PARALLEL ALGORITHMS AND SECONDARY STORAGE METHODS FOR X'X

by

Setrag N. Khoshafian
Douglas M. Bates
David J. DeWitt

Computer Sciences Technical Report #528

January 1984

Parallel Algorithms and Secondary Storage Methods For X'X

Setrag N. Khoshafian
Douglas M. Bates
David J. DeWitt

Computer Sciences Department
University of Wisconsin - Madison

This research was supported by the Department of Energy under contract #DE-AC02-81ER10920.

ABSTRACT

This paper compares several algorithms for the parallel evaluation of X'X. The data matrix X is assumed to be large and the algorithms are with respect to the transposed secondary storage layout of X. The algorithms differ in the looping scheme as well as buffer management strategies. The analysis compares the performances of the transposed and relational secondary storage organizations, as well as the relative performance of the different parallel algorithms for the transposed organization.

1. Introduction

In statistical computing the time required to perform a specific statistical analysis is usually measured by the time to perform the calculations involved. In fact, early comparisons of algorithms only considered the number of floating point multiplications involved as this time was assumed to be the predominant part of the execution time [GENT73]. Later, as the relative speeds of different floating point operations has changed, it has become customary to measure the computational cost of an algorithm by the number of floating point operations where a floating point operation includes a floating point multiplication, a floating point addition, and associated indexing operations. This still assumes, though, that the cost of moving the data within an algorithm is negligible compared to calculation times.

When working with large data sets, however, it is frequently the case that the time for data movement can be comparable with the calculation time. As the size of the database extends to tens of thousands of observations on hundreds of variables or more, the time for input/output operations can, in fact, dominate calculation time.

In this paper we analyse the data movement and calculation time for a particular part of many statistical algorithms applied to a large data base. By a large data base we mean that the entire data set will not fit into the available primary memory.

We present a detailed analysis of the step of forming X'X, where the prime denotes transpose and X is an n by p matrix with $n > p$. This is a basic step in many regression algorithms and algorithms for principal components analysis and factor analysis and it has the important property that it is the only time when the entire data set must be accessed. Subsequent steps in the algorithms work on the p by p matrix which is formed in this step so the greatest part of

the data movement cost will be encountered in the X'X step. Here we have considered only the evaluation of the non-diagonal elements of X'X. The diagonal elements, which are the norm squared of the columns of X, can be assumed to have been precalculated and available in a summary database [BATE82] which contains several aggregate values (such as sum, min, max, norms etc.) for each variable.

It can be argued that it is not appropriate to consider X'X at all. Alternative approaches to regression analysis and the other statistical methods mentioned above avoid the calculation of X'X altogether and instead use methods based on orthogonal-triangular factorizations of X providing greater numerical stability at the expense of more calculation. Despite the greater stability of other methods we have decided to analyse algorithms for X'X because it is a simpler calculation so the methods are more understandable. Also it is the algorithm which is actually used in many statistical packages such as SAS [RAY82].

The greater stability of the QR method or methods based on Given's rotations can be matched in an X'X method by accumulating dot products to a greater precision than the precision of the data. With the introduction of extended "working" precision registers in the I.E.E.E. floating point standard [COON80], this is feasible even for data which is stored in "double" precision.

It can also be argued that it is not necessary to consider the data flow requirements of this type of algorithm because one can simply rely on a virtual memory system to provide the necessary address space and then act as if the primary memory was sufficiently large. This does not solve the problem of data movement costs but only passes it from the user's program to the operating system which may not handle it particularly well. In fact, in experimental timings on actual multi-user systems using the Linpack package and large matrices, we have found the amount of time required by the operating system

for paging is comparable to and often to exceed the actual time spent in calculation.

Another approach is to randomly sample selected observations from the database and use the results from the sample. This is appropriate for initial, exploratory analyses of a data set where the purpose is to determine the structure of the data but in later, confirmatory analyses, where the detailed properties of the data are established, it is important to utilize all the data which is available.

This paper considers a dense data set X, and analyses the overhead of data movement for the evaluation of X'X. The analysis is with respect to three different looping schemes and a number of buffer management strategies for each. The buffer management and page reference string algorithms are with respect to the transposed [TURN79] organization of the database.

In Section 2, we present the alternative secondary storage organizations of the data file. Section 3 gives the system parameters for the cost evaluation model. Section 4 describes buffer management and page replacement algorithms for the parallel [1] evaluation of X'X. In Section 5 we give the cost functions of the different algorithms in terms of the system parameters, and plot different graphs which describe the performance of the algorithms with respect to the size of the primary storage, the number of active columns and cpu speed. Section 7 summarizes our conclusions and observations.

2. File Organization

In most conventional relational databases the n-tuples of each relation are stored record-wise. In a statistical application a data matrix represents a relation and therefore a conventional DBMS will store the data matrix row-wise.

[1] By "parallel" we mean the concurrent execution of the I/O and the processing subsystems.

However, there are at least two reasons why such a secondary storage organization might not be an attractive alternative for a large statistical database.

The first reason relates to data compression. In statistical analysis it is common to analyze "qualitative" data. That is, data with a very small range of values. Therefore, for qualitative attributes only a small number of bits are needed to code all the different data values. A particular relation might have several qualitative (each with a different number of code-bits) and quantitative attributes per record. Because of this non-uniformity, compression will be difficult in record oriented systems. Moreover, it is common in statistical databases to have data either of equal length or identical values clustered within the attributes. If the data is stored attribute-wise it is feasible to implement "run length" [EGGE81] compression techniques.

A second reason why a relational row-wise storage organization might not be a suitable organization for statistical databases, is that statistical operations usually encompass most (or all) of the observations (rows) and few of the variables (columns). That is most of the time only a fraction of the columns are referenced. For example, if all the rows and only 10% of the columns are accessed, all the relation must be retrieved. This implies an unnecessary retrieval of potentially 90% of the relation. For large databases this can have serious performance implications.

These objections, and especially the second one, suggest an alternative storage organization namely the *transposed* file organization [BATO79, TURN79, BURN81]. The two types of transposed file organizations are: (1) partially transposed (or clustered transposed) and (2) completely transposed, which is a special case of the partially transposed organization. In the partially transposed organization the set of attributes is decomposed into a mutually exclusive collection of subsets of attributes and the attributes in each subset are stored

together. That is, a collection of n-tuples is stored as k collections of n_i-tuples where $n_1 + n_2 + \cdots + n_k = n$. However, this process does not create new relations, new keys etc.. The attributes which are stored together are hopefully the ones which are also accessed together. If each attribute subset contains exactly one attribute we get the completely transposed organization.

As indicated, it is common in statistical analysis to process only a fraction of the columns (henceforth called the "active" columns). The page reference string algorithms in this paper are with respect to the completely transposed secondary storage organization. The algorithm for the relational organization is rather straight forward.

3. System Parameters

In this section the parameters used in the cost equations of X'X are introduced. Besides the disk and CPU parameters, several model parameters are also introduced. These parameters include the size of the relation, the size of the transposed files, the number of attributes, the number of active columns and the buffer size. For each parameter the value or range of values of the parameter are indicated.

3.1. Disk Parameters

The mass storage device is assumed to be a disk. The parameters for a disk are: (1) the track size (BSIZE); (2) the number of tracks per cylinder (DCYL); (3) the average time to read/write a page (Tio); (4) the average random seek time (Tdac); and (5) the cylinder-to-cylinder seek time (Tsk). The values of the disk parameters are those of the IBM 3350 disk drive [IBM77]. All the values are in milliseconds (ms.). A page (the unit of transfer between the disk and main

store) will be the same size as a disk track throughout.

BSIZE	page size	19069	bytes
DCYL	number of pages per cylinder	30	pages
Tio	page read/write time	25	ms.
Tdac	average access time	25	ms.
Tsk	time to seek 1 track	10	ms.

The I/O time to read U contiguous pages randomly is approximately:

$$T_{r-io}(U) = Tdac + \left\lfloor \frac{U}{DCYL} + 1/2 \right\rfloor Tsk + U \cdot Tio$$

3.2. CPU Parameters

The arithmetical computations of X'X consists of the "Inner Product" operation:

Inner Product: $a = a + x_j \cdot x_i$

Here the unit of execution is a "floating point operation" (a "flop") [DONG79], which involves the execution time of a double precision floating point multiplication, a double precision floating point addition, some indexing operations and storage references. Our own experiments and the timing indicated in [DONG83], suggest that, 0.5 to 25 μ seconds (micro seconds) per flop is a reasonable range of processing speeds.

For the transposed file organization we shall use the function

$$T_{INN}(U) = U \cdot q \cdot flop$$

where U is the number of page pairs whose inner product is to be accumulated, q is the number of elements per page, and flop is the time to perform a floating point operation. For the relational organiztion we shall use

$$T_{INN}(U) = U \cdot nt \cdot \frac{p \cdot (p-1)}{2} flop$$

where here, p is the number of active columns, nt is the number of tuples per page, and U is the number of pages.

3.3. Model Parameters

Since the relational and the transposed organizations are being compared it is imperative to define the parameters of each. In relational form, the data set is assumed to have 100 attributes and 230,000 tuples. All the attributes are eight byte numbers. With the above disk parameter values, this implies that there are 23 tuples per page (that is nt = 23). In the transposed organization, each page will contain 2383 elements. To simplify the subsequent analysis it is assumed 2300 elements are stored in each page of a transposed file. Hence each attribute will occupy 100 pages. Thus in both the row-wise organization and the transposed organization the relation will occupy 10,000 pages. These parameters are as follows:

R	number of pages occupied by the relation(matrix)	10,000	pages
N	number of pages for a column(transposed)	100	pages
w	number of attributes	100	
n	number of tuples	230,000	
p	number of active columns for the matrix operations the values considered are 10 to 100 active columns		
M	the size of the main storage the values considered are 10 to 200 pages		

4. Parallel Algorithms For X'X

In this section we present several schemes for implementing parallel evaluations of X'X. In Section 4.1, we give three high level algorithms for the evaluation of the matrix X'X. In Section 4.2, we specify page reference string and logical memory organization strategies for the different high level algorithms of Section 4.1.

4.1. Algorithms

The three (high level) algorithms evaluated, "Vector Building Block", "Vector Times Matrix," and, "Horizontal Stripes," are called VBB, VTM, and, ST respectively. In the "Vector Building Block" algorithm, X'X is evaluated as $\frac{p \cdot p-1}{2}$ inner products. This is perhaps the most obvious way of evaluating X'X. However, it is the most I/O intensive of all the three approaches. With the "Vector Times Matrix" algorithm, X'X is evaluated as (p-1) "vector times matrix" operations. The "vector" of the k^{th} operation is the k^{th} column, and the matrix is the remaining (p-k) columns. The "Horizontal Stripes" algorithm involves the fewest page transfers, since here the active columns of X are read once. In this algorithm, after reading a "stripe" of the active columns, the partial inner products are accumulated before processing the next stripe.

VBB:

```
For i := 1 to p-1
    For j := i+1 to p
        For k := 1 to n
            c_ij := c_ij + X_ki·X_kj
```

VTM:

```
For i := 1 to p-1
    For k := 1 to n
        For j := i+1 to p
            c_ij := c_ij + X_ki·X_kj
```

ST:

```
For k := 1 to n
    For i := 1 to p-1
        For j := i+1 to p
            c_ij := c_ij + X_ki·X_kj
```

4.2. Page Reference Strings and Memory Management

In this section we shall present several page reference string and memory management strategies for each of the high level algorithms of the previous section. In the following $R_i^{U,k}$ stands for the k^{th} U-page block of the i^{th} column of X. If there is no U, the block size is one.

4.2.1. Vector Building Block Algorithms

There are two strategies for the vector building block algorithm.

[1] -Divide the M pages of primary memory into three logical subdivisions: a memory unit of M-2 pages and two memory units of size one page each.

-For each inner product read the next M-2 pages of one column and then, page by page, read the corresponding M-2 pages of the other column accumulating the inner product of a resident page pair while reading the next page.

VBB 1:

For i := 1 to p-1

For j := i+1 to p

For k := 1 to N/(M-2)

$R_i^{M-2,k}$; $R_j^{(k-1)(M-2)+1}$

For q := 1 to M-2

\$ $c_{ij} := c_{ij} + X_i^{(k-1)(M-2)+q} \cdot X_j^{(k-1)(M-2)+q}$

\$ if q < M−2 then $R_j^{(k-1)(M-2)+q+1}$

[2] -Divide the M pages of primary memory into four logical subdivisions and allocate M/4 pages to each subdivision.

-For each inner product accumulate the inner product of the resident M/4 page blocks and, concurrently, read the next pair of M/4 pages.

VBB 2:

For i := 1 to p-1

For j := i+1 to p

$R_i^{(M/4),1}$; $R_j^{(M/4),1}$

For k := 1 to N/(M/4)

\$ $c_{ij} := c_{ij} + X_i^{(M/4),k} \cdot X_j^{(M/4),k}$

\$ if k < N/(M/4) then [$R_i^{(M/4),k+1}$; $R_j^{(M/4),k+1}$]

4.2.2. Vector Times Matrix Algorithms

We shall call the vector which multiplies the matrix of the remaining columns in a "vector times matrix" step of this algorithm, the "operating" column. Here again we have two strategies:

[1] -Divide the M pages of primary memory into three logical subdivisions: a memory unit of M-2 pages and two memory units of size one page each.

-For each operating column, read the next M-2 pages and then, page by page, read the corresponding M-2 pages of each of the remaining columns accumulating the inner product of a resident page pair while reading the next page.

VTM 1:

For i := 1 to p-1

For k := 1 to N/(M-2)

$R_i^{(M-2),k}$; $R_{i+1}^{(k-1)(M-2)+1}$

For j := i+1 to p

For q := 1 to M-2

\$ $c_{ij} := c_{ij} + X_i^{(k-1)(M-2)+q} \cdot X_j^{(k-1)(M-2)+q}$

\$ if q < M−2 then $R_j^{(k-1)(M-2)+q+1}$

else if j < p then $R_{j+1}^{(k-1)(M-2)+1}$

[2] -Divide the M pages of primary memory into three logical subdivisions and allocate M/3 pages to each subdivision.

-For each operating column, read the next M/3 pages and then read the corresponding M/3 page blocks of the remaining columns, accumulating the inner product of a resident M/3 page pairs while reading the corresponding M/3 pages of the next column.

VTM 2:

For i := 1 to p-2

$R_i^{M/3,1}$

For k := 1 to N/(M/3)

$R_{i+1}^{(M/3),k}$

For j := i+ 1 to p

$ $c_{ij} := c_{ij} + X_i^{(M/3),k} \cdot X_j^{(M/3),k}$

$ if j < p then $R_{j+1}^{(M/3),k}$

else if k < N/(M/3) then $R_i^{(M/3),k+1}$

$R_{p-1}^{(M/4),1}$; $R_p^{(M/4),1}$

For k := 1 to N/(M/4)

$ $c_{p-1,p} := c_{p-1,p} + X_{p-1}^{(M/4),k} \cdot X_p^{(M/4),k}$

$ if k < N/(M/4) then [$R_{p-1}^{(M/4),k+1}$; $R_p^{(M/4),k+1}$]

4.2.3. The Horizontal Stripes Algorithm

If we were to measure the effectiveness of an algorithm by the number of pages transfers, this algorithm is obviously optimal since it requires all the active columns of the data matrix to be read exactly once. Comparing the horizontal stripes algorithm with the vector building block and the vector times matrix algorithms, we make the following observations:

(1) with a completely transposed secondary storage organization, this algorithm requires at least p (p being the number of active columns) pages of primary storage

(2) since the algorithm requires corresponding pages of the active columns to be accessed simultaneously, with a given memory size, the algorithm processes the matrix columns in smaller blocks than either the vector building block or the vector times matrix algorithms. This could imply more disk seeks.

(3) the amount of computation within the overlap region is substantially greater than either the vector times matrix or the vector building block algorithms.

Here we have three different strategies:

[1] -Divide the memory into p (the number of active columns) logical subdivisions

-For each stripe, after reading the next M/p pages of the first two columns start accumulating the inner products of the columns which are already resident, while concurrently, reading corresponding M/p page blocks of the columns which are not yet accessed.

To avoid the indexing and timing details we have introduced a function called "Find", which accesses a shared linear array A (of size p) and a two dimensional array D (p by p) and finds two indexes i and j, such that the current M/p page blocks of X_i and X_j are resident and the inner product of X_i and X_j are not accumulated. The function Find waits and does not return until it finds a pair. The i^{th} position of A is set as soon as the current M/p pages of X_i are read. Since A is referenced by two independent processes, it should be in a critical region.

```
ST 1:
    For k := 1 to N/(M/p)
        clear(A); clear(D)
        DONE := 0
        $ For i:= 1 to p
            R_i^(M/p),k
            set(A[i])
        $ Repeat
                Find(i, j)
                c_ij := c_ij + X_i^(M/p),k . X_j^(M/p),k
                set(D[i,j])
                DONE := DONE + 1
          Until DONE = p(p-1)/2
```

[2] -Divide the main storage into p+1 equal subdivisions

-If there is a free M/(p+1) page block, read the next block while concurrently accumulating inner products of resident block pairs

-If all the partial inner products for a column are acumulated free the block occupied by the column.

The main difference between this algorithms and [1], is that in [1] a horizontal stripe is processed completely before the next horizontal stripe is read. In [2], M/(p+1) page blocks of the next horizontal stripe are referenced while processing the current stripe. In order to allow more blocks of the next stripe to be accessed, we have introduced the procedure "Release", which releases a block (of M/(p+1) pages) from the current horizontal stripe. For each active column, a counter is incremented when an inner product of the active column is accumulated. When the counter achieves the value p-1, the block which holds the M/(p+1) pages of the column is released. Since there are p+1 subdivisions, there could be blocks from at most two horizontal stripes. Therefore, there are two shared linear arrays (of size p each), which indicate the resident blocks from the two stripes. Finally, the boolean function "Free" attempts to find a free block of M/(p+1) pages, and does not return until it finds one.

ST 2:

```
$ q := 0
 For k := 1 to N/(M/(p+1))
        For i := 1 to p
            If Free(M/(p+1)) then R_i^(M/(p+1)),k
            set(A[q,i])
        q := (q+1) mod 2
$ clear(A); q:= 0
 For k := 1 to N/(M/(p+1))
        clear(D)
        DONE := 0
        For j:= 1 to p [ count[j] := 0 ]
        Repeat
            Find(q,i,j)
            c_ij := c_ij + X_i^(M/(p+1)),k . X_j^(M/(p+1)),k
            count[i] := count[i]+1; count[j] := count[j]+1
            If count[i] = (p-1) then Release(X_i^(M/(p+1)),k)
            if count[j] = (p-1) then Release(X_j^(M/(p+1)),k)
            set(D[i, j])
            DONE := DONE + 1
        Until DONE = p(p-1)/2
        clear(A[q,])
        q := (q+1) mod 2
```

[3] -Divide the memory into 2p logical subdivisions

-Read the next stripe of M/2p pages from each active column, while evaluating the inner products of the current stripe

Note: With this scheme we need at least 2p pages of primary memory

ST 3:

$R_1^{M/2p,1} R_1^{M/2p,1} \ldots R_p^{M/2p,1}$

For k:= 2 to N/(M/2p)

$ For i:= 1 to p-1

For j := i+1 to p

$c_{ij} := c_{ij} + X_i^{(M/p),k-1} \cdot X_j^{(M/p),k-1}$

$ if k < N/ (M/ 2p) then

$R_1^{M/2p,k} R_1^{M/2p,k} \ldots R_p^{M/2p,k}$

5. Cost Functions

In the previous sections we presented several algorithms for implementing X'X. In this section we give the cost functions for each algorithm. More specifically, for each algorithm we find: (1) Total I/O cost; (2) Average execution time in overlap region; (3) Total execution time (without cost of messages).

5.1. I/O Costs

The table below gives the total I/O costs for each algorithm in terms of the function T_{r-io}. We have also indicated the minimum memory size which must be available in order for the algorithm to be implementable.

Algorithm	Total I/O	Min. Memory Size (Pages)
VBB 1	$p(p-1)\cdot \frac{N}{M-2}\cdot T_{r-io}(M-2)$	3
VBB 2	$p(p-1)\cdot 4\frac{N}{M}\cdot T_{r-io}(\frac{M}{4})$	4
VTM 1	$(\frac{p(p+1)}{2}-1)\cdot \frac{N}{M-2}\cdot T_{r-io}(M-2)$	3
VTM 2	$(\frac{p(p+1)}{2}-3)\cdot 3\frac{N}{M}\cdot T_{r-io}(\frac{M}{3}) + 8\cdot \frac{N}{M}\cdot T_{r-io}(\frac{M}{4})$	4
ST 1	$p^2\cdot \frac{N}{M}\cdot T_{r-io}(\frac{M}{p})$	p
ST 2	$p(p+1)\cdot \frac{N}{M}\cdot T_{r-io}(\frac{M}{p+1})$	p+1
ST 3	$2\cdot p^2\cdot \frac{N}{M}\cdot T_{r-io}\frac{(M}{2p})$	2p

In Figure 1 we have the curves of the number of page transfers as a function of the number of active columns. The curve of VBB is $p\cdot(p-1)\cdot N$. That of VTM is $(\frac{p\cdot(p+1)}{2}-1)\cdot N$ and of ST is $p\cdot N$. Since the number of page transfers of ST is linear in p it appears constant in the graph. It is interesting to note that the "Vector Building Blocks" algorithms will involve fewer page transfers than the number of pages transferred with the relational secondary storage organization (that is 10,000 pages), only if p (the number of active columns) is less or equal to 10. The corresponding critical value for the "Vector Times Matrix" algorithms is 13. Of course the "Stripes" algorithms will always involve fewer page transfers (unless all the columns are active).

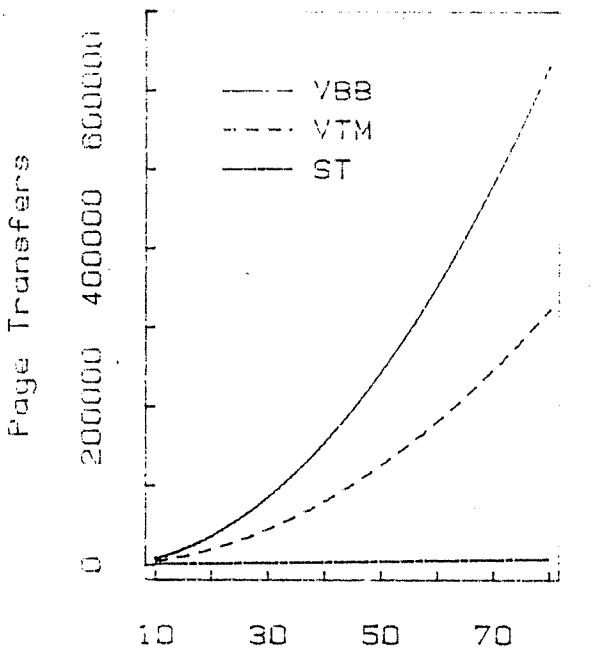


Fig. 1

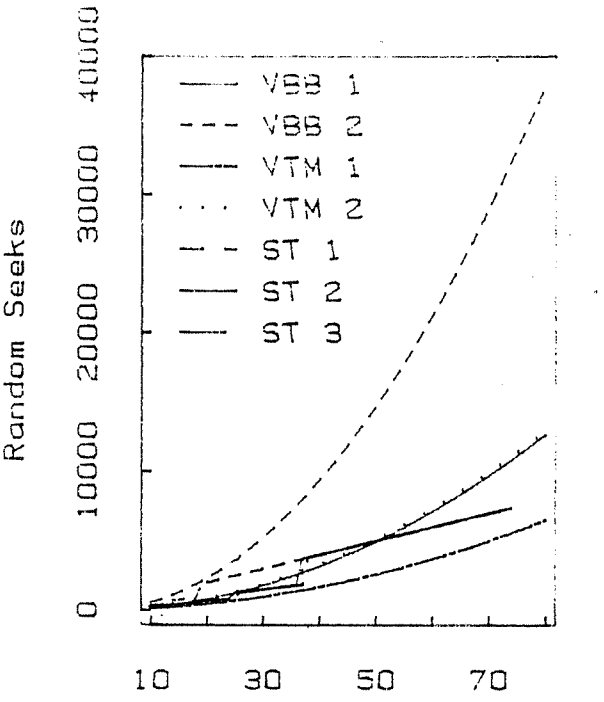


Fig. 2

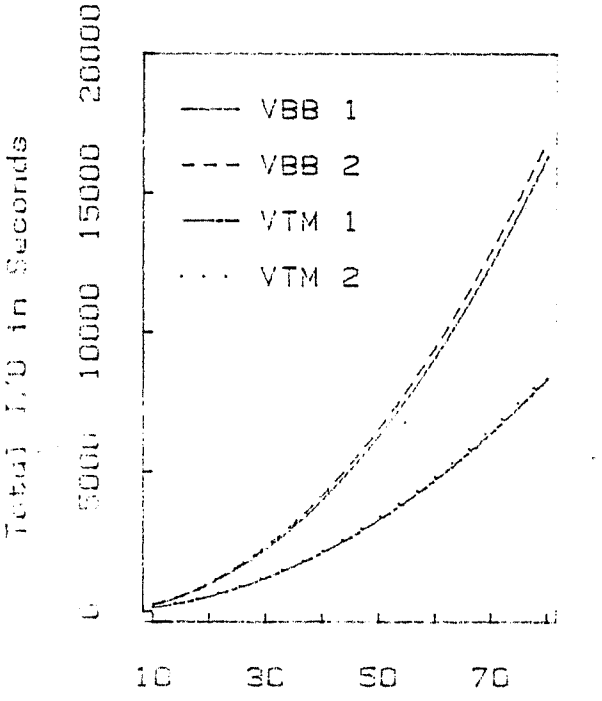


Fig. 3

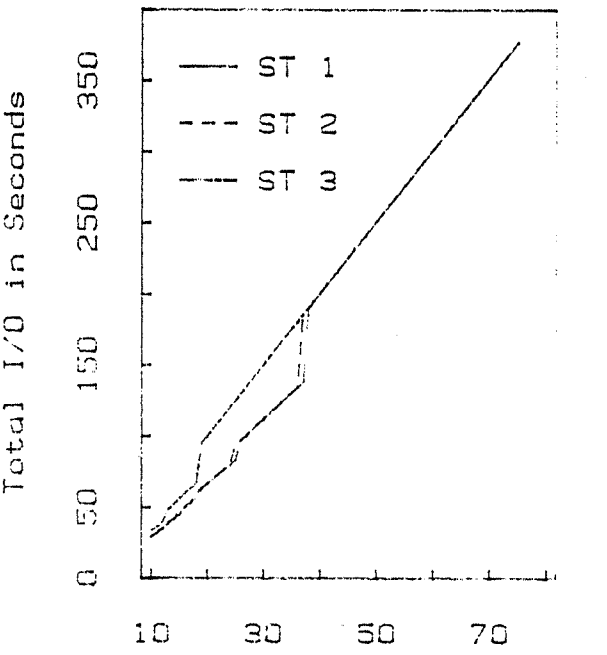


Fig. 4

As the secondary storage medium is a disk, another measure of the I/O is the number of random seeks. Figure 2 gives the number of random seeks also as a function of the number of active columns. The main storage size is held at 75 pages (tracks). It is clear from Figure 2 that the first algorithm of the "Vector Times Matrix" approach involves the least number of random seeks for any number of columns. On the other hand the second algorithm of the "Vector Building Blocks" approach has the largest number of random seeks also for any number of active columns. The curves for the number of random seeks of the other algorithms are between these two extrema. It is interesting to note that the first and second algorithms of the "Stripes" approach outperform the others (except VTM 1) only when the number of active columns exceed approximately 50. The reason is that although the "Stripes" algorithms reference the data matrix only once, the main storage is divided into O(p) logical subdivisions. As we noted earlier, the large number of random seeks is the consequence of accessing the transposed columns in smaller blocks.

In Figure 3 we have the curves for the total (transfer plus seek) I/O time as a function of the number of active columns for the first four algorithms (that is all the implementations of VBB and VTM). The time is in seconds and the main storage is again held at 75 pages. The figure clearly shows the effect of fewer page transfers for the "Vector Times Matrix" algorithms. The first algorithm of the "Vector Building Bocks" approach outperforms the second algorithm. This was to be expected, since the second algorithm involves a larger number of random seeks (see Figure 2). Figure 4 gives the corresponding curves for the three implementations of ST. The curves for the first two algorithms are almost identical, except that the first algorithm involves fewer random seeks (since the memory is divided into p logical subdivisions rather than (p+1)) and therefore is slightly better. The breaks in the curves correspond to the same breakpoints as in Figure 2. In other words, these breaks are due to a substantial increase in

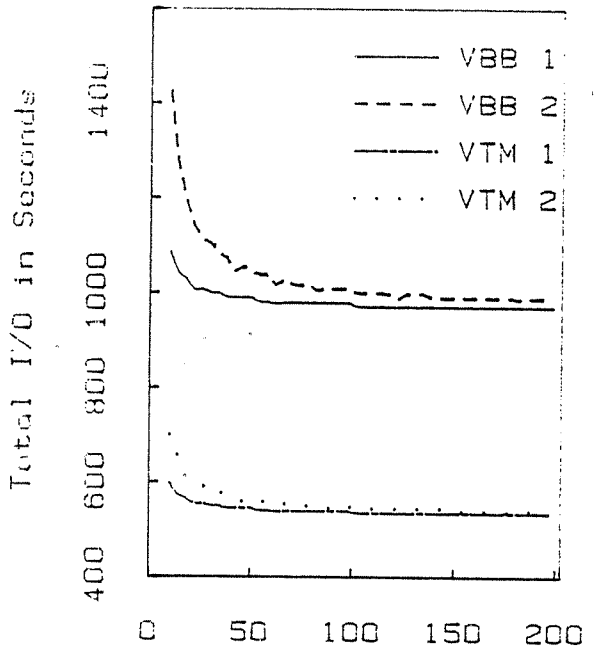
the number of random seeks. For example, with M = 75, when p = 25 M/p is equal to 3. Therefore ceil(100/3) = 34, and the total number of random seeks is 25.34 = 850. On the other hand when p =26, then M/p = 2 and the total number of random seeks is 26.50 = 1300. Note that with the stripes algorithm a substantial percentage of the main storage can be wasted. With the previous example, 49 pages (or approximately 65%) of the main storage are not used.

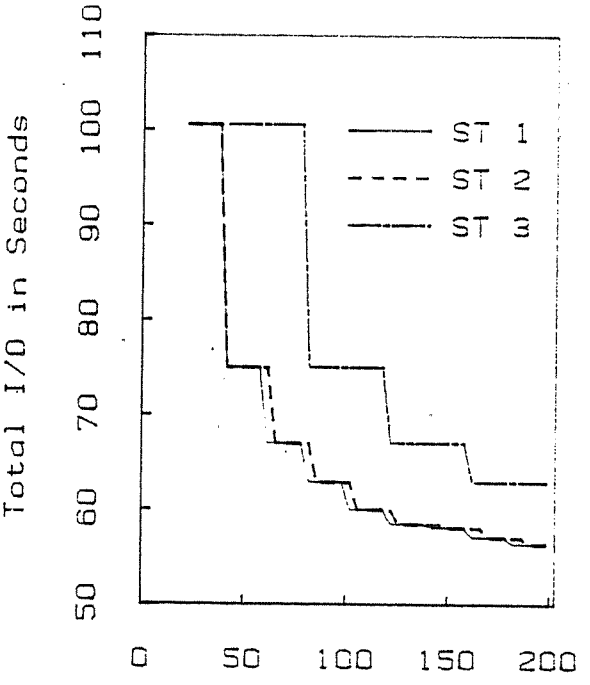
Again comparing the I/O cost of these algorithms with the I/O cost of the relational organization, let us point out that the total I/O for the relational organization is approximately 253 seconds. The critical number of active columns for the "vector Building Block" and "Vector Times Matrix" algorithms are again 10 and 13 respectively. However, for ST 1 and ST 2 the critical number of active columns are, respectively, 50 and 41. This shows that a substantial percentage of the total I/O time for the stripes algorithms is seeks [2]. Finally Figure 5 and Figure 6 are the curves of the total I/O times (in seconds) as functions of the main storage size for p (the number of active columns) = 20. Here the general observation is that, if we neglect track-to-track seeks the curves generally look like:

$$F(M) = C_1 + \frac{C_2}{M}$$

where, C_1 and C_2 are in terms of N, p and the disk parameters. The oscillations in the curves of Figure 5 are due to the overhead of the track-to-track seeks. For example, if M = 120 there are no track to track seeks for the first algorithm of VBB. On the other hand when M = 124, there are three track-to-track seeks per column. The sharp edges of the curves in Figure 6 are due to the fact that trunc(M/Pv) remains constant for Pv consecutive values of M. In ST 1, Pv = p; in ST 2 Pv = p+1; and in ST 3, Pv = 2·p.

[2] In an earlier analysis we have shown that with 50 pages of main storage approximately 50% of the total I/O time is seeks


Total I/O in Seconds
400
600
800
1000
1400
VBB 1
VBB 2
VTM 1
VTM 2
0
50
100
150
200
Memory Size
Fig. 5


Total I/O in Seconds
50
60
70
80
90
100
110
ST 1
ST 2
ST 3
0
50
100
150
200
Memory Size
Fig. 6

5.2. Execution Time in Overlap Region

Here we find the execution time in the overlap regions. These overlap times are the maximums of the I/O and computational times in the overlap regions of an algorithm. However, since ST 1 and ST 2 require some synchronization between the I/O and processing subsystems, we have not given the overlap times for these two algorithms.

$$T_{overlap}^{(1,1)} = MAX(T_{INN}(q) , T_{io})$$

$$T_{overlap}^{(1,2)} = MAX(T_{INN}(q\cdot(\frac{M}{4})) , 2\cdot T_{r-io}(\frac{M}{4}))$$

$$T_{overlap}^{(2,1,r)} = MAX(T_{INN}(q) , T_{r-io}(1))$$

$$T_{overlap}^{(2,1,s)} = MAX(T_{INN}(q) , T_{io})$$

$$T_{overlap}^{(2,2)} = MAX(T_{INN}(q\cdot(\frac{M}{3})) , T_{r-io}(\frac{M}{3}))$$

$$T_{overlap}^{(3,3)} = MAX(\frac{p(p-1)}{2}\cdot T_{INN}(q\cdot\frac{M}{2p}) , p\cdot T_{r-io}(\frac{M}{2p})$$

5.3. Total Executions Times

The total execution time of an algorithm is the sum of three terms: $T_{start} + Q\cdot T_{overlap} + T_{flush}$, where T_{start}, and, T_{flush} are respectively the initial read time to start the execution and the final arithmetic time after all the data has been read. Q is the number of times the overlap region is executed.

$$\text{VBB 1}: \frac{N}{M-2}\cdot\frac{p(p-1)}{2}\cdot\left[T_{r-io}(M-2) + T_{r-io}(1) + (M-3)\cdot T_{overlap}^{(1,1)} + T_{INN}(q)\right]$$

$$\text{VBB 2}: \frac{p(p-1)}{2}\cdot\left[2\cdot T_{r-io}\left(\frac{M}{4}\right) + \left(\frac{N}{(M/4)} - 1\right)\cdot T_{overlap}^{(1,2)} + T_{INN}\left(q\cdot\left(\frac{M}{4}\right)\right)\right]$$

$$\text{VTM 1}: (p-1)\cdot\frac{N}{M-2}\cdot\left[T_{r-io}(M-2) + T_{r-io}(1) + T_{INN}(q)\right] +$$

$$\frac{N}{M-2}\cdot\left[\frac{p(p-1)}{2}\cdot(M-3)T_{overlap}^{(2,1,s)} + \frac{(p-1)(p-2)}{2}\cdot T_{olap}^{(2,1,r)}\right]$$

$$\text{VTM 2}: (p-2)\cdot\left[T_{r-io}\left(\frac{M}{3}\right) + \frac{3\cdot N}{M}\cdot T_{r-io}\left(\frac{M}{3}\right) + T_{INN}\left(q\cdot\frac{M}{3}\right)\right]$$

$$+ \left[\left(\frac{N}{(M/3)} - 1\right)\cdot\left(\frac{p(p-1)}{2} - 1\right) + \frac{(p-2)\cdot(p-1)}{2}\right]\cdot T_{overlap}^{(2,2)}$$

$$+ 2\cdot T_{r-io}\left(\frac{M}{4}\right) + T_{INN}\left(q\cdot\left(\frac{M}{4}\right)\right) + \left[\frac{N}{M/4} - 1\right]T_{overlap}^{(1,2)}$$

In the previous section we did not specify the overlap time for ST 1 and ST 2. The overall execution time of these algorithms depends on the speeds of the I/O and processing subsystems. Therefore, suppose that there exists an integer k such that:

$$T_{INN}((k+1)\cdot U\cdot q) \geq T_{r-io}(U) \geq T_{INN}(k\cdot U\cdot q)$$

where, $U = \frac{M}{p}$ in ST 1 and $U = \frac{M}{p+1}$ in ST 2. Then the total execution time of ST 1 is given by:

$$\text{ST 1}: \frac{N}{(M/p)}\cdot\left[(m+2)\cdot T_{r-io}\left(\frac{M}{p}\right) + \left[\frac{p(p-1)}{2} - \frac{m(m+1)}{2}\right]\cdot T_{INN}\left(q\cdot\left(\frac{M}{p}\right)\right)\right]$$

where $m = \min(k, p-2)$.

For ST 2 we have two cases:

Case 1: $k \geq \frac{(p-1)}{2}$. It is clear that the algorithm will be I/O bound and the execution time is:

$$ST2 : p \cdot \frac{N}{(M/(p+1))} \cdot T_{r-io}\left(\frac{M}{p+1}\right) + \left[(p-1) + \frac{(p-m-1)(p-m-2)}{2}\right] \cdot T_{INN}\left(q \cdot \frac{M}{p+1}\right)$$

where here $m = \min(k, p-1)$.

Case 2: $k < \frac{(p-1)}{2}$. Here the execution is computation bound and the total execution time is:

$$ST\ 2 : (k+1) \cdot T_{r-io}\left(\frac{M}{p+1}\right) + \left[\frac{N}{M/(p+1)} \cdot \frac{p(p-1)}{2} - \frac{k(k-1)}{2}\right] \cdot T_{INN}\left(q \cdot \frac{M}{p+1}\right)$$

Finally,

$$ST\ 3 : p \cdot T_{r-io}\left(\frac{M}{2p}\right) + \left(\frac{N}{(M/2p)} - 1\right) \cdot T_{overlap}^{(3,3)} + \frac{p(p-1)}{2} \cdot T_{INN}\left(q \cdot \frac{M}{2p}\right)$$

Figure 7 and Figure 8 give the curves for the the total execution times as functions of the number of active columns where in Figure 7 the time per floating point operation is 0.5 microseconds and in Figure 8 it is 25. We have considered only the second version of the algorithms VBB, VTM, and, ST. We have also included the curve for the total time of the relational organization. The main storage size is 100 pages in both figures. In Figure 7, VBB 2 performs better than the relational organization for $p \leq 10$. VTM 2 outperforms the relational for $p \leq 13$ and ST 2 for $p \leq 51$. However, in Figure 8 (where the time per flop is 25 microseconds), the corresponding critical values for p are 9, 9, and, 10. Therefore, the general observation is that with a very slow CPU, any implementation for the transposed organization will not yield significantly better performance than the relational organization if more than 10% of the columns are active. With a very fast CPU the situation is better (up to 50%) if we choose a

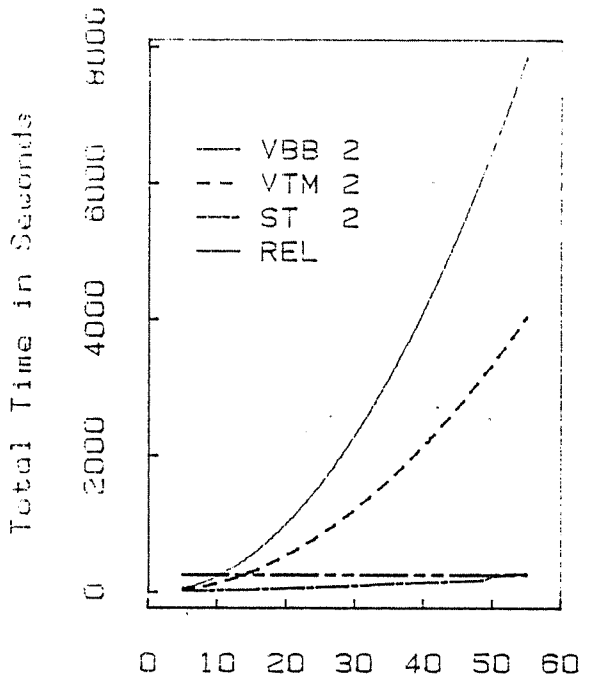

Total Time in Seconds
0
2000
4000
6000
8000
VBB 2
VTM 2
ST 2
REL
0 10 20 30 40 50 60
Number of Active Columns

Fig. 7

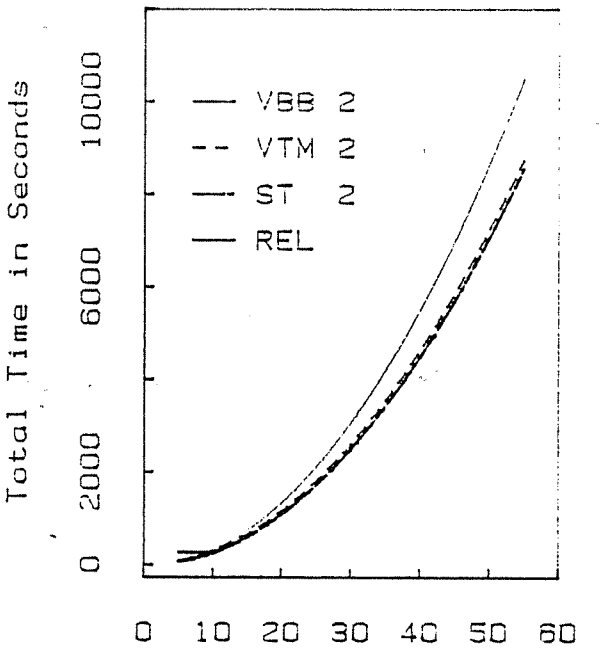

Total Time in Seconds
0
2000
6000
10000
VBB 2
VTM 2
ST 2
REL
0 10 20 30 40 50 60
Number of Active Columns

Fig. 8

good algorithm.

Figure 9 and Figure 10 are similar plots but here the number of active columns is held fixed (p = 20) and the total execution time is plotted against the main storage size. Figure 9 clearly shows the relative performance of the three algorithms. As it was expected, the curves look similar to Figure 5 (where we plotted only the I/O). The performance of VBB 2 in Figure 10 is interesting since it actually possesses a minimum. The shape of the curve can be easily explained by the fact that as the size of the main storage increases, the "startup" and "flush" times per inner product also increase. In fact for each inner product the "startup" time is the time to read M/4 pages from each column and the "flush" time is approximately the time to accumulate the inner products of the last M/4 block pair. For a small memory size the time in the overlap region is I/O. However, since the CPU is slow, soon (for M greater than 26) the overlap region becomes CPU bound. Therefore the total time for an inner product becomes the startup I/O time plus the inner product time. A larger memory size will imply a longer startup time, and, therefore, a greater total execution time. This observation also holds for VTM 2. Here however, the degradation is not serious.

In Figure 11 we have the total execution times for all the algorithms in terms of the time per floating point operation (which is inversely proportional to the CPU speed). Here the main storage size is 75 pages and the number of active columns is 20. All the curves (except VBB 2) have a I/O bound region and a CPU bound region. The critical value for the "Stripes" algorithms is approximately 2 microseconds per flop and for the other algorithms about 10. The curves show that the "Stripes" algorithms become CPU bound much faster than the others.

Finally, to provide an idea about the required memory bandwidths to support high CPU speeds, we have plotted in Figure 12 the memory bandwidth

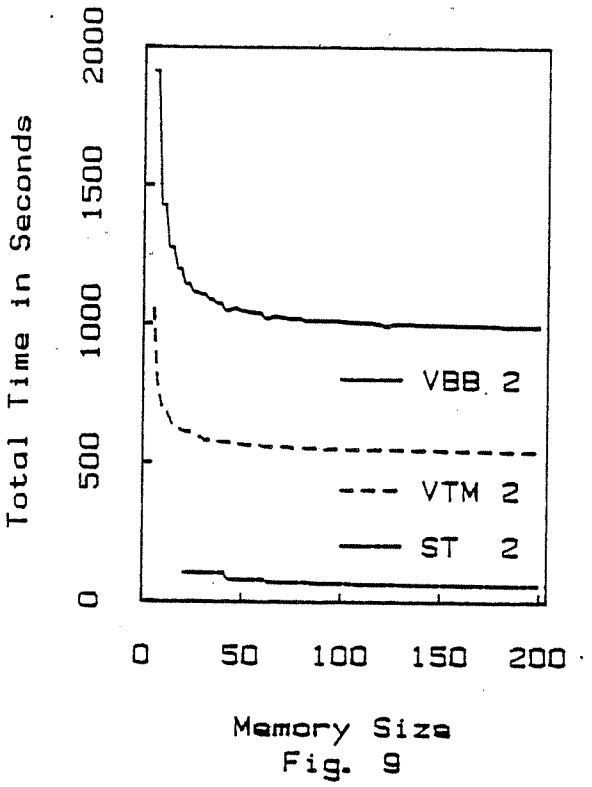


Fig. 9

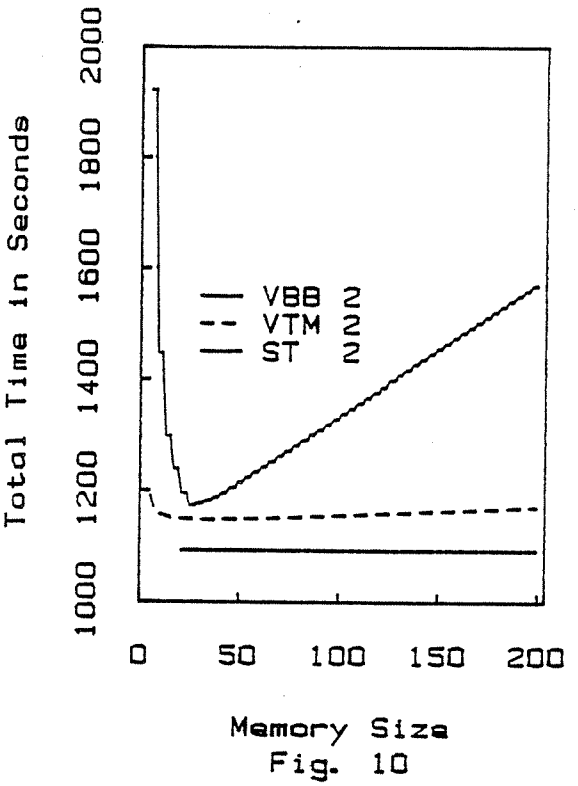


Fig. 10

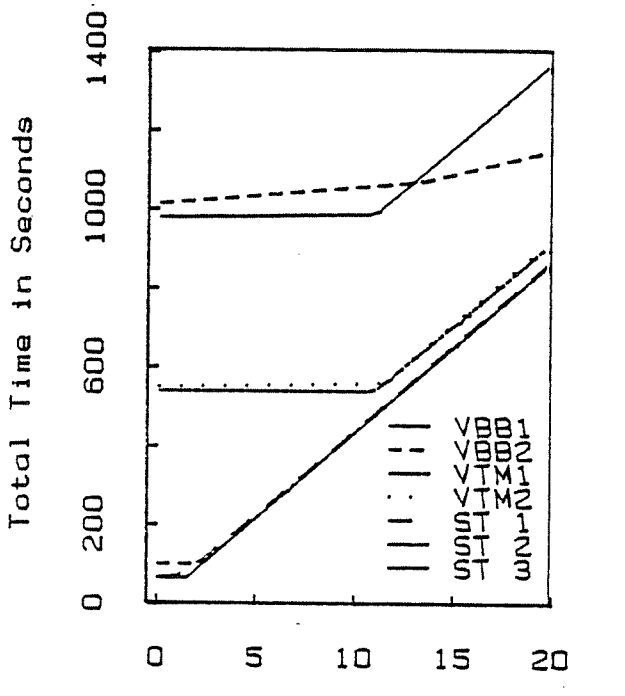


Fig. 11

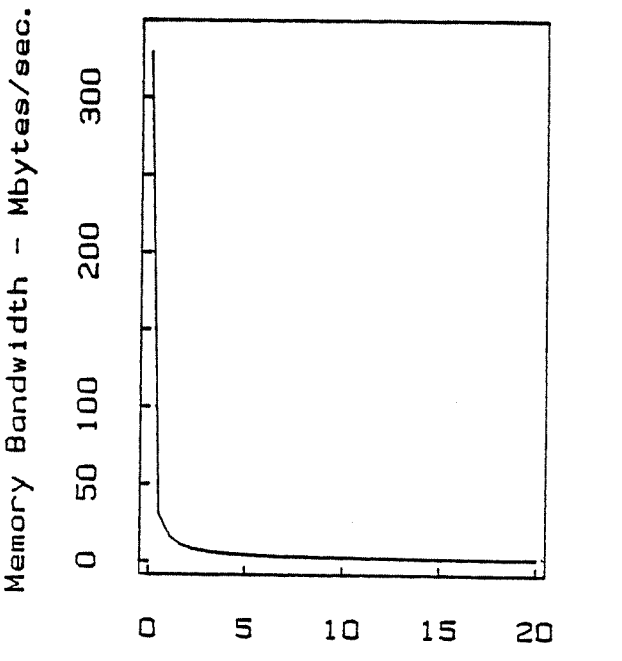


Fig. 12

versus the time per flop. An interesting point is that the required memory bandwidths of all the algorithms were quite close, especially for fast CPU's. Suprisingly, the algorithm which required the least memory bandwidth was ST 3. The algorithms which required the highest memory bandwidths were VBB 1 and VTM 1. But, as we said earlier, the differences between these extrema were not significant. For example, if the time per flop is 2 microseconds, the required memory bandwidth for VTM 1 and VBB 1 was 8.5 megabytes per second and for ST 3 it was 8.1 megabytes per second. Since the curves for the different algorithms looked almost identical, this curve is that of ST 2 (with $p = 20$ and $M = 75$).

6. Conclusions

This paper presents three looping alternatives for the evaluation of X'X. These are labeled "Vector Building Blocks" (or VBB), "vector Times Matrix" (or VTM), and "Stripes" (or ST). For each we have given a number of buffer management and page replacement algorithms in a high level concurrent programming language, wherein we have explicitly stated the page reference strings and the overlap regions. The only type of parallelism considered here is the overlapped execution of the processing and I/O subsystems.

This paper provides a comparative performance evaluation and analysis at several levels. First we show that the number of page transfers grows quadratically in p (the number of active columns) for the "Vector Building Block" and "Vector Times Matrix" algorithms and linearly in p for the "Stripes" algorithms. On the other hand, the number of random seeks of the "Stripes" algorithms is less than the other algorithms only for $p \leq 50$. The effect of these is seen in the curves of the total I/O times as functions of the number fo active columns. Although random seeks is a substantial percentage (approximately 50% for p and $M = 50$) of the total I/O time of the "Stripes" algorithms, the corresponding

percentages for the other algorithms are approximately 3% for seeks and 97% for page transfers. Therefore, the curves of the total I/O times for the "Vector Building Block" and "Vector Times Matrix" algorithms are quadratic in p, but the substantial percentage of seeks in the curves of the "Stripes" algorithms show as jumps in the linear curve.

The curves of the total I/O times as functions of the main storage size are asymptotic. After a certain threshold, the size of the main storage does not effect the total I/O time to any significant degree. More specifically, if the main storage is divided into k equal subdivisions [3] then to be within f percent of the asymptote the memory size needs to be:

$$M = \frac{k \cdot Tdac}{f \cdot Tio}$$

For example with 20 active columns, to be within 5% of the least I/O, M needs to be approximately 20 for VBB 1 and VTM 1, 80 for VBB 2, 60 for VTM 2, 400 for ST 1 and ST 2, and 800 for ST 3.

In comparing the total execution times we observed that with a fast CPU the "Stripes" algorithm with the transposed organization performs better than the relational organization, provided the number of active columns is less than or equal to approximately 50. This percentage decreases as we consider slower CPU's. In fact as the CPU gets slower, the total time of all the algorithms tend to the computation time. Moreover, with the transposed organization and any CPU speed the "Vector Times Matrix" and the "Vector Building Block" algorithms will perform better than the relational, only if the active columns are less than 10% of the total number of columns.

Therefore the general conclusions of this paper are (1) the "Stripes" algorithms perform better than the building blocks algorithms by an order of magnitude. (2) if it is anticipated that a substantial percentage of the columns will be

[3] where k is taken to be 1 for VBB 1 and VTM 1

active it could be preferable to consider the relational organization. (3) increasing the memory size does not always improve the total execution time. In fact the curves of the total times for the "Vector Building Block" and "Vector Times Matrix" algorithms as functions of the main storage size, possess minimums.

REFERENCES

[BATE82] Bates, D., Boral, H., and D. DeWitt, "A Framework for Research in DAtabase Management for Statistical Analysis," Proceedings of the 1982 SIGMOD International Conference on Management of Data, June 1982.

[BATO79] Batory, D. S., "On Searching Transposed Files," ACM TODS Vol. 4, No. 4, pp. 531-0.25i44, December, 1979.

[BURN81] Burnett, R. and J. Thomas, "Data Management Support for Statistical Data Editing," Proceedings of the First LBL Workshop on Statistical Database Managment, December, 1981.

[COON80] Coonen, Jerome,T., "An Implementation Guide to a Proposed Standard for floating-Point Arithmetic," COMPUTER, Vol. 13, Number 1, pp.68-79, January, 1980

[DONG79] Dongarra, J., Moler, C., Bunch, J., and G. Stewart, "Linpack Users' Guide," SIAM, 1979.

[DONG83] Dongerra, J. "Redesigning Linear Algenra Algorithms," Bulletin De La Direction Des Etudes Et Des Rescherches Serie C, No 1, pp. 51-0.25i9, 1983.

[EGGE81] Eggers, S.J., Olken, F., and A. Shoshani, "A Compression Technique for Large Statistical Databases," Proceedings of the 7th International Conference on Very Large Data Bases, France, 1981, pp. 424-434.

[GENT73] Gentleman, W. M., "Least Square Computations by Givens Transformations Without Square Roots," J. Inst. MAths Applics 12, pp.329-336, 1973.

[IBM 77] IBM, "Reference Manual for IBM 3350 Direct Access Storage," GA26-1638-2, File No. S370-07, IBM General Products Division, San Jose, California, April, 1977.

[RAY82] Ray,A.,A., SAS Users's Guide: Statistics, 1982 Edition, SAS Institute Inc., Cary, NC

[TURN79] Turner, M.J., Hammond, R., and Cotton, P., "A DMBS for LArge Statistical Databases," Proceedings of the 5th International Conference on VLDB, pp. 319-327, 1979.