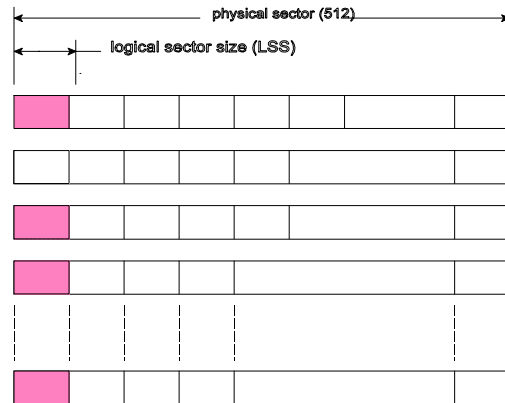


Fig 6 A sector block consists of num_hds sectors, one from each track and taken in a diagonal pattern. A record space is defined by the area $LSS \times num_hds$ and is shown shaded in gray.

5.1 Logical Sector Size (LSS)

Given that database tables are much longer than they are wide, we choose to map the records across the heads, a fixed portion under each. Accordingly, we subdivide the physical sector into a set of smaller logical sectors. This defines a record space that is $lss \times num_hds$ in area where the logical sector size, lss , is defined as

$$LSS \equiv \lceil rec_len / num_hds \rceil = 4, 8, \dots, 4n. \quad (4)$$

and rec_len represents the length of the record. The LSS is an important construct, and is a key value when mapping from lineal to two dimensional space. All cylinders associated with a particular table must have the same LSS value. It will be shown later that if the tables span spindles, then all spindles so spanned must be of comparable geometry i.e., the same number of heads in order for the LSS to be constant.

Finally, we round the LSS value up to be some multiple of 4 to accommodate the data path width associated with disks and their controllers.²

5.2 Mapping Records to Sector Blocks

Consider a 64 byte record type defined as indicated in Table 1. Assuming 10 heads, the LSS can be determined to be

$$lss = \lceil rec_len / num_hds \rceil = 64/10 = \lceil 6.4 \rceil \Rightarrow 8. \quad (5)$$

Thus, the record space is defined as...

$$rec_space = lss \times num_hds = 80 \text{ bytes}. \quad (6)$$

and is depicted in Figure 7

```

typedef      struct  _record
{
    char      employee_no [8]  // field A
    char      name [12];      // field B
    char      address [24];    // field C
    char      zip [5];        // field D
    char      salary [6];     // field E
    char      doh [6];        // field F
    char      dept [3];       // field G
} Record;
    
```

Table 1 Definition of Record Structure

² Contemporary disks and their controllers have a minimum data path width of 4 bytes. Given that we are paying for 4 bytes whether we need it or not, we round the LSS value up to the next higher multiple of 4.

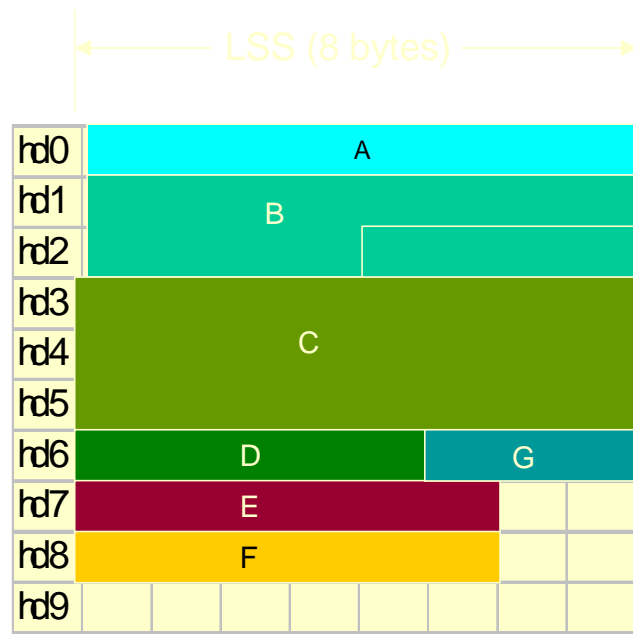


Fig 7 Fields are assigned to record space using the modified best fit algorithm. Note that variables may be assigned to fragmented space as long as a LSS boundary is not crossed unnecessarily.

5.3 Modified Best Fit (MBF) Algorithm

In Fig 7, we depict the two dimensional space that is to contain the 64 byte record. Because the LSS algorithm rounds up, we are assured that there is sufficient space: $lss \times num_hds$ or 80 bytes. We refer to the placement algorithm as “modified” because we attempt to avoid crossing unnecessarily an LSS boundary. Where possible, then, fields will be placed at the start of a new space or will be wholly contained within a space. As will become apparent, for every instance where this is not the case, an extra spin penalty will be incurred.

For the example depicted in Fig 7, Field A (*employee_no*) is 8 bytes and exactly fits in the space defined by head 0. Field B (*name*) is 12 bytes and will cross the boundary; the first part will be under head 1, the balance under head 2. Because the field is 12 bytes, it is impossible to avoid crossing a boundary as is the case with Field C. Fields D and G are able to share the same space. In this fashion, the fields are placed in the sector block.

As is evident from Fig 7, the LSS algorithm yields ample space to map a record and typically without unnecessarily crossing an LSS boundary. While a more aggressive algorithm may achieve higher density, it will likely require more time (spins) for processing.

5.4 Variable Length Records

So far, all the record fields have been a fixed length. In general, variable records (records having variable length fields) tend to detract from the orderly and predictable mapping of records. To preserve the regularity and coherence of our data, variable fields are divided into two parts: a fixed length part that shares the same record space along with the other fields, and a variable part.

A user would specify the fixed part to be as large as

desired. This field would include a 2 byte pointer to a heap at the end of the cylinder which would contain the variable part. The variable part could be arbitrarily long.

It should be noted that only the fixed portion of the field will be scanned. Thus, variable fields involved in the selection criterion will identify candidate records for selection. Those candidates then will be qualified by examining the heap as part of a post-processing operation in the host system.

6.0 Anatomy of an SQL Query

6.1 Problem Decomposition

To quantify the advantage of the proposed 2D mapping, consider two simple queries operating on a cylinder's worth of records as defined in Table 1. To make the analysis simple, we assume no involvement of indexes and operations are restricted to a single table. Admittedly, this is a simple example by SQL standards but will serve to illustrate the difference between the conventional lineal map and the proposed 2D map.

- Q1 Select employee_no and zip_code from Table where zip_code= ZIP
- Q2 Select name, address, and salary from Table where salary > \$50K

In the lineal case, all the records are examined (an exhaustive read). We will assume a large track read (or that the disk drive has an anticipatory track read). Those records that satisfy the predicate are retained.

In the 2D case, we divide the operation into two steps: a select followed by an extract. The select will identify the records of interest by scanning the fields involved in the expression. For Q1, this would be the track corresponding to hd 6 in Fig 7. For Q2, it would be the track corresponding to hd7. Once the ordinal position of the records are identified, we extract only those sector blocks.

The number of spins required to extract the selected sector blocks is based on many factors- the number and relative position of the records, the state of the track cache in the I/O processor (IOP), and the exact fields to be extracted. However, in all cases, the total number of spins is bounded by the sum of the tracks involved in either operation. Since this constitutes the worst case, we will assume those times.

6.2 Results

Given the data map defined in Fig 7, we can analytically determine the run times. In addition to computing the times, we calculate the efficiency, data utility, and the FOM of all the cases. Of particular interest is the relative analysis between times and the FOM. The ratio of time (lineal/2D) is seen to be 5.8 as is the ratio of FOMs (2D/lineal).

7.0 Model Validation

The underlying assumption with the 2D mapping is that

Test Runs	Q1	Q2
<u>Lineal Statistics</u>		
efficiency	.36	.36
data utility	.20	.66
fig of merit (FOM)	.07	.24
per cyl time (ms)	230.40	230.40
<u>2D Statistics</u>		
per cyl time (ms)	31.71	66.37
packing factor	0.80	0.80
adjusted time	39.63	82.96
wgt avg FOM	.43	.66
Ratio Analysis		
FOM (2D/lineal)	5.81	2.78

Fig 8 Performance comparison : Lineal vs 2D for Q1 and Q2 query examples.

once on the cylinder, we are able to reliably determine where the head is relative to the cylinder. By making operations atomic to the cylinder, we insure only one random access for the set of records contained therein. Within the cylinder, operations typically start and end at the same point, thereby maintaining a large degree of synchronism. Indeed, the only random operation associated with the cylinder is the initial random access.

7.1 Introduction

To verify our model, we implemented a prototype model. We wrote a cylinder's worth of data on a dedicated raw disk and then proceeded to scan the cylinder for records using a selection.. We compared observed time to predicted time. Several read and select operations were performed. Times were wall clock measured at the application program interface (API). For operations taking more than 60 ms, a context switch occurred with 10 ms of overhead. The high level block diagram of the prototype is shown in Fig 9.

7.2 Prototype Hardware

The model was implemented in C using Solaris 2.5. Two 4GB Segate Barracuda [Sega95] drives are dedicated to the data under study. A third disk is used as a general purpose system disk. These particular disks have 29 zones. The IOP was emulated in software and was connected to a dedicated Adaptec SCSI controller.

7.3 LBN Map

To facilitate mapping, the disk was reformatted so that each track had 2 spare sectors. Using SCSI commands, the starting LBN in each zone was noted. With this and the number of sectors in each zone, we are able to obtain a complete logical to physical map directly. Excluding the reformatting time, the entire mapping process was accomplished in the time required to execute several SCSI commands.

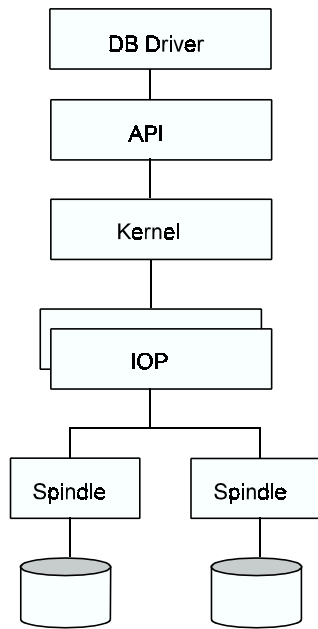


Fig 9 Block diagram of the prototype

7.4 Record Particulars

A record space of 168 bytes was defined based on an LSS value of 8 bytes and 21 heads. Because 512 is evenly divisible by 8, we preempted the last record spot (for potentially storing transaction IDs) to net 63 records per sector block. The first 123 sector blocks contained records for a total of 7,749 records per cylinder. The last (124th) sector block was defined as the heap storage.

7.5 IOP Optimizer

Given a bit vector, which is the result of some scan operation, the optimizer determines the number of spins required to extract the data as sector blocks. This will depend on distribution of records, the skew, and the particulars of a project. Alternatively, the fields involved in the extraction that are not yet in cache may be read as tracks (one spin/track). The lesser of the two dictates the outcome, with ties favoring tracks. Thus, taking all the factors into account – the vector, disk skew, the fields of interest (project), columns in cache, and available space – the optimizer will choose the minimum time option.

7.5 Test Runs

The following 4 operations were performed:

- (a) write a cylinder's worth of data with Optimizer disabled. (124 sector blocks; 2604 I/O ops)
- (b) write the same amount with the Optimizer enabled. (124 sector blocks; 21 I/O ops)
- (c) scan the cylinder involving 3 columns; extract 2 sector blocks
- (d) repeat operation (c).

7.6 Discussion of Results

In the first case, we disabled the optimizer. We wrote 124 sector blocks, each comprised of 21 sectors. For a total of 2604 I/O ops. For the second case, we repeated the operation but enabled the optimizer. Enabled, the optimizer choose to write the cylinder as tracks: hence 21 I/O operations. For case (c), we selected two entire records from disjoint sector blocks. The selection took two spins and 4 more for extraction. Finally, we repeated the case, but since the track data was in cache, only the extraction incurred I/O time: 4 spins. This data is summarized in the table and reflects an adjustment due to context switch time; latency of ± 4 ms applies to all the times. Overall, there is reasonable agreement between observed and predicted times. Thus the model appears valid.

	<u>Observed</u>	<u>Calculated</u>
case (a)	2.5 sec	2.427 sec
case (b)	196 ms	216 ± 4 ms
case (c)	51 ms	54.5 ± 4 ms
case (d)	42 ms	37.6 ± 4 ms

8.0 Benchmark Runs

Typically, benchmarks have a personality that could favor one system over another. For our tests, we choose three benchmarks, each perhaps with its unique bias. However, taken together the set should offer a balanced picture of a "typical" set of I/O operations.

Also, we avoided using cases that involved table joins. We argue that such cases are more dependent on host based optimizers and processing power, not I/O. In Section 10 we will address the use of indexes and briefly touch on joins..

8.1 Procedure

Following the pattern established earlier, we proceed as follows:

- for a given record length, define LSS
- compute the number of records per cylinder, and thus the total number of cylinders required
- map the record fields in the order they are given in the benchmark
- determine time to process all cylinders

Since the reported results lack the detail necessary to determine the FOM, we worked backwards so as to come up with a best guess estimate. While somewhat error prone, it gave us a metric other than time to judge the results.

8.2 Assumptions

Since the IOP outcome of the optimizer is not deterministic, we assumed that it would choose to read the records by track (as opposed to by sector blocks). We assumed that the reported times were largely dictated by I/O with minimal system overhead. This was in keeping with the assumption that we are dealing with I/O limited applications. The

narrative accompanying the benchmark results seemed to reflect this as well. Runs were single user.

Times were based on the Seagate Barracuda disk used in the prototype. [Sega95]. For the 2D case, indexes were used when specified and required to enforce unique fields. For all other cases, exhaustive reads were used.

8.3 Benchmark Description

Three benchmarks were selected:

- The Wisconsin [Bitt83]
- The Set Query [Oneil91]
- TPC D/H [Raab94]

8.3.1 Wisconsin

We chose the 5 queries for which results were reported (and that were not joins) as reasonable representatives of an I/O limited case. Selections of 1% and 10% were defined. Also, the benchmark called for the entire record to be extracted. The cited results were based on a 1-disk version of the Gamma machine [DeWi90]. The times were reduced to 58% of the reported values to compensate for the slower disk technology. Because the 8Kblock represented almost a track read, we did not run the full track case.

8.3.2 Set Query

In contrast to the Wisconsin Benchmark, this one emphasized counts of record instances with very little or no field/record extraction. As a result, many of the queries were satisfied with pre-built indexes, allowed by the benchmark and obviating the need for I/O operations. The same query would be run against different population densities³ (ranging from most selective to least selective). As more records were selected, the advantage of the index-based optimizer strategy would be expected to wane. Two databases (DB2 and M204) are compared to the 2D case.

8.3.3 TPC D/H

Of all the benchmarks, this one is probably the most representative of contemporary relational database applications. We choose Q6 (the only non-join application defined). Because the reported results are normalized to performance per dollar cost (over a 5 year period including maintenance), we did not have a time to compare to. Instead, we modeled the query for the lineal case assuming first 8K blocks, then full track reads.

9.0 Experimental Results⁴

³ For example, a field that had one hundred unique items would be labeled "K100". A field representing gender would be designated as "K2".

⁴In all cases we assume a disk with 21 heads (over which we mapped the record)

For each benchmark, we discuss the particular strategy used, we summarize the 2D times, we compare the 2D times to the reported data, and finally discuss the results.

9.1 Wisconsin Benchmark

Map Detail

To hold a 208 byte record, we require an LSS = 12 bytes. This defines a 252 byte record space with a packing overhead of 21%. We get 42 records per Sector Block and 5,376 records per cylinder. To store the required 1M records we require 186 cylinders. Because the benchmark assumes a single user, the optimum cylinder-to-cylinder access time can be assumed when we do not incur an intervening index operation.

Strategy

The queries call for a simple select and then extracting the entire record. The order of the operations was as defined by the query statement. On the first two index operations, we used an index (even though not doing so would have improved results slightly). On the last query, we proceeded without using the index and produced a slight improvement.

For each query, we broke down the times spent on selection as well as extraction. In each category, we further indicated the I/O efficiency as well as the utility of the data access. Finally, we noted the number of I/O disturbances associated with each operation. An I/O disturbance is any discontinuity such as a random access (page fault) or a latency delay (hit). Seek time when moving to an adjacent cylinder is a planned synchronous operation and not viewed as a discontinuity.

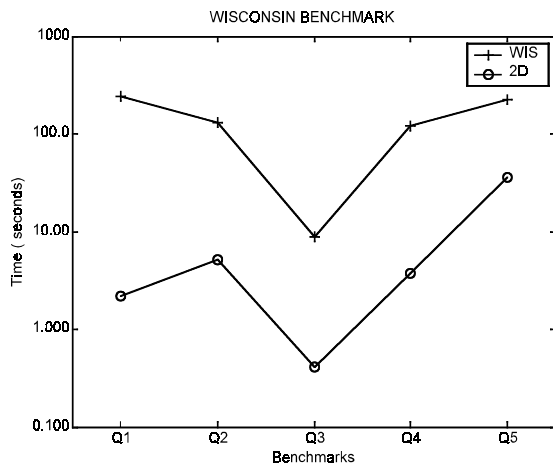


Fig 10 Wisconsin Benchmark times for both the 2D case and the reported system case. Note that on average, the 2D runs are about 15 times lower which reflect the improved granularity of record access (5% vs 100%). See Section 10.1.

Discussion

Central to our research is that we are attempting to define a coherent sequence of I/O operations since randomness (i.e., the lack of coherence) is the root cause of inefficiency. We argued that if the FOM was high, the I/O was being effectively used and the resulting time should be low. Initially, we thought that a time*FOM product might be a useful metric. In certain cases this gave us misleading results.⁵ A more predicible approach is to a relative analysis.

If indeed the figure of merit is a predictor of performance, then we would expect that the relative times should result in the same ratio i.e., if we are twice as efficient, we should be twice as fast. This information is shown in the Table 2 below where we compare the ratio of lineal time / 2D time and compare it to the FOM ratio. .

Benchmark	1	2	3	4	5
Ratio Times	110.8	25.4	21.1	32.9	6.2
Ratio FOM	46.0	21.9	3.2	3.5	2.2
Delta ratio	.416	.860	.151	.105	.351

Table 2 The ratio of relative times and FOMs for the 2D and base cases.

9.2 Set Query Benchmark

Map Detail

To hold a 200 byte record, we again require an LSS = 12 bytes. This defines a 252 byte record space with a packing overhead of 26%. We get 42 records per Sector Block and 5,376 records per cylinder. To store the required 1M records we require 186 cylinders. Again, the single user assumption insures that we can achieve an optimum cylinder-to-cylinder access time.

Strategy

Because the column values in the SEQ field are unique, we built an index to enforce the uniqueness. For the remaining fields, no index was assumed.⁶ Except for operations involving the SEQ field, we used exhaustive reads of tracks. Aside from the index-related scans, the times were generally

⁵ A high FOM case may have long time associated with it due not to the nature of the operation (and not inefficiency) and could be cast in a poorer light compared to a less efficient operation that took a shorter time.

⁶This is stark contrast to the reference databases which built an index on all the fields as well stored statistics to guide the host-based optimizer. Optimizer overhead ranged from 33% to 56% (DB2)

constant and corresponded to 1-4 spins of the disk. All runs assumed the 12 byte LSS. This was a bit coarse as typically only one of the three fields under any head was needed. In this case, we are operating at a data utility of around $\psi = 33\%$.

For each class, there are a number of runs. In the 2D case, there was no discernable difference between “A” and “B” versions, thus only “A” was reported. Finally, Q6 was not run. This was a join operation where the issue was the host-based CPU power, not I/O. For this reason, an “apples-to-apples” comparison could not be practically made.

Test Results

The comparative times are shown in the figures below. Note that because the times vary significantly within the class, they are shown plotted on a logarithmic scale. The exhaustive case is constant for all runs and is based on full track reads; the time is 71.4 seconds and is shown as the dotted line in the plots.

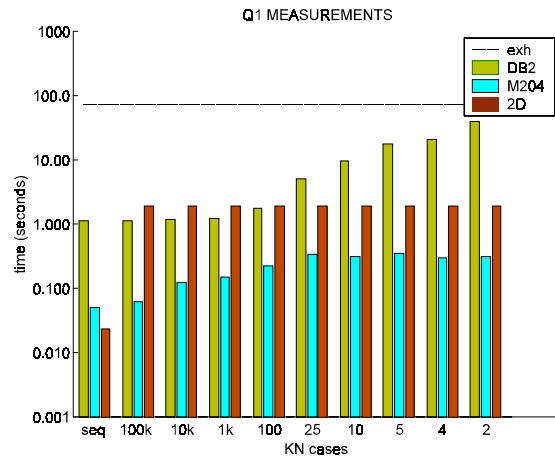


Fig 11-Q1 Count the record instances that satisfy simple query. Note that the 2D times are flat as is the exhaustive case. The DB2 times increase as the selectivity is reduced

Analysis

On the surface, the benchmark data would suggest that index based-schemes are superior to conventional schemes. Certainly in this particular benchmark, the index-based schemes seem to offer a distinct advantage over the exhaustive case. Even for the 2D map, the index-based approach offers an advantage. To understand why, we should better understand indexes, this particular benchmark, as well as the data we observed.

The purpose of an index is to circumvent inefficient I/O operations. Indeed, in the narrative associated with the benchmark, the optimizers of the commercial databases structured strategies that allowed solving the problems by accessing pre-stored information such as metadata and indexes.

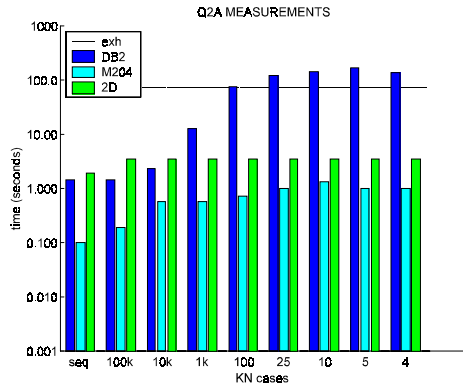


Fig 11-Q2A Extract count or records meeting simple criterion. Note that the DB2 seriously degrades with the loss of selectivity. Note the 2 orders of magnitude delta between DB2 and M204 (misguided optimizer)

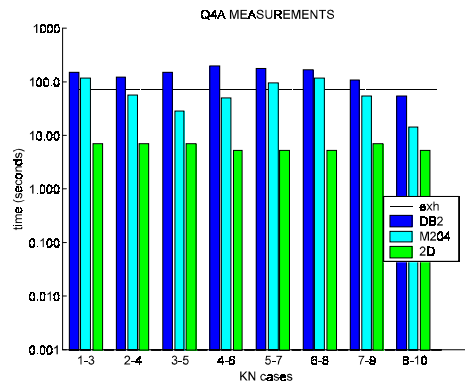


Fig 11-6.Q4 Q4A. Here we are extracting two fields based on a compound expression involving three conditions. The 2D case is about 10X better..

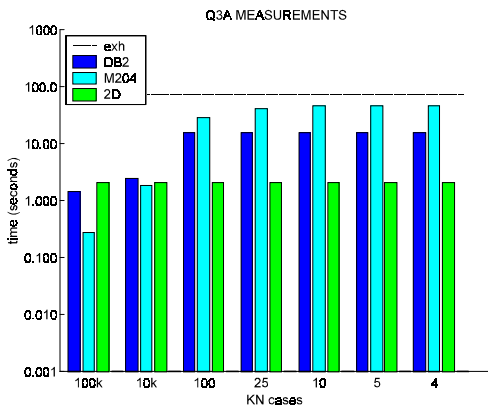


Fig 11-Q3A. For this run, we are extracting one field. Thus, the index approaches are beginning to falter starting at a selectivity of 1%. The 2D approach is constant.

In a properly designed index, all but the leaf will be in memory. Thus, for each data element we encounter, we would expect to fault on the leaf and again at the record pointed to by the leaf. If we had 20 siblings that shared the same physical block, then on average we would have 1.05 faults per data instance.

In those cases where the query requested a count (as opposed to a field value), we eliminated the record fault. The net was .05 faults per data instance. As the selectivity of the query decreased (referring to the plots, moving from left to right), we had more data items to contend with and therefore more random accesses and poorer performance.

Finally, displaying data in logarithmic fashion can be deceptive. Variations on the left side tend to be clustered around 1 sec. On the right, we are dealing with tens and hundreds of seconds. Yet, they look about the same. This point is illustrated in Fig 12 where we took the average of all the runs within each query group and plotted them on a linear plot. In this case, the 2D is shown to be significantly better than the index-based schemes by 1-2 orders of magnitude

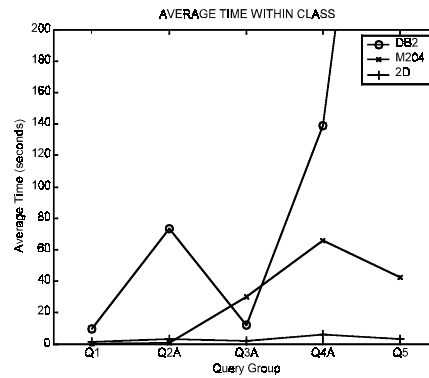


Fig 12 Average of all runs within group vs time

9.3 TPC D/H Q6

Map Detail

The LINEITEM table is defined to hold 600K records. Each record is 144 bytes. With a LSS = 8 bytes, we define a record space that is 168 bytes long. Thus, we have 64 records per sector block or a total of 8,192 records per cylinder. To store the table, we require 74 cylinders.

Strategy

Based on the record map, heads 2 and 4 provide the information for discrimination. In this case, unless the bit vector is null, the optimizer will chose the exhaustive option (since the DISCOUNT field was already in cache) and direct the system to read the data beneath head three for the remaining output field (EXTENDEDPRICE) Thus, 3 revolutions are sufficient to process all the data on any particular cylinder.

Test Description and Results

Four runs were made. In the lineal case, we assumed an 8K

block as well as the more advantageous track read (64K). The 2D runs had LSS values of 8 and 4. In Table 3 we show the times, FOMs, and the various ratios. Note the lineal case was significantly favored by the larger block size. The 2D performance gain was roughly 10-100X.

LINEAL				
block size	8,192	65,536		
time (sec)	155	19.38	8.00	ratio
FOM	0.009	0.05	5.47	ratio
2D				
lss (bytes)	8	4		
time (sec)	1.95	1.59	1.23	ratio
FOM	0.454	0.639	1.41	ratio
FOM ratio	49.38	12.71		
time ratio	79.52	12.22		

Table 3 TPC Q6 Summary: lineal vs 2D

Ratio Analysis

Since we derived both the lineal and 2D times analytically, we have a reasonably accurate idea of the FOM in both cases. Comparisons were made horizontally (within lineal or 2D) or vertically (lineal vs 2D). The ratios are shown shaded in the table and correlate reasonably well to each other, ranging from $\pm 2\%$ to $\pm 23\%$.

10.0 Related Work

While we earlier cited the IBM 3740 system, there have been many variations on the basic theme.

Slotnick [Slot70] is credited with the first proposal of a processor-per-track (PPT) scheme, however there were many others that followed. More practical were the processor-per-head (PPH) schemes [DeWit82] that used "intelligent filters" between the disk and the host. Unfortunately, these schemes involved custom hardware that rendered them expensive which restricted their use to the government and research labs.

Starting around the late 70's, there was a flurry of activity on stand-alone systems dedicated to processing "non-numeric" data. These came in different forms, but all attempted to off-load the host machine [Cham80] [Lowe79] [Bray79] [Lang 76] [Dewitt82] [Boral82] [Mall80].

Database machines like Teradata's DBC/1012 (up to 1024 386 processors), Britton-Lee DBM 500, and ICL's CAFS in time deferred to more cost-effective solutions using software running on fast generic platforms⁷ [Haw87].

About the same time Intel offered a chip set to accomplish the same thing. One of the problems with these machines was the lack of a standard upstream interface

(though Teradata eventually supplied a SQL interface). Probably more important, database software running on the latest (generic) platform would cost less and typically deliver comparable (or better) performance than dedicated custom hardware that was built using earlier technology.

In the last few years, we have seen a renewed interest in the use of back-end intelligence to improve the I/O performance of today's systems. Proponents of such systems 20 years ago saw the same thing that current researchers see, namely, that it is often more efficient to bring the processing to the data than the data to the processing [Keet97] [Acha98] [Ried97] [Ried01]. The emphasis here, however, appears to be the exploitation of disk intelligence rather than placing it in the I/O controller as is suggested in this paper.

Schindler et al [Schi02] cited improvements in disk efficiency of up to 50% by aligning data to the physical track boundaries due to reduced latency and track-to-track crossings. Clearly, if one wants to extract performance from I/O understanding the physical makeup is the first step.

11.0 Discussion and Concluding Remarks

We have examined several benchmarks. On average superior results were obtained with the 2D approach. These gains were in part due to improved granularity and in part due to synchronous operation. Specific cases where the 2D approach failed to improve results were index based.

11.1 Granularity

Given that we are mapping the record across the heads, we are able to access a particular portion that happens to fall under a particular head. With 21 heads, our granularity is one part in twenty one or roughly 5%. If the query only involved several fields, we would read only those tracks as opposed to reading all 21 for an exhaustive read, an order of magnitude gain. If on the other hand, the query involved many fields, the advantage of granular access would wane.

11.2 Synchronous Access

By defining operations to be atomic to the cylinder, we are able to amortize the random access to (at least) a track's worth of data. In the 2D case, we required only one random access. Once at the cylinder, operations were synchronous. The lineal case had 21 random accesses. Using the prototype numbers, this amounts to around 300-400 ms of needless overhead for each cylinder.

11.3 Index Operations

Note that in the Q1 set of the Set Query benchmark, the M204 results were consistently better than either the 2D case or DB2. As the query was defined, the answer could be obtained without accessing the record data-only the index (which was conveniently pre-built).

While clearly being able to accomplish an operation in logarithmic time will always be superior to accomplishing the same operation in linear time, there are certain inefficiencies that will detract from the coherence we are attempting to foster.

In Figure 13, we contrast an exhaustive read to an

⁷Teradata continues to offer fast (and expensive) solutions.

index based read. The index cases have times that are proportional to the extraction rate and are thus shown as diagonals. The 2D cases (for various skew conditions) are shown as horizontal. For extractions less than the crossover point, use an index. Otherwise, use the exhaustive scan. If we are not sure, opt for the exhaustive scan since the penalty for a wrong guess is less (the area of larger wedge is 7 orders of magnitude more than the small: 7 hours vs 3 seconds).

In summary, we can conclude with the following: unless a field is known to be unique, assume the exhaustive option, even if an index is provided.⁸

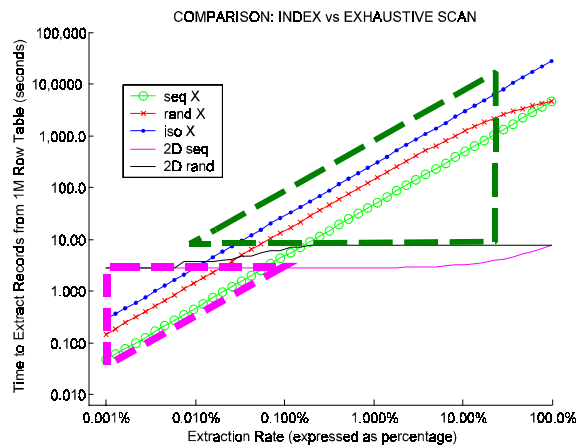


Fig 13 Comparison between an exhaustive read and an index based read for three classes of indexes.

11.4 Summary and Future Work

We have examined I/O constrained applications with the idea of improving performance. Conventionally, operating systems tend to obscure the physical attributes of the disk and all but guarantee randomness... the primary cause of poor I/O performance.

By treating the disk as a two dimensional medium, we are able to preserve the logical topology when mapping data to disk. For systematic processing, the next data sought will be close at hand, thus, eliminating random access. By mapping records across heads within a cylinder, we achieve a high degree of granularity. The combined effect is an overall efficiency that approaches the ideal with improvements on the order of 10-100X.

Byproducts of this improved efficiency is the need to process the data in the controller. The need for the IOP to be computational partner is particularly important if we wish to scale performance with additional spindles. Because of the granularity, the dependence on indexes is reduced. Unless the particular field is known to be unique, its use may be counter productive.

For the proposed mapping to be effective, we have to change a few paradigms. In the context of 2D mapping, then, topics for future work include:

- Expand the application areas to include scientific and spatial databases.
- Methods for operating systems to reveal rather than obscure disk detail.
- Application techniques for passing computational chores to the I/O subsystem for processing.
- Techniques by disk OEMs to enhance or facilitate the computational role of disk systems.
- Expansive system measurements that include more detail relative to I/O operations.
- Methods of balancing loads across spindles so as to achieving high degrees of parallelism as a viable alternative to classical MPP designs.

In short, the disk is not the problem; it is how we use it.

References

[Acha98] Acharya, Anurag, Mustafa Uysal and Joel Saltz, "Active Disks", Technical Report TRCS98-06, Univ of California, Santa Barbara, 3/98.

[Bitt83] Bitton, D., D.J. Dewitt, C.Turbyfill, "Benchmarking Database Systems, a Systematic Approach", *Proc of the International Conference on VLDB*, Florence, Italy 1983

[Boral82] Boral, H. et al, "Implementation of the Database Machine DIRECT", *IEEE Trans on Software Eng*, Vol SE-8, no 6, Nov 1982.

[Bray79] Bray, O., and Ken Thurber, "Whats Happening in Data Base Processors?", *Datamation*, 1/79 .

[Cham80] Champine, G. A., " Back-End Technological Trends", *IEEE Computer*, Feb 1980.

[Date82] Date, C.J., "An Introduction to Database Systems", Addison-Wesley Publishing Company, Menlo Park, CA 1982

[DeWi82] Dewitt, D. and D Friedland, " Exploiting Parallelism for Perf Enhancement of Non-Numeric Applications", *Nat'l Comp Conf 82*

[DeWi90] DeWitt, Ghandeharizadeh, and Schneider, "The Gamma Machine Project 1990", *IEEE Trans Know & Data Eng*, 1990

[Haw87] Hawthorn, P., "Optimization Techniques Improve Performance of Relational Databases", *Computer Tech. Review*, Summer 1987

[IBM73] "IBM 3740 Data Entry Sys", Ref Manual GA21-9151, 1973

[Keet97] Keeton, K, R Arpac-Dusseau and D Patterson, "IRAM and SmartSIMM: Overcoming the I/O Bus Bottleneck", Tech Report, Univ of California, Berkeley, 1997

[Lang76] Lang, T., et al, "An Architectural For a Large Database System Incorporating a Processor for Disk Search", *Computer Science Dept., UCLA., 1976*

[Lowe79] Lowenthal, E., "Data Base Processors: What Can They Do?", *Computerworld*, 6/79

[Mall80] Maller, V., "Retreiving Information", *Datamation*, 9/80

[Oneil91] O'Neil, Patrick, "Set Query Benchmark", *Database Systems Performance Handbook*, Morgan Kauffman.

[Raab94] Raab, Francis (editor), "TPC Benchmark (TM) D (Decision Support)", Working Draft 7.0, Transaction Performance Processing Council, San Jose, Calif., May 6, 1994.

[Ried97] Riedel, Erik and Garth Gibson, "Active Disks - Remote Execution for Network-Attached Storage", Tech Report, Carnegie Mellon Univ, CMU CS-97-198, Dec 1997

[Ried01] Riedel, E, C Faloutsos, G Gibson and D Nagle, "Active Disks for Large-Scale Data Proc", *IEEE Comp*, 6/01, pp 68

[Schi02] Schindler, Griffin, Lumb, and Ganger, "Track-aligned Extents: Matching Access Patterns to Disk Drive Characteristics", *FAST Conference Proc*, Monterey, CA, 1/28/02

[Sega95] Segate (Barracuda) ST15150N/ND, Segate Technical Product Manual no. 83328880, vol 1, June 1995

[Slot70] Slotnick, D.L., "Logic per Track Devices", *Advances in Computers* (Vol 10), NY, Academic Press, 1970, pp 291

⁸ The exception to this rule is the SQL join.