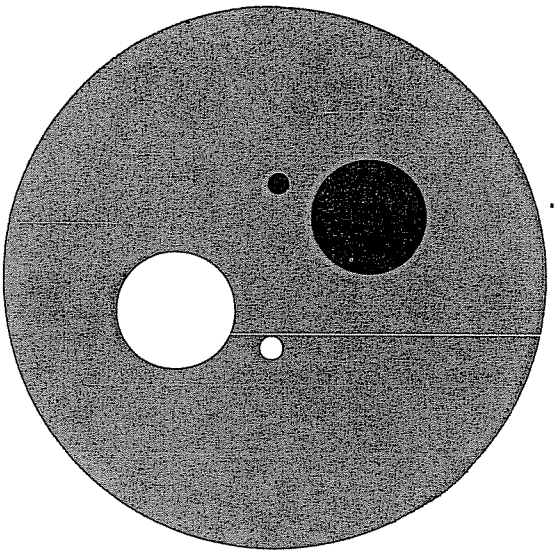
COMPUTER SCIENCES
DEPARTMENT

University of Wisconsin-
Madison



A BUILDING BLOCKS APPROACH TO STATISTICAL DATABASES

by

Setrag Nishan Khoshafian

Computer Sciences Technical Report #543

May 1984

A BUILDING BLOCKS APPROACH TO STATISTICAL DATABASES

by

SETRAG NISHAN KHOSHAFIAN

A thesis submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy
(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN - MADISON

1984

ABSTRACT

In recent years, considerable attention has been given to various data management issues for statistical databases. However, most of the research in statistical databases has concentrated on retrieval, sampling, and aggregation type statistical queries. Data management issues associated with the complex and computational statistical queries have been ignored.

This dissertation endorses an integrated approach to statistical databases. In an integrated system, the database management software supports both data retrieval and computational queries. As a first step towards an integrated system we identify and analyze the performance of three important statistical computational methods: X'X, the QR decomposition, and the Singular Value Factorization.

Our performance evaluation compares two alternative secondary storage organizations: the transposed and relational organizations. We also propose several implementation strategies for each computational method. The alternative implementations correspond to vector building block, vector-matrix, and direct algorithms. We develop buffer management algorithms for each implementation strategy, and, evaluate corresponding I/O and total execution time cost functions in terms of data and system parameters for a general system architecture. The variable parameters in our performance curves are the number of active columns, the size of main storage, and the time per floating point operation. We also propose special purpose multiprocessor architectures for each computational method. A multistage shuffle/exchange interconnection network (the ZETA network) is proposed for X'X. A multidisk/multiprocessor organization is proposed for the QR decomposition, and a cyclic shift multiprocessor interconnection is proposed for the Singular Value Factorization.

Three main conclusions which can be drawn from our extensive performance evaluations of X'X, the QR decomposition, and the Singular Value Factorization. The first conclusion is that integrated statistical database management systems which support the computational methods through vector buiding blocks or vector-matrix operations have significantly inferior performance compared to direct implementations of the computational methods. The second conclusion is that for those statistical computational methods whose algorithms involve an iterative decrease in the number of active columns, the preferable underlying storage structure is the fully transposed secondary storage organization. Finally, our results indicate that developing a general-purpose statistical database machine appears to be a difficult task.

ACKNOWLEDGEMENTS

My foremost gratitudes go to my co-advisors Professor David DeWitt and Professor Doug Bates. Professor David DeWitt provided me with generous support, excellent direction, and most valuable advice. Professor Doug Bates helped me enormously with the intricacies of statistical computational methods, and complemented the support, direction, and encouragement I needed to pursue my research. I felt very privileged to work with both of them, and I enjoyed th research environment very much. I would like to also express my gratitude to Professor Olvi Mangasarian for helping me in my preparation for the Mathematical Programming qualifiers, as well as his suggestions on an optimization problem in my dissertation.

Finally, special thanks to my wife Silva, for her patient endurance and irreplaceable support during these past three years.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1. Statistical Databases

Statistical databases are databases constructed for the primary purpose of statistical analysis. In recent years considerable attention has been given to the characterization of data management issues for large statistical databases. This is illustrated by the number of papers (e.g. [TURN79, SHOS82, DENN83, ROWE83] etc.) and projects (e.g. the ALDS project [THOM83], SEEDIS [McCA82], SAM* [STAN83] etc.) which have attempted to identify some of the data access, retrieval, and management characteristics of large statistical databases. Some of these research efforts have proposed and sometimes implemented innovative solutions to non-conventional database management issues. The bi-annual "International Workshop"s on statistical database management[1] have provided a means of exchanging research results by the computer scientists and statisticians interested in the area.

The characterization and the proposed solutions for statistical database management systems can be classified into two interdependent categories:

(1) those pertaining to the statistical data sets themselves

(2) those pertaining to the manipulation of the statistical data sets, or, statistical analysis.

---

[1] The first one held in December of 1981 at Menlo Park, California, and the second one in September of 1983 at Los Altos, California

For the first category, it has been observed that statistical data sets tend to be large and complex. Statistical data sets inevitably contain several types of missing observations. Structured missing data (or sparse data) are also common in large statistical databases. Another characteristic of statistical data sets is the distinction between category and summary attributes. Category attributes span a small range of values. Usually a combination of category attributes serve as a composite key into the numeric, or summary, attributes. Finally, it has been observed that statistical data sets tend to be more or less static. Observations over large samples do not tend to change as often as transactions for corporate databases. This is also a characteristic of statistical analysis.

It has been observed that the process of statistical analysis passes through two phases. The **exploratory** phase and the **confirmatory** phase [BORA82, TUKE77]. In the exploratory phase the analyst attempts to obtain a "feel" for the data set by editing and browsing through the database, extracting samples and subsets and performing some initial hypothesis testing. From a data management point of view, the problem is the generation and the dynamic maintenance of these samples and subsets. Another problem is maintaining "dictionary" information, or metadata, about the samples and the operations. Later, after this initial exploratory phase, the analyst tries to "confirm" the hypotheses over larger data sets. In this confirmatory phase, the entire data set is manipulated. However, it has been observed that most of the observations and only a few of the attributes are retrieved in this phase.

There have been several interesting solutions proposed for these types of problems. For example, since statistical operations process most of the observations and few of the attributes, transposed files [TURN79] have been proposed as an alternative underlying secondary storage organization. Different encoding and compression techniques have been proposed for the category attributes and for repetitive missing observations [EGG81]. Alternative semantic

models have been proposed to capture the distinction of the category and summary attributes (e.g. SAM* [STAN83], GRASS [RAFA83], SUBJECT [CHAN81]). It is generally agreed that metadata should be referenced as ordinary data and should be maintained and updated automatically [McCA82]. This is a brief summary of the various solutions to the characteristics and problems of statistical databases.

However, these characteristics problems are primarily based on retrieval, sampling, and aggregation type of statistical queries. The data management issues of more complex, computational, or analytical statistical queries (such as multiple linear regression, analysis of variance, canonical correlation etc.), have not been considered in current research work in statistical databases. The underlying assumption here has been that computational or analytical statistical queries are handled by statistical packages rather than by the statistical database management systems. Therefore, for these type of queries the statistical database management system provides and maintains interfaces to one or more statistical packages. Then the user is able to interface with the statistical package. A case in point is the the ALDS project [THOM83]. The data sets and the meta-data are organized in self describing transposed files and the system provides an interface to the statistical package Minitab [RYAN76].

There are four main problems with the interface approach to statistical database management. First, as pointed out in [BORA82], the data sets manipulated by a statistical package cannot exceed the virtual memory size of the machine. Second, the buffer management is that of the operating system, which often uses a global buffer management strategy and does not take into consideration the particular data access patterns of the statistical operations. Third, algorithms which implement the operations do not attempt to minimize the secondary storage over-

head.[2] Finally, the data sets which are manipulated by the statistical package must be copied into the workspace of the statistical package, processed and, sometimes, the resulting data sets must be copied back into the database. In other words, there will be a substantial amount of extra overhead with the interface approach.

The alternative, of course, is to have an integrated system and let the statistical database management system support both the data retrieval and computational queries. This dissertation endorses the integrated approach and characterizes the data management issues of the statistical computational operations. The primary focus of the dissertation is the identification and the performance evaluation of three important computational operations: X' X, the QR decomposition and the Singular Value Factorization. The performance evaluation considers two alternative secondary storage organizations: the transposed and relational organizations and several alternative implementation strategies. The alternative implementations correspond to vector building block, vector-matrix, and direct algorithms for the computational operations. Special purpose multiprocessor/multidisk architectures for these operations are also proposed.

1.2. Organization of The Dissertation

This dissertation is organized as follows: Chapter 2 presents a motivation and explanation for the building blocks approach. A justification is given for the choice of the three operations: X' X, QR and SVF. We have identified several alternative implementations for each computational method. These alternatives can be categorized at three abstraction levels: Vector Building Block, Vector-Matrix. and Direct. These three alternative implementation strategies will be discussed in detail in Chapter 2.

---

[2] For example, in LINPAK [DONG79], the algorithm for the QR decomposition utilizes the dot-product and "axpy" vector operations, and, as we shall demonstrate, this type of implementation involves the greatest number of page transfers

Chapter 3 introduces the two alternative secondary storage organizations that we have evaluated: the relational and transposed. Chapter 4 presents the system and data model for the performance evaluation. Chapter 5, 6, and 7, are dedicated to the analytical evaluation of the operations X'X, QR, and SVF. For each operation, high level algorithms as well as several alternative concurrent algorithms are presented. The "concurrency" in this context specifies the parallel execution of the I/O and processing subsystems. Finally in Chapter 8, we present our conclusions and suggest a number of future research directions.

CHAPTER 2

BUILDING BLOCKS FOR STATISTICAL OPERATIONS

2.1. Introduction

An important issue in the development of special purpose database management systems is the specification of a set of primitive data structures, data retrieval techniques, and operations to construct a functionally complete[1] system. A major goal in this specification is to introduce considerable improvements in the overall system performance. Therefore, at one level, the system developers must try to specify a set of primitives which form a functionally complete system. At a lower level, decisions must be made in order to select the data structures and the algorithms to implement and support the complete set of operations.

A case in point is relational algebra. The relational algebra model is defined through the operations union, set difference, cartesian product, projection and selection [CODD70]. Other relational algebra operations, such as join, can be expressed in terms of these "building blocks." However, an implementation of a relational algebra query language will probably implement the join through more efficient and direct algorithms rather than through the cartesian product, select and project operations of which the join consists. Moreover, although supporting the commonly used join operation directly is the correct approach from a performance viewpoint, at the lower level the designers of the database management system must decide on the algorithm(s) which implement the join operation (e.g. nested loops, merge-sort etc.).

---

[1] By a "functionally complete" system we mean all the queries and and primitives of the special purpose DMBS are supported.

Unfortunately, as we shall show, for statistical database management systems the notion of a functionally complete system does not exist. Typically, to support statistical queries either an existing database management system is augmented with a set of aggregate functions (e.g. [KLU81]), or a special purpose statistical database management system is constructed (e.g. SEEDIS [McCA82]). Although some of the proposals attempt to support the envisioned retrieval, sampling, or aggregation type statistical queries, it is not clear if any system will be able to support all the possible type of queries. The problem is that, unlike the relational model, there does not exist a well developed theoretical model of statistical database management systems. This problem is further aggravated if the computational queries are also supported by the database management systems.[2]

2.2. Integrated Systems

The primary difficulty with the integrated approach is the large (and growing) number of techniques that a statistician can choose from when analyzing a data matrix X. A statistician's set of tools includes a variety of regression techniques (linear, stepwise, robust, ridge, iterative reweighted, non-linear), analysis of variance and covariance, canonical correlation, principal component analysis, discriminant analysis, etc. Moreover, better and alternative analytical techniques are continually developing. Therefore it is difficult to define a "closed" or functionally complete statistical database management system.

Faced with the formidable problem of the large number of analytical techniques, we took a simplistic approach. We decided that the first step towards an integrated system was the detailed analysis of alternative computational methods for one important analytical technique.

---

[2] As indicated in Chapter 1, most existing statistical database management systems interface with one or more statistical packages

We chose to concentrate on the method of fitting linear statistical models. This is also called multiple linear regression analysis. While fitting linear models is the mainstay of statistical analysis, it may seem an oversight to neglect other methods that have been or may yet be developed. However, most other statistical models utilize linear models as a basic step in iterative fitting procedures. The Generalized Linear Models of Nelder and Wedderburn [McCU83], logistic regression models, and nonlinear regression models all utilize the linear models calculations as a basic iterative step. It should also be noticed that the statistical area called analysis of variance (and analysis of covariance) is also part of linear regression analysis.

There are a number of alternative computational methods for multiple linear regression. However, one or the other of the following two approaches is generally used: either the symmetric matrix X' X is formed [SEBER77, KENN80] and the analysis proceeds with this matrix, or the matrix X is decomposed and the analysis proceeds with the factors of X. The most common decompositions of a data matrix X are the upper triangularization of X (through Givens rotations or Householder transformations [GOLU73, SEBER77, KENN80, DONG79]) and the Singular Value Decomposition [STEW73, DONG79, WILL71 , KENN80, CUPP81, MAND82]. Therefore, a list of matrix operations which covers the evaluation of X' X and the most common decompositions of X will provide the computational tools of the multiple linear regression statistical technique. The statistical technique can be viewed as an application program.

2.3. Implementations at Three Abstraction Levels

As mentioned in the introduction of this Chapter, we should specify the set of operations and primitives that must be supported by a special purpose database management system. We need also to consider the data structures and the algorithms to implement and support the opera-

tions. We gave the example of alternative and direct implementations of the join operation in relational algebra. The main issue here is performance.

The solutions (e.g. transposed files, compression techniques etc.) proposed for the retrieval and aggregation type statistical queries have been primarily motivated by performance issues. Even the graphical logical models (e.g. SUBJECT [CHAN81]) imply substantial savings in data storage. These logical models are sufficient to store the category attributes only once. Considering that in many applications summary values for the cross product of the category attribute values exist, the savings in data access and storage are substantial.

For the computational methods which we have considered (namely, X' X, the QR decomposition, and the Singular Value Factorization), the data retrieval and update problems arise because of the data sets' large volumes. It could be argued that sampling and subsetting will reduce the data size and, hence, eliminate the data retrieval and update problems of the complex computations. Although this could be true for the exploratory phase of the analysis, the purpose of the confirmatory phase is to apply the hypothesis testing to the cleansed version of the complete database. Moreover, if the samples are large, the same data management problems could be introduced into the exploratory phase of the analysis.

This is the first time that the secondary storage problems of the computational methods X' X, the QR decomposition, and the Singular Value Factorization have been considered. However, other research studies have attempted to identify the data management issues of different classes of computational operations. We shall briefly describe two examples. The first example is presented in [BELL83]. This paper characterizes several storage structures that are used in solving weather forecasting and other types of prediction methods. The storage structures are determined according to the "update dependencies" which occur while solving these prediction models. It is suggested that the records of the underlying data files be structured according to

one of the categories identified in the analysis. A second example with a similar approach is presented in [PERR81]. Here the problem is the solution of triangular or banded systems of linear equations. The author proposes a number of secondary storage methods and parallel algorithms. He compares his algorithms with the execution of the same operation with a virtual memory system (with a LRU page replacement strategy). We believe both these studies are in the right direction and attempt to provide "optimal" solutions for the secondary storage retrieval and update problems of a particular type of scientific database application.

In this dissertation we have tried to analyze the performance of three important computational methods commonly used in multiple linear regression, with respect to three abstraction levels. Below we identify these three abstraction levels for implementing the computational methods and briefly describe the relative merits of each.

2.3.1. Vector Building Block

The first level attempts to implement the computational methods through vector operations only. The idea is that it is possible to express the computational methods as a set of vector operations. Once these vector operations are identified, efficient implementations of the vector building blocks will correspondingly yield efficient implementations of the computational methods. For example, the computational method $X'X$ can be implemented as a set of inner products.[3] Therefore, an efficient implementation of the inner product vector building block will yield an efficient implementation of $X'X$. The advantage of this approach is that a relatively small set of vector operations will enable us to implement all the computational methods. Moreover, alternative and novel computational methods will, most probably, be supportable through existing vector building blocks. The disadvantage is the performance, since the set of vector

[3] the inner products of the columns of X

operations which implements a computational method is completely unaware of the computational method's overall execution.[4] As we shall see in subsequent chapters, in some cases the performance degradation will become intolerable.

2.3.2. Vector-Matrix

The approach of the second level is intermediate between the approaches for levels one and three. With the vector-matrix approach, the computational methods are implemented through vector-matrix operations. The multiplication of a vector with a matrix and Householder transformations are two examples of vector-matrix operations. For example, the non-diagonal elemements of X' X can be evaluated through p - 1 vector-times-matrix operations, where p is the number of columns of X. As we shall see, in most applications a vector-matrix implementation of a particular matrix operation will involve considerably fewer page transfers than a vector building block implementation of the same operation. The same objection, however, which was raised for the vector building block approach still holds. Although the performance degradation will be less serious, it is still feasible that a more direct implementation of a computational method will have better performance than an implementation through vector-matrix building blocks.

2.3.3. Direct Algorithms

The approach of the third level is to discover the most efficient direct implementations of the computational methods considered. With this approach there will be one implementation per operation. The goal is to develop optimal algorithms: optimal in secondary storage

[4] for X' X, a vector building block implementation will reference each column of X for each inner product. However, a better algorithm will buffer the columns (or blocks of the columns) which are needed in subsequent computations.

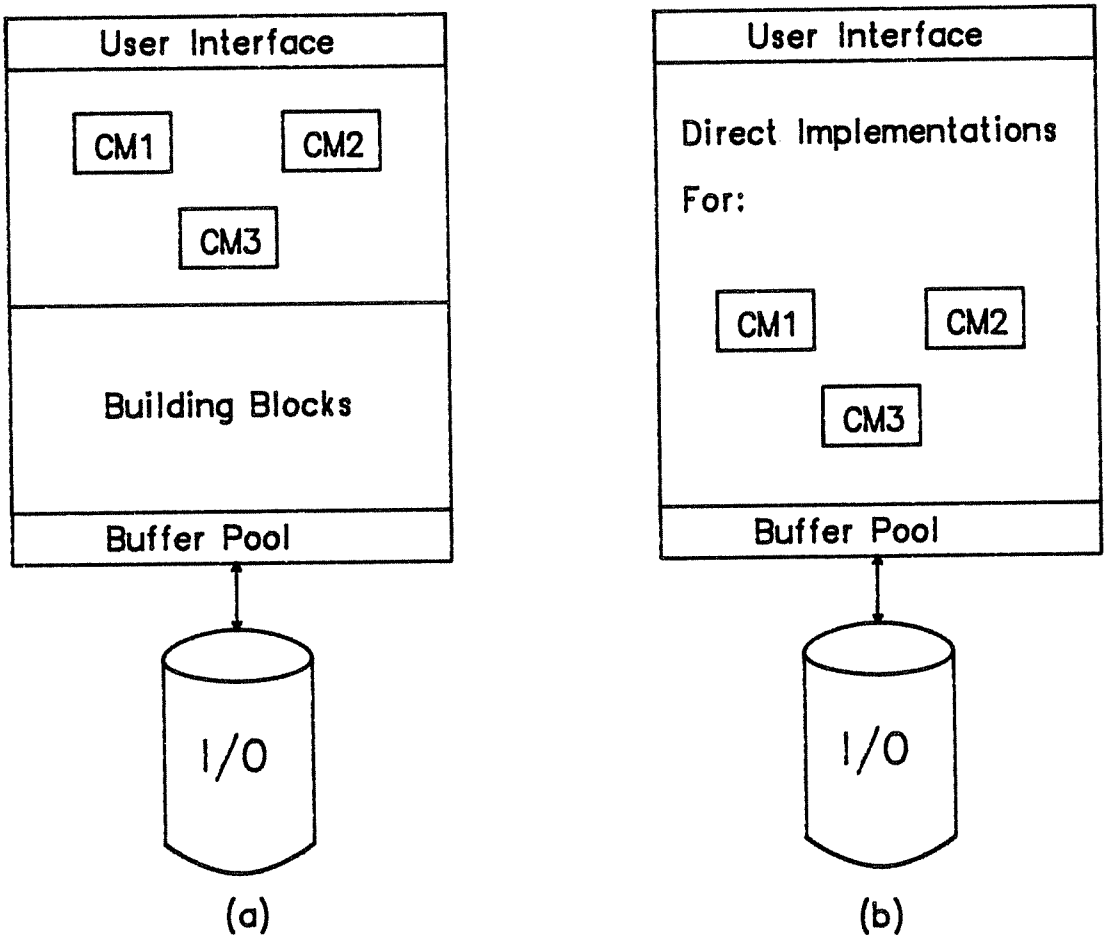

User Interface
CM1
CM2
CM3
Building Blocks
Buffer Pool
I/O
(a)
User Interface
Direct Implementations
For:
CM1
CM2
CM3
Buffer Pool
I/O
(b)

Figure 2.1

reference and also in overall total execution time. The advantage of this approach is clearly in performance. It is definitely desirable to have direct implementations for large data sets. However, there must be a direct implementation for each computational method. If a new operation is introduced, the data base management system software must be extended to include a direct implementation of the new operation.

2.4. Summary

A principal objective of this dissertation is to compare and analyze the performance of the three alternative implementation strategies. Figure 2.1 summarizes this objective. In (a), the computational methods CM1, CM2, and CM3 are supported through either vector building blocks or vector-matrix operations. In (b), CM1, CM2, and CM3 are implemented directly. It is assumed that any implementation of an algorithm performs its own buffer management, explicitly specifying the page replacement policy. Each algorithm is also aware of the underlying secondary storage organization of the data. The two alternative secondary storage organizations considered in this dissertation are the transposed and relational secondary storage organizations. Chapter 3 discusses and contrasts these two storage organizations.

CHAPTER 3

TRANSPOSED AND RELATIONAL STORAGE ORGANIZATIONS

One of the primary goals of this dissertation is to compare the performance of the computational methods with respect to two secondary storage organizations: the relational and the transposed. Most statisticians prefer to view their data as "flat files." Therefore the conventional relational secondary storage organization where the n-tuples of each relation are stored record-wise appears to be the natural choice. However, there are at least three reasons why such a secondary storage organization may not be suitable for storing a large statistical data set.

The first reason deals with data compression. Three properties of statistical data sets facilitate application of data compression techniques:

(1) Some of the attributes (such as the category attributes) of statistical databases span relatively small ranges of integer values.

(2) Different types of missing or "zero" values that are very common in statistical databases are obvious candidates for data encoding.

(3) It is typical in large statistical databases to have "clusters" of similar values. These clusters are either identical values or values in the same subrange.

Two strategies are commonly used for data compression in statistical databases: encoding and run length compression techniques [EGGR81]. The relational secondary storage organization is not suitable for data compression for the following reasons:

(1) In many systems consecutive data values must be aligned on byte boundaries. If a data set is stored as a relation when only two bits are needed to present the full range of values of an attribute, 75% of the storage space occupied by values of the attribute will be wasted [TURN79].

(2) Since missing and zero values occur frequently in statistical databases, a relational organization will need to support variable length records ([GEY83] discusses the implementation of such a scheme).

(3) It is not clear how the relational organization would implement compression techniques for clustered values.

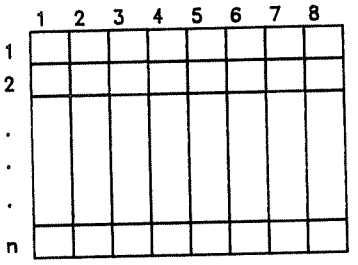
A second reason why a relational storage organization might not be a suitable organization for statistical databases, is that statistical operations usually encompass most of the observations and only a few of the attributes [KLEN83, SHOS82, TURN79]. For example, if all the rows and only 10% of the columns are processed, all the relation must still be retrieved. This implies the unnecessary retrieval of 90% of the relation from mass storage. For large databases this can have serious performance implications.

Finally, it is not uncommon in statistical databases to create or delete attributes. An example where such a dynamic modification occurs frequently is given in [GOLD83]. Relational (or corporate) databases on the other hand are tailored to tuple operations (insertion, deletion, and modification of tuples). Most relational database systems (e.g. INGRES, DB-2, IDM 500) do not allow dynamic creation and deletion of attributes.

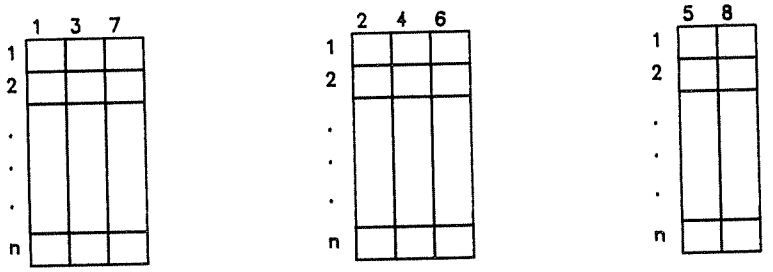
These drawbacks suggest an alternative storage organization, namely the transposed file organization. Transposed file organizations are commonly used in statistical databases. For example RAPID [TURN79], the ALDS project [BURN81], IMPRESS [MEYE69], and PICKLE [BAKE76] all use the transposed file organization. The two types of transposed file organizations are: (1) partially 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. The attributes in each subset are stored together.[1] The attributes which are stored together are also, hopefully, those that are accessed together. The problem of pre-determining a "clustering" scheme for the attributes is difficult. In fact in [BATO79] it is proven that the problem is NP-complete. If each attribute subset contains exactly one attribute we get the completely transposed organiza-

---

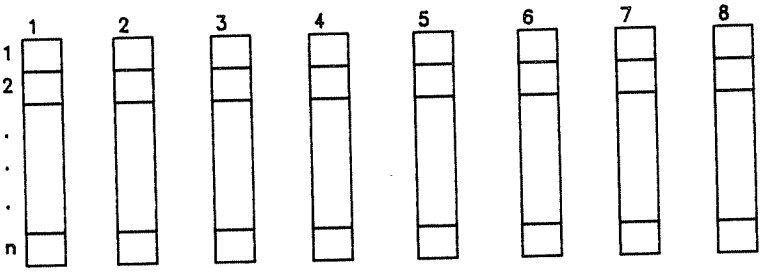
[1] That is, a collection of n-tuples is stored as k collections of n_i-tuples where $n_1 + n_2 + \cdots + n_k = n$.



Relational



Partially Transposed



Fully Transposed

Alternative File Organizations

Figure 3.1

tion.

In Figure 3.1, the relational, partially transposed, and fully transposed organizations of a data matrix with 8 attributes and n tuples are shown. For the relational secondary storage organization, the data file is paginated and each page contains a (fixed) number of tuples of the data matrix. Each tuple contains corresponding elements of all of the attributes. It is assumed that the data pages of a data matrix are stored contiguously on the secondary storage medium. For the partially transposed secondary storage organization, the layout is identical, except that each page of a partially transposed file contains a fixed number of subtuples from a partition of the attributes. For example in Figure 3.1, the first partially transposed file contains subtuples from attributes 1, 3, and 7 only. With the fully transposed secondary storage organization, each page of the fully transposed file contains elements from one attribute only. Again we shall assume that the pages of a partially or fully transposed file are stored contiguously on the secondary storage medium.

In Figure 3.2, we have examples of paginated relational, partially transposed, and fully transposed files. It is assumed that all the elements are of equal size and each page contains 96 elements. Note that if we are interested only in attribute 1, retrieving a page with the relational secondary storage organization will yield 12 elements of the attribute. With the partially transposed file we will obtain 32 elements of attribute 1 and with the fully transposed secondary storage organization 96 elements of attribute 1 will be retrieved.

There have been a number of studies analyzing transposed files for relational queries. In [HOFF76] for example, a non-linear integer programming model is proposed to determine the partitioning of the attributes for a particular mix of queries. The model assumes the availability of the selectivity factor for each attribute. For a particular set of system and data parameters (and assuming the selectivity factors are given), the problem could be solved through a branch

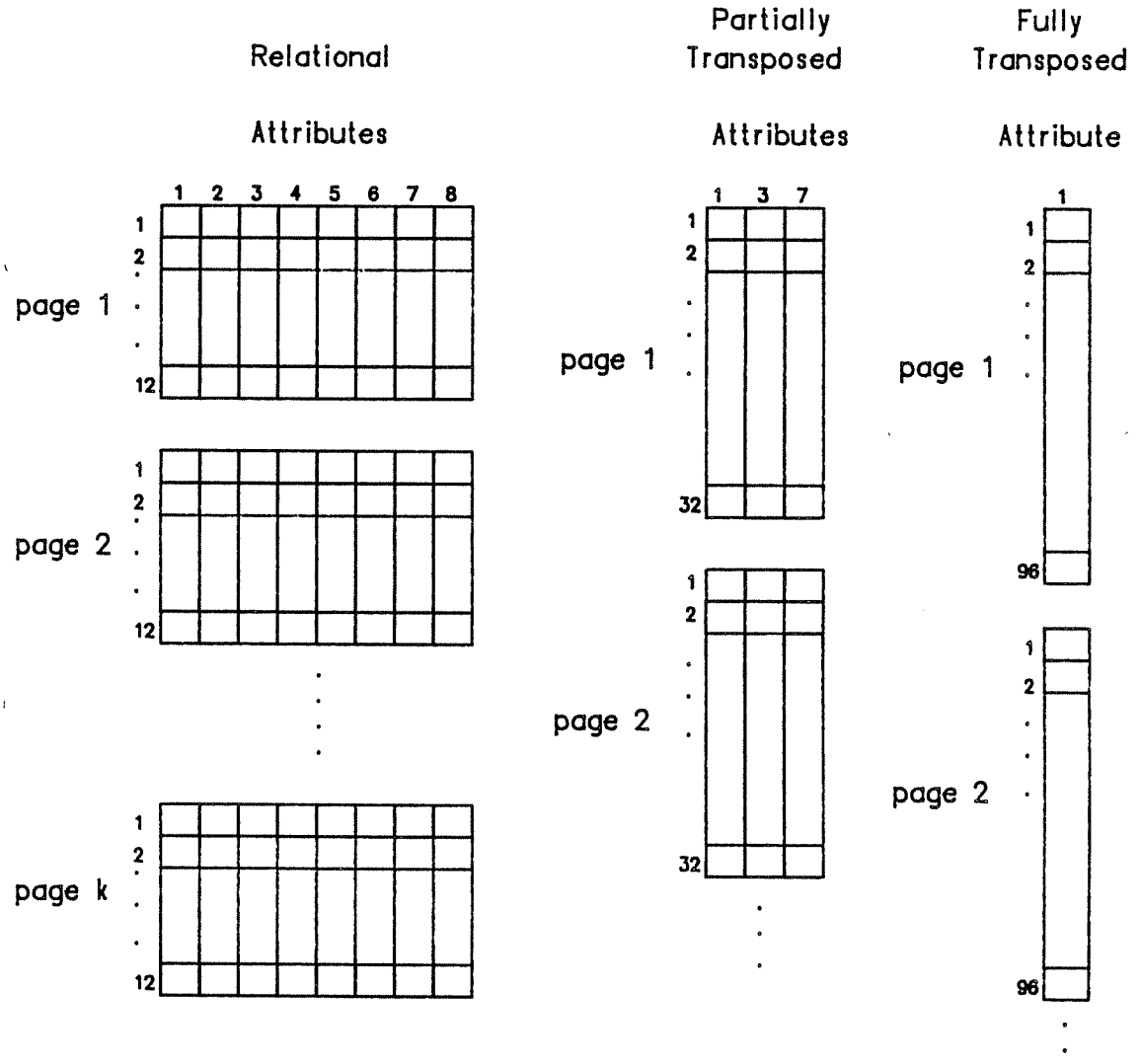

Relational
Attributes
1 2 3 4 5 6 7 8
page 1
page 2
page k
1
2
12
Partially
Transposed
Attributes
1 3 7
page 1
page 2
32
Fully
Transposed
Attribute
1
page 1
page 2
96

Figure 3.2

and bound algorithm. Another study [BATO79] evaluates closed form cost equations for selection queries with a completely transposed organization. Batory also proposes a heuristic algorithm for determining the search sequence of the transposed files of the query attributes. Both studies assume the selectivity factor of each attribute is known.

However, there have been no analytical studies comparing the performance of the relational and transposed secondary storage organizations for operations on statistical databases. In this dissertation we have developed cost equations for the total number of page transfers, the total I/O time, and the total execution time for both the relational and fully transposed secondary storage organizations. One of the varying parameters in our study has been the number of "active" columns. The active columns are the columns considered by the analyst. For example, although the original data matrix of observations might contain 100 attributes, the analyst might be interested in a least squares fitting model with five of the attributes. The remaining 95 attributes are ignored and the 5 attributes constitute the active columns.

Some of the computational methods (such as the QR decomposition and the SVF) iteratively update the active columns. In these cases the active columns are first copied and the matrix transformations are subsequently applied to the copies. The original data matrix X is kept intact. If this original matrix X is stored as a relation, a partially transposed file of the active columns is constructed first. The operations are then applied to this file. If the matrix is stored fully transposed, the updated columns are also stored fully transposed. In the following Chapters "transposed" will stand for fully transposed, unless stated otherwise.

CHAPTER 4

SYSTEM AND MODEL PARAMETERS

This Chapter introduces the system and data model parameters used in the cost equations of the algorithms. Section 4.1 presents the I/O and CPU parameters and Section 4.2 lists the data model parameters. Section 4.3 summarizes the overall scheme of the performance evaluations.

4.1. I/O and Processing Subsystem Parameters

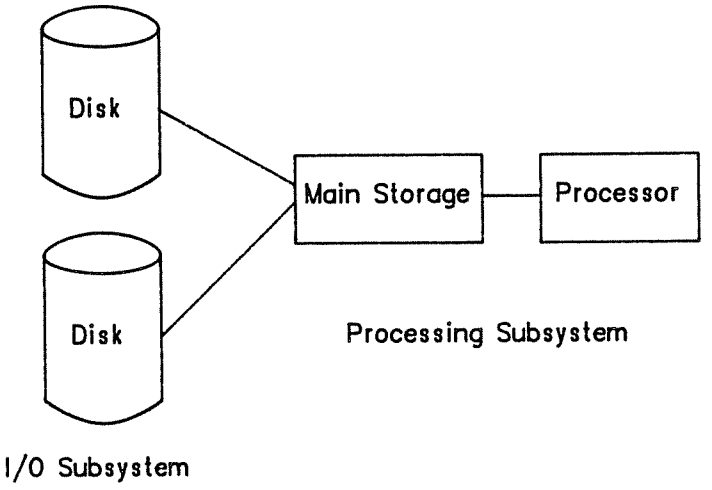


Figure 4.1

Figure 4.1 is a simple diagram of the system model used throughout the dissertation. The system consists basically of an I/O subsystem and a processing subsystem. The I/O subsystem is

composed of one or more magnetic disk drives which have direct access to the main storage of the processing subsystem. The processing subsystem is composed of a main storage module[1] and a processor dedicated to numeric calculations. The key point here is the simplicity of the underlying architecture. Our objective is to obtain parametrized cost equations for a general architectures. This would enable the designers of the database management system to easily modify the parameter values for their particular architectures.

4.1.1. Disk Parameters

The basic 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); (5) the cylinder-to-cylinder seek time (Tsk) ; and (6) the cylinder-to-cylinder move time after an initial move (Tmv). That is, the time to traverse K cylinders is Tsk + K·Tmv. The values of the disk parameters are those of the IBM 3350 disk drive [IBM 77]. 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.

---

[1] which has a bandwidth high enough to support the concurrent access by the processor and I/O subsystems

BSIZE	page size	19069 bytes
DCYL	number of pages per cylinder	30 pages
Tio	page read/write time	25.0 ms.
Tdac	average access time	25.0 ms.
Tsk	time to seek 1 track	10.0 ms.
Tmv	time to move to next track after an initial move	0.07 ms.

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

$$T_{r-io}(U) = Tdac + \left\lfloor \frac{U}{DCYL} + 0.5 \right\rfloor \cdot Tsk + U \cdot Tio$$

We shall assume that for those operations that only retrieve (or read) data, there exists exactly one disk drive. On the other hand, for those operations which produce temporary files we shall assume the existence of two disk drives. One disk drive will be dedicated to the "read" operations and the other disk drive will be dedicated to the "write" operations.

4.1.2. CPU Parameters

The arithmetical operations of the computational methods either use the basic step of an "Inner Product" or the basic step of an "AXPY" function in their innermost loops. These basic steps are given by:

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

AXPY : $y_i = a \cdot x_i + y_i$

The "Inner Product" function accepts as arguments two vectors and yields a real number. The "AXPY" function accepts as arguments two vectors, X and Y, and a scalar. It then yields a

transformed vector Y. As indicated earlier the transformed Y does not replace the old value of Y, but is stored elsewhere.

We have taken the unit of execution in the basic step for both of these functions equal to a Tflop, which is the time of a "floating point operation" (a "flop") [DONG79]. A Tflop involves the execution time of a double precision floating point multiplication, a double precision floating point addition, some indexing operations and memory 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. Therefore, in the cost equations we shall use the functions T_{AXPY} and T_{INN}. In this dissertation these functions are assume to be identical. For n floating point operations we have:

$$T_{AXPY}(n) = T_{INN}(n) = n \cdot Tflop$$

4.2. Data Model Parameters

Since the relational and the transposed organizations are being compared it is imperative to define the parameters of each. The data set is assumed to be dense with 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. 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 relational organization and the transposed organization the data matrix will occupy 10,000 pages. These parameters are as follows:

R	number of pages occupied by the data set X	10,000 pages
N	number of pages for a column(transposed)	100 pages
w	number of attributes	100
n	number of tuples/observations	230,000
q	number of elements per page	2,300
p	number of active columns for the matrix operations the values considered are 4 to 100 active columns	
M	the size of the main storage the values considered are 10 to 200 pages	

4.3. Summary

In Chapters 5, 6 and 7 we shall propose a number of algorithms for the computational methods. An underlying assumption for the system has been the capability of the concurrent execution of the I/O and processing subsystems.

Figure 4.2 summarizes the main features of our work. First, we have identified three important computational methods: X' X, the QR decomposition, and the Singular Value Factorization. Second, the performance of these computational methods is studied with respect to both the relational and transposed secondary storage organizations. Third, we have considered and analyzed vector building block, vector-matrix, and direct implementations for the transposed secondary storage organization. For the relational secondary storage organization we have considered only direct implementations. Fourth, we have proposed alternative algorithms at each abstraction level.

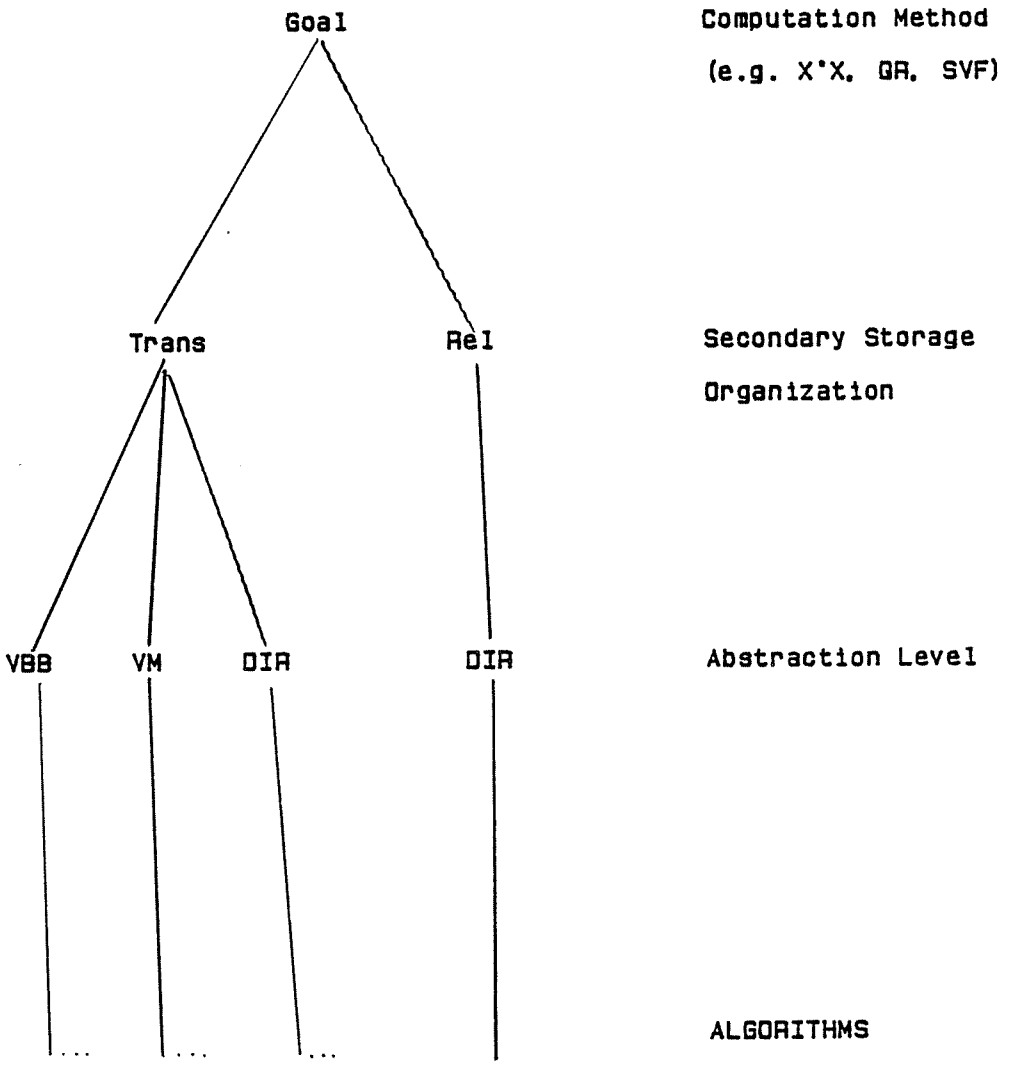

Goal
Computation Method
(e.g. X'X, QR, SVF)
Trans
Rel
Secondary Storage
Organization
VBB
VM
DIR
DIR
Abstraction Level
ALGORITHMS

Figure 4.2

The algorithms explicitly specify:

(1) The number of subdivisions of the main storage

(2) The page replacement strategy

(3) The concurrent execution of the I/O and processing subsystems

In the algorithms of the following chapters, the statement lists which are executed concurrently are preceded by a "$." For example in:

$ A := 0

For k := 1 to L

$$A := A + \frac{1}{k}$$

$ Read (Y, Z)

W := Y + Z

the evaluation of the series executes concurrently with the statement list which reads Y and Z, and sets W equal to their sum. If two statements in concurrent statement lists reference the same variable, we assume the variable is in a critical section. For our algorithms, there could be two or three concurrent statement lists. If there are two concurrent statement lists, one is executed by the I/O subsystem and the other by the processing subsystem. If there are three concurrent statement lists, the I/O subsystem consists of an input subsystem and an output subsystem. Therefore, the three concurrent statement lists are executed by the processing subsystem, and each of the input and output I/O subsystems.

CHAPTER 5

X′X

In this Chapter we present several schemes for evaluating X′X. Section 5.1 discusses the alternative vector building block, vector-matrix, and direct algorithms. Section 5.2 presents the closed form cost equations and the performance evaluation of X′X. Section 5.3 summarizes the basic observations and conclusions of the performance evaluation and Section 5.4 introduces a special purpose multiprocessor interconnection network for evaluating X′X.

5.1. Algorithms

There are three high level algorithms for evaluating X′X. These correspond to the vector building block, vector-matrix and direct implementations of X′X and will be labeled VBB, VTM, and, ST respectively. In the vector building block (VBB) 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, as we shall see, it is the most I/O intensive of all three approaches. With the vector-matrix (VTM) 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 stripes (ST) algorithm involves the fewest page transfers, since here the active columns of X are read once. In this algorithm, after reading a "horizontal stripe" of the active columns, the partial inner products are accumulated before processing the next stripe.

VBB:

```
For i := 1 to p - 1     /* 1 .. p are the active columns */
    For j := (i + 1) to p     /* n is the number of tuples */
        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
```

In the following section we shall present buffer management algorithms for each of these high level algorithms.

5.1.1. Vector Building Block Algorithms

There are two buffer management strategies for the vector building block algorithm.

[1] -Divide the M pages of primary memory into three logical subdivisions: one of M - 2 pages and two of 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. Accumulate 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)
            Read ( X_i^(M-2,k) , X_j^((k-1)(M-2)+1) )
            For q := 1 to M - 2
                $ c_ij := c_ij + X_i^((k-1)(M-2)+q) . X_j^((k-1)(M-2)+q)
                $ if q < M - 2 then Read ( X_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
        Read ( X_i^((M/4),1) , X_j^((M/4),1) )
        For k := 1 to N / (M / 4)
            $ c_ij := c_ij + X_i^((M/4),k) . X_j^((M/4),k)
            $ if k < N/(M / 4) then Read ( X_i^((M/4),k+1) , X_j^((M/4),k+1) )
```

5.1.2. Vector Times Matrix Algorithms

We shall call that vector which multiplies the matrix consisting of the remaining active columns, 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. Accumulate 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)
        Read ( X_i^((M-2),k) , X_{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) . X_j^((k-1)(M-2)+q)
                $ if q < M-2 then Read ( X_j^((k-1)(M-2)+q+1) )
                  else if j < p then Read ( X_{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 Accumulate the inner product of the resident M / 3 page pairs while reading the corresponding M / 3 pages of the next column.

VTM 2:

For i := 1 to p - 12

Read ($X_i^{M/3,1}$)

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

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

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 Read ($X_{j+1}^{(M/3),k}$)

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

Read ($X_{p-1}^{(M/4),1}$, $X_p^{(M/4),1}$)

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 Read ($X_{p-1}^{(M/4),k+1}$, $X_p^{(M/4),k+1}$)

5.1.3. The Horizontal Stripes Algorithms - Direct Implementations of X'X

If we were to measure the effectiveness of an algorithm only by the number of page transfers, the horizontal stripes algorithm would obviously be optimal as 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, one can 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 that are already resident, while concurrently, reading corresponding M / p page blocks of the columns which have not yet been accessed.

To avoid the indexing and timing details we have introduced a function called "FindPair", 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 FindPair 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

Read ($X_i^{(M/p),k}$)

set (A [i])

$ Repeat

(i , j) := FindPair (A, D)

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

set (D [i,j])

DONE := DONE + 1

Until DONE = $\frac{p \cdot (p-1)}{2}$

[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 accumulated free the block occupied by the column.

The main difference between this algorithm 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 "Free", 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 "Find" 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 Find (M / (p + 1)) then Read ($X_i^{(M/(p+1)),k}$)

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

(i , j) := FindPair (A [q ,] , D)

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

count [i] := count [i]+ 1; count [j] := count [j]+ 1

If count [i] = (p - 1) then Free ($X_i^{(M/(p+1)),k}$)

if count [j] = (p - 1) then Free ($X_j^{(M/(p+1)),k}$)

set (D [i, j])

DONE := DONE + 1

Until DONE = $\frac{p \cdot (p-1)}{2}$

clear (A [q ,])

q := (q+ 1) mod 2

[3] -Divide the buffer space 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

ST 3:

Read ($X_1^{M/2p,1}$, $X_2^{M/2p,1}$, . . . $X_p^{M/2p,1}$)

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

Read ($X_1^{M/2p,k}$, $X_2^{M/2p,k}$ . . . $X_p^{M/2p,k}$)

5.2. 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 evaluate: (1) Total I/O cost (seek + transfer time); (2) Average execution time in overlap region; (3) Total execution time.

5.2.1. I/O Costs

Table 5.1 specifies 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.

Table 5.1

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

In Figure 5.1 we present 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 organi-

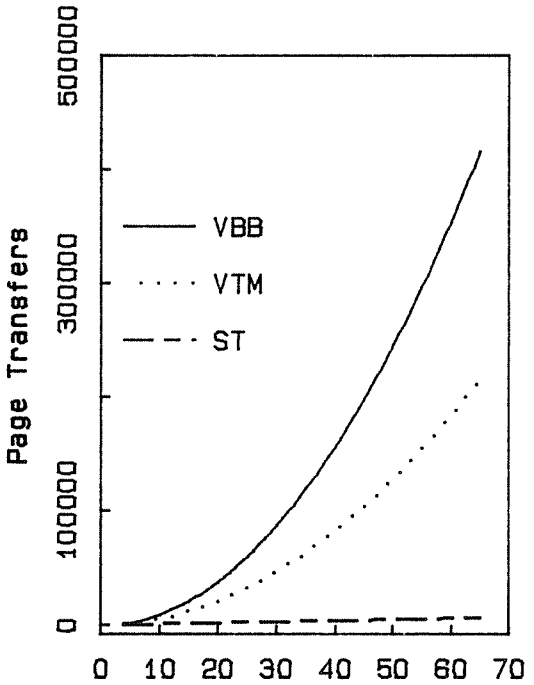


Fig. 5.1

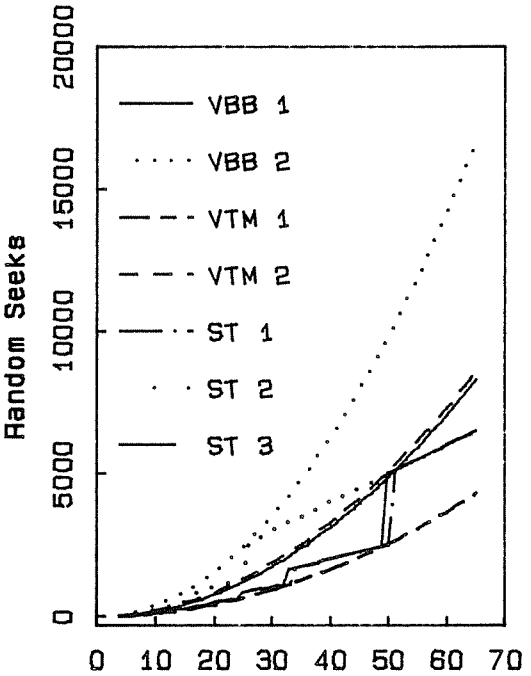


Fig. 5.2

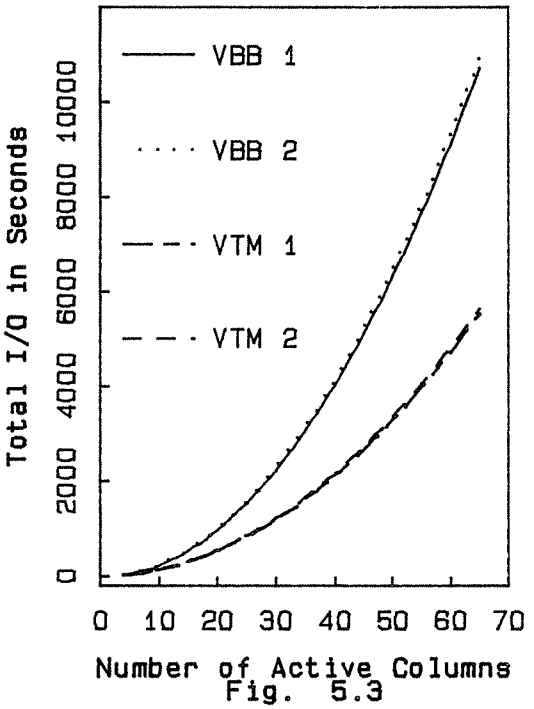


Fig. 5.3

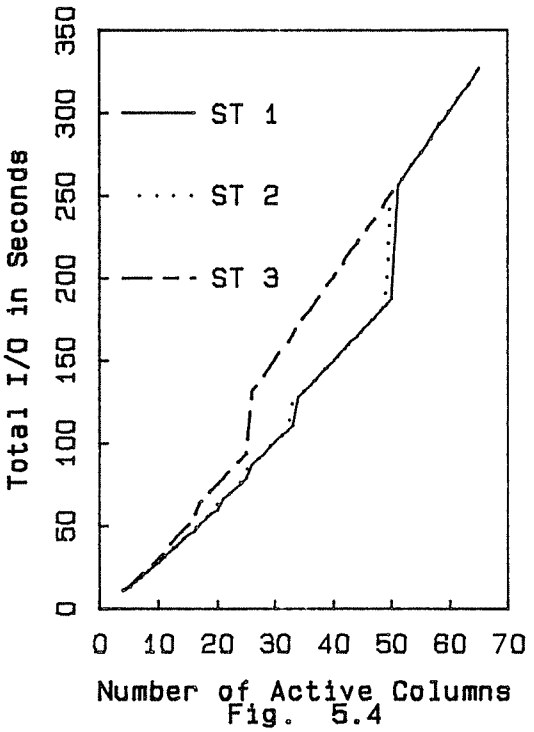


Fig. 5.4

zation (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).

As the secondary storage medium is a disk, another measure of the I/O is the number of random seeks. Figure 5.2 gives the number of random seeks as a function of the number of active columns. The main storage size is held at 100 pages (tracks). It is clear from Figure 5.2 that the first algorithm of the vector-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 random seeks of the other algorithms are between these two extrema. It is interesting that the number of random seeks for the stripes approach are comparable to the number of random seeks for the other approaches. The reason is that although the stripes algorithms reference the data matrix only once, the main storage is divided into O(p) subdivisions. As we noted earlier, the large number of random seeks is the consequence of accessing the transposed columns in smaller blocks.

In Figure 5.3 we have the curves for the total I/O time (transfer plus seek) as a function of the number of active columns. The curves are 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 100 pages. This figure clearly shows the effect of fewer page transfers for the vector times matrix algorithms. The first algorithm of the vector building blocks approach outperforms the second algorithm. This was to be expected, since the second algorithm involves a larger number of random seeks (see Figure 5.2). Figure 5.4 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 subdivi-

sions rather than (p + 1)) and therefore is slightly better. The breaks in the curves correspond to the same breakpoints as in Figure 5.2. In other words, these breaks are due to a substantial increase in the number of random seeks. For example, with M = 100, when p = 33 M / p is equal to 3. Since $\left\lceil \frac{100}{3} \right\rceil = 34$, the total number of random seeks is 33*34 = 1122. On the other hand when p = 34, then M / p = 2 and the total number of random seeks is 34*50 = 1700.

The total I/O for a direct implementation of X'X using the relational organization is approximately 253 seconds. The vector building block and vector matrix algorithms exhibit superior performance only when the number of active columns is less than 10 or 13 respectively. The direct implementations using the stripes algorithm remain superior to the relational organization until the number of active columns exceeds 50. This shows that seek operations constitute a substantial percentage of the total I/O time for the stripes algorithms.

In Figure 5.5 and Figure 5.6 the total I/O time (in seconds) is presented as a function 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 5.5 are due to the overhead of the track-to-track seeks. For example, if M = 40 there are no track to track seeks for the second algorithm of VBB. On the other hand when M = 45, there are three track-to-track seeks per column. The sharp edges of the curves in Figure 5.6 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.

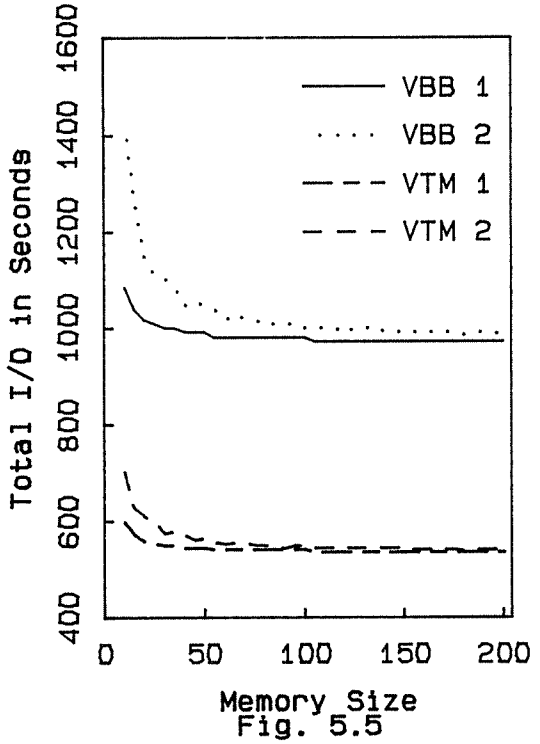

Total I/O in Seconds
400
600
800
1000
1200
1400
1600
0
50
100
150
200
Memory Size
VBB 1
VBB 2
VTM 1
VTM 2

Fig. 5.5

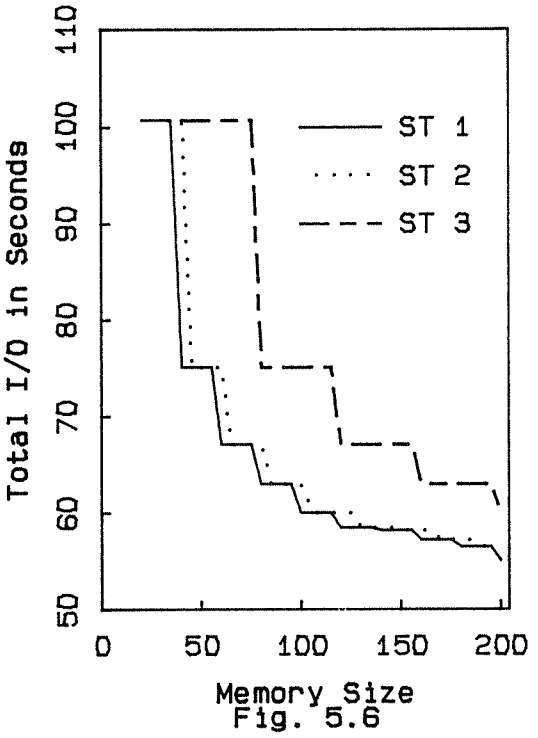

Total I/O in Seconds
50
60
70
80
90
100
110
0
50
100
150
200
Memory Size
ST 1
ST 2
ST 3

Fig. 5.6

5.2.2. Execution Time in the Regions of CPU-I/O Overlap

This section presents cost functions of the regions of CPU-I/O overlap. Each overlap time is the maximum of the I/O and computational times in the overlap region of an algorithm. Since ST 1 and ST 2 require some synchronization between the I/O and processing subsystems, we have not stated the overlap times for these algorithms.

$$T_{ol}^{(1,1)} = MAX\left(T_{INN}(q), T_{io} \right)$$

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

$$T_{ol}^{(2,1,r)} = MAX\left(T_{INN}(q), T_{r-io}(1) \right)$$

$$T_{ol}^{(2,1,s)} = MAX\left(T_{INN}(q), T_{io} \right)$$

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

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

5.2.3. Total Executions Times

The total execution time of an algorithm is the sum of three terms: $T_{start} + Q \cdot T_{ol} + T_{flush}$. T_{start} and T_{flush} correspond to, respectively, the initial I/O 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.

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

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

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

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

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

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

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

To evaluate the total execution times of ST 1 and ST 2, suppose that there exists an integer k such that:

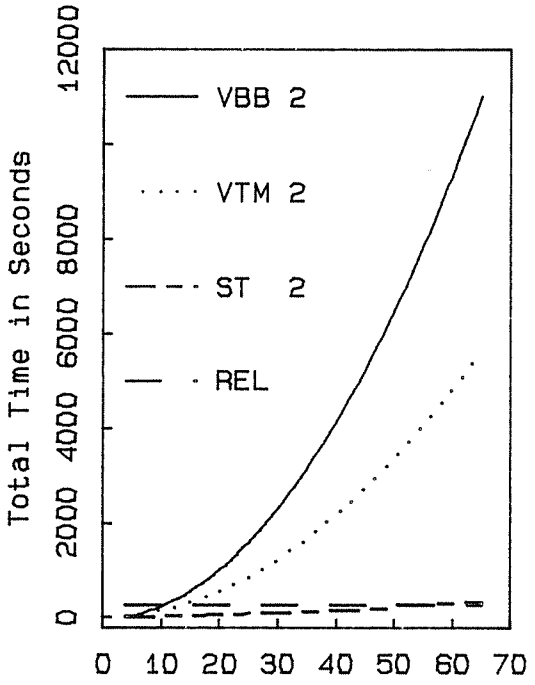
$$T_{INN}((k+1)\cdot U\cdot q)\ \geq\ T_{r-io}(U)\ \geq\ T_{INN}(k\cdot U\cdot q)$$

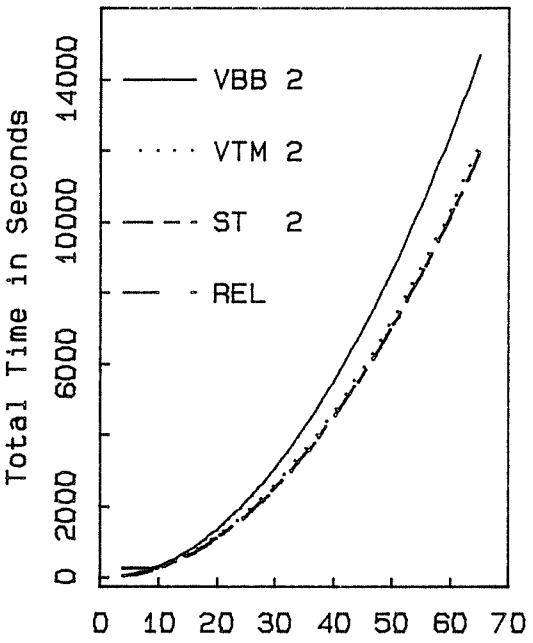
$U=\frac{M}{p}$ in ST 1 and $U=\frac{M}{p+1}$ in ST 2. Then the total execution time of ST 1 is given by:

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

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

For ST 2 we have two cases:


Total Time in Seconds
0
2000
4000
6000
8000
12000
VBB 2
VTM 2
ST 2
REL
0 10 20 30 40 50 60 70
Number of Active Columns
Fig. 5.7


Total Time in Seconds
0
2000
6000
10000
14000
VBB 2
VTM 2
ST 2
REL
0 10 20 30 40 50 60 70
Number of Active Columns
Fig. 5.8

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

$$\left((p-1) + \frac{(p-m-1)\cdot(p-m-2)}{2}\right)\cdot T_{INN}\left(q\cdot\frac{M}{p+1}\right)$$

$$+\ p\cdot\frac{N}{(M/(p+1))}\cdot T_{r-io}\left(\frac{M}{p+1}\right)$$

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 :

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

Finally, the total execution time of ST 3 is given by:

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

In Figures 5.7 and 5.8 the total execution times for each algorithm as a function of the number of active columns is presented. In Figure 5.7 the time per floating point operation is 0.5 microseconds and in Figure 5.8 it is 25.[1] We have considered only the second version of VBB, VTM, and ST algorithms. We have also included the curve for the total execution time of the relational organization. The main storage size is 100 pages in both figures. In Figure

---

[1] these two values were chosen in order to analyze I/O bound and CPU bound executions for all the algorithms.

5.7, the VBB 2 algorithm performs better than the relational algorithm for $p \leq 10$. VTM 2 outperforms the relational algorithm for $p \leq 13$ and ST 2 for $p \leq 51$. In Figure 5.8, which corresponds to a CPU bound case, VBB 2 and VTM 2 perform better than the relational organization for $p \leq 9$. ST 2 performs better for all p. However, the difference between the algorithms for transposed and relational organizations is significant only for p approximately less or equal to 10. Figure 5.9 and Figure 5.10 are also total execution time curves. But here the number of active columns is held fixed ($p = 20$) and the total execution time is plotted against the main storage size. In Figure 5.9 Tflop is 0.5 microseconds and in Figure 5.10 it is 25 microseconds. As was expected, the curves in Figure 5.9 look similar to the curves in Figure 5.5 (where only I/O was shown). The performance of VBB 2 in Figure 5.10 is noteworthy 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" 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. For a small memory size the time in the overlap region is I/O. However, since the CPU is slow, for M greater than approximately 25, the overlap region becomes CPU bound. Therefore the total time for an inner product becomes the startup I/O time plus the inner product computation 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 5.11 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 again 100 pages and the number of active columns is 20. The curves clearly possess an I/O bound region and a CPU bound region. The critical value for the stripes algorithms is approximately 1.5 microseconds per flop and for the other algorithms about 10 microseconds.

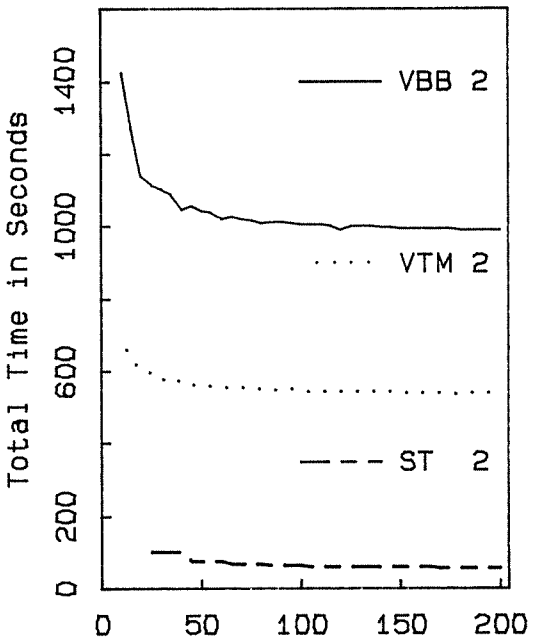


Fig. 5.9

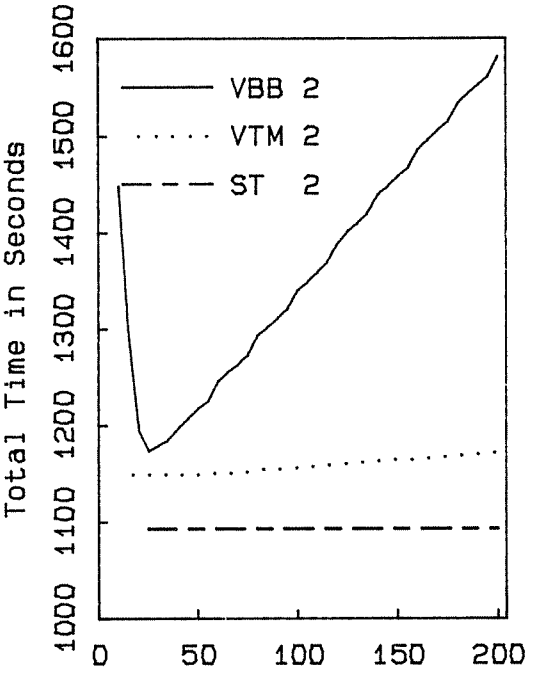


Fig. 5.10

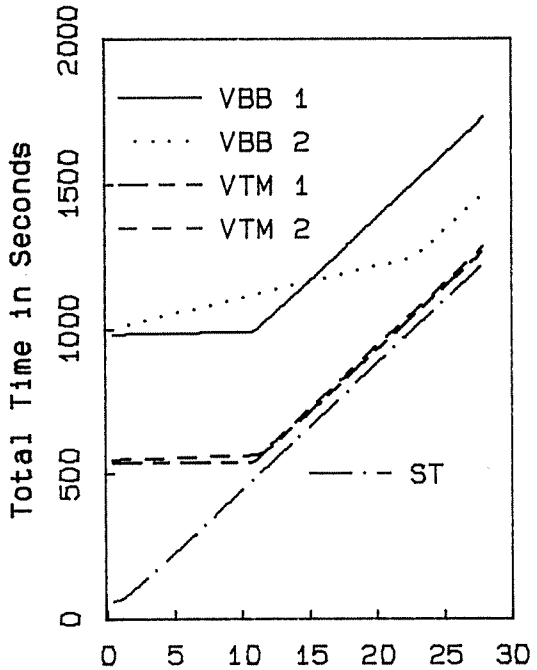


Fig. 5.11

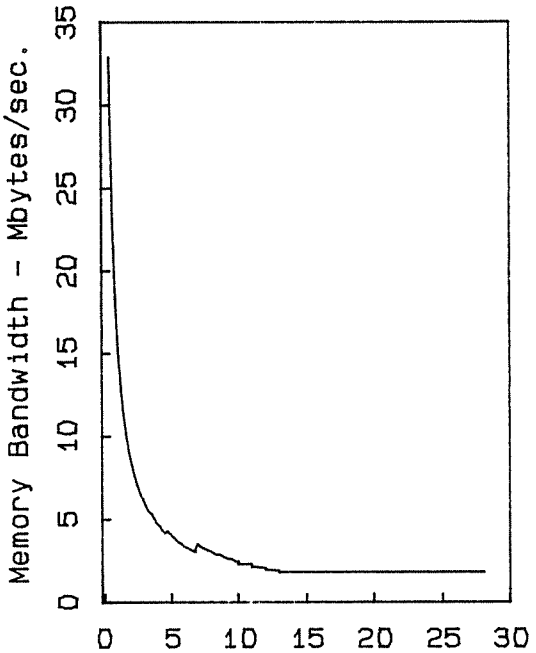


Fig. 5.12

The curves show that the stripes algorithms become CPU bound much faster than the others.

Finally, to provide an idea about the memory bandwidths required to support high CPU speeds, Figure 5.12 illustrates the memory bandwidth 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. Since the curves for the different algorithms looked almost identical, this curve is that of ST 2 (with $p = 20$ and $M = 100$).

In an efficient implementation of $X'X$ all is needed is one pass over the active columns. Moreover, the operation $X'X$ is achieved by forming all the inner products $X_i \cdot X_j$, where X_i and X_j are the i^{th} and j^{th} columns of X. Therefore, assuming it was possible to access the corresponding pages of the active columns, we considered the problem of interconnecting $\frac{p \cdot (p-1)}{2}$ processors, so that all possible pairs of the p columns of X are processed. In the next section we present a multistage shuffle/exchange network which allows an efficient parallel evaluation of $X'X$.

5.3. A Multiprocessor Interconnection for $X'X$

5.3.1. Introduction

In this Section we consider a type of interconnection network for an array of processors constructed so that the processors act on all possible pairs of p distinct input streams, $X_0, X_1, \cdots, X_{p-1}$. This type of network, the ZETA network, is a $p-1$ stage shuffle/exchange [STON71, CHUA81] where the pattern of exchanges is identical in each column. The processors at one stage process $\frac{p}{2}$ pairs and "pipe" their data to the processors at the next stage. Therefore, if there are several consecutive sets of p element input streams to be processed, the ZETA network acts as a pipeline, with consecutive stages processing pairs of

consecutive input sets. All the processors will be executing the same operation but on different pairs of data (that is an SIMD [FLYN66] configuration).

The ZETA network was motivated by the need to evaluate the non-diagonal elements of the matrix $X'X$ where X is an $n \times p$ matrix of numerical values from a large statistical database. The main problem here is to determine the repetitive exchange pattern such that each pair (X_i, X_j) is formed in some switching element at some stage.

Multistage shuffle/exchange networks have been extensively studied and analyzed [LAW75, PARK80, FREU81, LENF78, NASS81, LANG76]. However, in these studies the main emphasis has been the realizable permutations of an input stream {0, 1, 2, ..., p - 1}. With ZETA networks the problem is not the determination of particular permutations, but the possibility of forming all the pairs X_i, X_j for $i \neq j$. With p inputs there are exactly $G = \frac{p \cdot (p - 1)}{2}$ such pairs. If there are G processing elements available, and if the input pairs are formed in (p - 1) stages (p/2 pairs per stage) the problem is determining an interconnection which will guarantee that each possible pair is formed in some switching element at some stage.

We first consider the special case where $p = 2^m$. The network can be pictured as having $\frac{p \cdot (p - 1)}{2}$ processors arranged as $p - 1$ columns with $\frac{p}{2}$ processors in each column. Each processor has two inputs and two outputs and the processors in one column are connected to the processors in the next column through a perfect shuffle. The exchange part of the shuffle/exchange comes from each processor being able to switch its inputs or send them straight through to its outputs. In a ZETA network, whether or not a processor switches its inputs is determined only by the row in which the processor is. That is, all the processors in a row switch their inputs, or all the processors in a row send their inputs straight through. This means that

when $\frac{p \cdot (p-1)}{2}$ processors are not available, the network could be emulated by $\frac{p}{2}$ processors where successive stages are performed at successive times.

In Section 5.3.2 we define and prove some of the important properties of ZETA networks. We show that it is always possible to determine ZETA networks with the property that each possible pair of a stream of inputs is formed. The formation of the patterns has an elegant algorithm which is related to error-detection codes and pseudo-random number generation. The general case of p not necessarily a power of 2 is considered in Section 5.3.3.

For the binary arithmetic operations, "." is binary multiplication (AND) and "+" is mod 2 addition (exclusive OR).

5.3.2. ZETA Networks

In this section we define ZETA networks and show that for a special class of ZETA networks we will be able to generate all the (X_i, X_j) pairs. As indicated earlier we assume that $p = 2^m$. The more general case will be discussed in Section 3.

Definition 5.1: A **ZETA** network is a p - 1 stage shuffle/exchange network where, at each stage, the pattern of exchanges is the same.

Since the pattern of exchanges is the same at each stage, it is possible to emulate a ZETA network with just one stage of shuffle/exchange. At each stage of a ZETA network there are $\frac{p}{2}$ switching elements. We label these switching elements 0, 1, ..., $\frac{p}{2} - 1$. Each switching element has two outputs. At each stage, we label these outputs 0, 1, ..., p - 1. Switching element P has outputs labeled 2P and 2P+1. An output labeled j issues from a switching element

labeled $\left\lfloor \frac{j}{2} \right\rfloor$. Moreover, an output labeled j is some element X_i of the set X_0, X_1, ..., X_{p-1}. We shall call i the index of the output labeled j. At each stage of a ZETA network the outputs from the switching elements will form some permutation of this set or, equivalently, of 0, 1, ..., $2^m - 1$.

Definition 5.2: A ZETA network is 0-preserving if the first switching element is not set (that is, it does not switch its inputs). This results in 0 being the index of the output labeled 0 at each stage.

Definition 5.3: A ZETA network is i-complete if i (that is X_i) is paired with each j (that is X_j) in some switching element.

Since our goal is to form all the (i, j) pairs, we are interested in ZETA networks which are i-complete for all i.

Definition 5.4: A ZETA network is complete if it is i-complete for all i= 0, 1, ..., p - 1.

Example: In Figure 5.13 we have a 7 stage ZETA network. Switching elements 2 and 3 exchange their inputs. The ZETA network is 0-preserving and complete. At each stage we have labeled the switching elements, the outputs and indicated the binary representation of the indices. Note that, at each stage, the output indices constitute a permutation of 0, 1, ..., 7. For example, at stage 3 the outputs constitute the permutation X_0, X_7, X_6, X_1, X_3, X_4, X_5, X_2 of X_0, X_1, ..., X_7. This ZETA network has an interesting property. At the bottom of each column of the switching elements we have given the binary representation of the

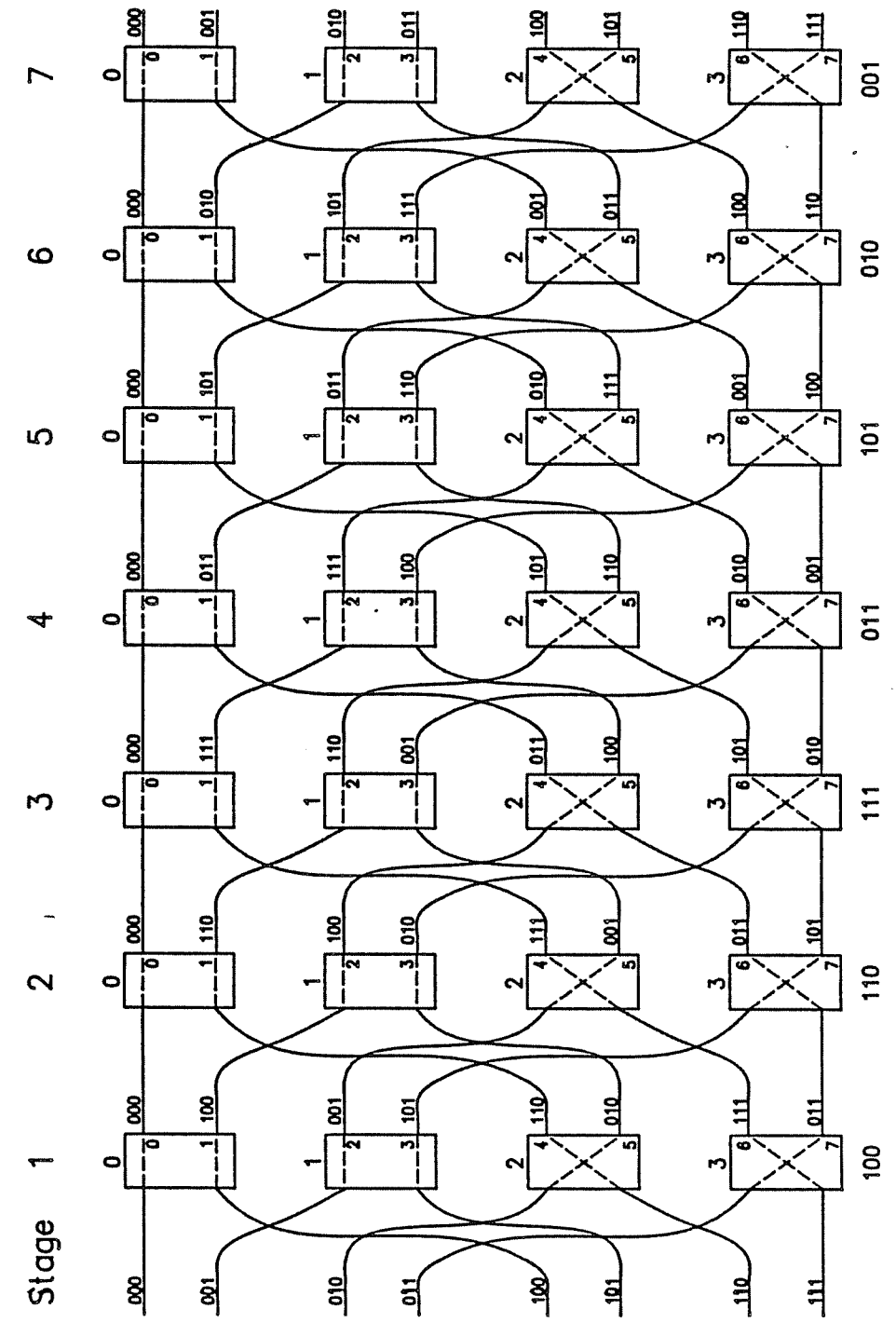


Figure 5.13

index of of the element which is paired with 0. For example at stage 3, 0 is paired with 7 or 111. Now observe that if we take the exclusive OR of the binary representations of any other pair of indices at this stage we also get 111. This property is true for any stage of this ZETA network. In other words, at each stage of this ZETA network the binary representations of the indices of the outputs from the switching elements differ in the same bit positions.

Definition 5.5: A ZETA network is bit difference preserving if at each stage the binary representations of the indices of the outputs from the switching elements differ in the same bit positions.

If a ZETA network is bit difference preserving then for any pair (i, j) the binary representations of i and j will differ according to $d_{i,j} = i + j$ and all the $d_{i,j}$'s at the same stage will be identical. But if the network is also 0-complete, any m - bit number will be paired with 0 at some stage. Hence any (i, j) will be the output of some switching element at some stage. Therefore:

Theorem 5.1: A bit difference preserving ZETA network is complete iff it is 0-complete.

Again consider the example in Figure 5.13. Note that the switching elements 0, 1, 2, 3 (or 00, 01, 10, 11) interchange their inputs iff $1 \cdot P_0 + 0 \cdot P_1$ is 1 (P_0 P_1 is the binary representation of the label). Therefore 10 (in binary) "determines" the exchange pattern. Moreover, in any row of the network, consider the binary representations of the output indices from the switching elements. It can be shown that the least significant two bits of a binary representation

in a stage are obtained from the binary representation of the previous stage through a right shift. For example, at stage 3, 111 is paired with 0 and at stage 4, 011 is paired with 0. The least significant two bits of 011 are obtained from 111 through a right shift. The most significant bit of 011, however, is the complement of the least significant bit of 111. In fact it can be shown that for this ZETA network, the most significant bit of each output index is the exclusive OR of the least significant bit of the previous output index with the "inner product" of 10 (in binary) with the remaining bits of the previous output index. In our example the most significant bit 0 of 011 is equal to $1 + 1 \cdot 1 + 1 \cdot 0$.

Definition 5.6: Let $t = t_0 t_1 \cdots t_{m-2}$ be any fixed (m - 1) bit number. Then if the switching elements 0, ..., $2^{m-1} - 1$ switch their inputs when

$$t_0 \cdot P_0 + t_1 \cdot P_1 + \cdots + t_{m-2} \cdot P_{m-2} = 1 \qquad (5.1)$$

where $P = P_0 P_1 \cdots P_{m-2}$ is the binary representation of switching element P, the ZETA network is said to be t-determined.

In what follows we shall show that for t-determined ZETA networks the binary representations of the permutation at stage k are determined from the binary representations of the permutation at stage k-1 through a right shift and the complementation of the most significant bit depending upon t as well as the remaining bits (see (5.2)). We shall also show that t-determined ZETA networks are bit difference preserving.

Theorem 5.2: Let $t = t_0 t_1 \cdots t_{m-2}$ and consider the t-determined ZETA network. If $S(j, k) = S(j, k)_0 S(j, k)_1 \cdots S(j, k)_{m-1}$ is the binary representation of the index of the output labeled j at stage k, then:

$$S(j,k)_i = S(j,k-1)_{i-1} \quad \text{for } i = 1, \cdots, m-1$$

and (5.2)

$$S(j,k)_0 = S(j,k-1)_{m-1} + S(j,k-1)_0 \cdot t_0 + \cdots + S(j,k-1)_{m-2} \cdot t_{m-2}$$

Proof: We shall prove the theorem by induction. Clearly in the first stage of the shuffle exchange network we have:

$$S(j,1)_i = S(j,0)_{i-1} \quad \text{for } i = 1,\ldots,m-1 \tag{5.3}$$

and in fact (5.3) holds for any exchange pattern. We need to show the second equality of (5.2) holds for k=1. First note that $S(j,0)_0\, S(j,0)_1 \cdots S(j,0)_{m-2}$ is switching element $\left\lfloor \frac{j}{2} \right\rfloor$ and, since the network is t-determined, it is switched iff (5.1) holds. But switching corresponds to complementing $S(j,0)_{m-1}$. Therefore (5.2) holds for k=1.

Assume (5.2) holds for $k \leq (n-1)$. Now there exists an r such that $S(j,n)$ and $S(j,n-1)$ are equal to $S(r,n-1)$ and $S(r,n-1)$ respectively : r is the index which, when shuffled and exchanged, gets mapped into j. But then, by the induction hypothesis, $S(r,n-1)$ is obtained from $S(r,n-2)$ through (5.2).

Q.E.D.

Theorem 5.3: The t-determined ZETA networks are bit difference preserving.

Proof: Again we shall prove our contention by induction. Now for k=1 (that is at the first stage), no matter what the exchange pattern is, the binary representations of the output indices from the switching elements differ only in the most significant bit position. Now assume for $k \leq n-1$ the output indices from the switching elements have the bit difference given by $S(k)_1$. That is assume for the two outputs $S(j_1, k)$ and $S(j_2, k)$ from any switching ele-

ment j

$$S(j_1, k)_i + S(j_2, k)_i = S(1, k)_i \quad \text{for } i = 0, \cdots, m-1 \qquad (5.4)$$

holds for k = 1, ..., n - 1. That (5.4) holds for k= n follows from the fact that $S(j_1, n)$, $S(j_2, n)$ and $S(1, n)$ are obtained from $S(j_1, n-1)$, $S(j_2, n-1)$ and $S(1, n-1)_1$ through (5.2).

Q.E.D.

Since t-determined ZETA networks are obviously 0-preserving and, as shown above, also bit difference preserving, to show that a t-determined ZETA network is complete we need to show it is 0-complete (Theorem 5.1). It is easy to see that not all t-determined ZETA networks are 0-complete. For example if t= 00...0 we get the perfect shuffle (without any exchanges) at each stage and 0 will be paired, repeatedly, with 2^k for k = (m - 1), (m - 2) , ... 0. To find t-determined ZETA networks which are 0-complete we need to determine the t's which, starting with 2^{m-1}, "generate" through (5.2) all the $2^m - 1$ non-zero m - bit numbers. This problem has been solved algebraically and $2^m - 1$ corresponds to the maximum period possible for a linear feedback shift register of m stages [GOLO67]. It has applications in generating m - bit pseudo random numbers with maximum periodicity [KNUT81].

In what follows we assume our reader has some familiarity with Galois fields GF(2^m) and the algebra of polynomials over a field (see [BIRK70, PETE72]). Below we give some preliminary definitions and two theorems which will guarantee the existence of 0-complete ZETA networks for any m. Since these results are well known in algebra and in the literature of error generating codes we state them without proof.

Definition 5.7: The order of a nonzero element α of a multiplicative group G is the smallest

positive integer r such that $\alpha^r = \alpha$. Note that $\alpha^{k+r} = \alpha^k$ for all k and the sequence $\{ \alpha^k \}_{k=0}^{\infty}$ is periodic of period r, with the first r elements all distinct.

Definition 5.8: An element α of GF(2^m) is called primitive if its order is 2^m-1. Every non-zero element of GF(2^m) can be expressed as a power of α and the multiplicative group of the non-zero element of GF(2^m) is cyclic.

Definition 5.9: An irreducible polynomial of degree m over GF(2) is called a primitive polynomial if it has a primitive element of GF(2^m) as a root.

Theorem 5.4 [KNUT81, BIRK70]: Over GF(2) there are $\frac{\phi (2^m - 1)}{m}$ primitive polynomials of degree m, where ϕ is Euler's ϕ function.

Theorem 5.5 [BIRK70, PETE72]: Let

$$T(X) = 1 + t_0 \cdot X + t_1 \cdot X^2 + \cdots + t_{m-1} \cdot X^m \tag{2.5}$$

be a primitive polynomial of degree m over GF(2) (note that $t_{m-1} = 1$). Then starting with any initial non-zero m-bit number $S(0) = S(0)_0 S(0)_1 \cdots S(0)_{m-1}$ the sequence $\{ S(i) \}_{i=0}^{\infty}$ generated by

$$S(k)_i = S(k-1)_{i-1} \quad \text{for } i = 1, \cdots, m-1$$

and

$$S(k)_0 = S(k-1)_{m-1} + \sum_{i=0}^{m-2} S(k-1)_i \cdot t_i \tag{2.6}$$

is periodic of period $2^m - 1$.

Theorem 5.6: For any m we can always find a ZETA network of $2^m - 1$ stages which is complete.

Proof: For any m we can always find a primitive polynomial of degree m over GF(2) (Theorem 5.4). Let T(x) given by (2.5) be such a primitive polynomial. Consider the t-determined ZETA network, where $t = t_0\, t_1 \ldots t_{m-2}$. By Theorem 5.3, any t-determined ZETA network is bit difference preserving. Moreover, Theorem 5.5 implies this t-determined ZETA network is also 0-complete. We are done by Theorem 5.1.

Q.E.D.

5.3.3. The General Case

In the previous section we showed that we can always find a ZETA network which will generate all the pairs (X_i, X_j) of $X_0, X_1, \ldots, X_{p-1}$, where $p = 2^m$. There are two basic problems in constructing a ZETA network for an arbitrary p: (1) it is not realistic to assume that the number of inputs will always be a power of two; (2) for large p it might be unreasonable to construct a ZETA network of p - 1 stages (which requires $\frac{p \cdot (p-1)}{2}$ processors). In this section we propose a number of solutions for these two problems.

(a) Introducing null elements: perhaps the easiest way to handle the first case is through the introduction of null elements to the original input. That is find the smallest m such that $p \leq 2^m$ and append $2^m - p$ null elements to the original input. Note that with this scheme there are $\frac{2^m \cdot (2^m - 1)}{2} - \frac{p \cdot (p-1)}{2}$ null operations. If p is $2^k + s$ for a relatively small s (for example s= 1) this could degrade the throughput considerably.

(b) Emulation (for large p): if p is slightly less than a power of 2 but still large, it might be unreasonable to construct the ZETA network. We mentioned earlier that the repetition of the exchange patterns allows us to emulate the ZETA network through just one stage of

shuffle/exchange (by piping the data back to the switching elements a total of p - 2 times).

If $\frac{p}{2}$ is still large, it is possible to use quotient networks [FISH82] and emulate a shuffle/exchange of 2^{m+q-1} switching elements with a shuffle/exchange of 2^{m-1} switching elements. Each switching element will have 2^{q+1} inputs/outputs and must process 2^q pairs at each stage (for a total of 2^{m+q-1} stages).

(c) Partitioning: another way to handle the general case is by partitioning. Suppose $p = q \cdot 2^k$. There are two methods for partitioning:

(1) We can partition the p inputs into q subsets of 2^k elements each. Then a $2^k - 1$ stage ZETA network can be used to process the input pairs within each subset. The pairs across the subsets might be processed serially, through broadcasting etc.. The latter gives maximum efficiency if the ZETA network is emulated through a one stage shuffle/exchange. In this case if the switching elements contain blocks from one subset, the corresponding blocks from another subset can be broadcast to all the switching elements, one by one. Each switching element will process two pairs across subsets and all the switching elements will be operating in parallel.

(2) We can partition the p inputs into 2^k subsets of q elements each. With this scheme, the ZETA network is used to process the pairs across the subsets and each switching element will process q^2 pairs at each stage. As with the previous scheme, there are several choices for processing the pairs within each subset. Unlike (1), however, the ZETA network is accessing all the input. Therefore, it is conceivable to uniformly distribute the amount of computation within each subset (that is processing $\frac{q \cdot (q-1)}{2}$ pairs), across the $2^k - 1$ stages of the network.

5.3.4. Summary

In summary, we have shown that we can determine a pattern of exchanges for a ZETA network, such that all the (X_i, X_j) pairs of a given set of inputs $X_0, X_1, \ldots, X_{p-1}$ are processed. We defined ZETA networks with this property "complete". We saw that primitive polynomials can determine complete ZETA networks, and since there are $\frac{\phi(2^m - 1)}{m}$ primitive polynomials (over GF(2)) for any m, it is always possible to construct a complete ZETA network for any m.

ZETA networks can be used efficiently to evaluate the nondiagonal elements of X'X. In the types of applications we are currently investigating, X is large. In particular, X is much larger than the primary memory of the underlying system. Therefore, with a $p - 1$ stage ZETA network, it is possible to process the matrix X in horizontal stripes. As we mentioned in Section 5.3.1, the ZETA network acts as a pipeline, with consecutive stages processing consecutive stripes of X. Therefore, if X is divided into N horizontal stripes and the time to access a horizontal stripe from secondary store is synchronized with the processing time needed at a stage of the ZETA network, X'X can be evaluated in $p + N$ time steps.[2]

Finally, the repetitive exchange pattern allows the ZETA network to be emulated through just one stage of shuffle/exchange. This requires only $\frac{p}{2}$ processors. This emulation is one way to handle a large p. Emulation through quotient networks, introduction of null values and partitioning are some of the other methods that can be used to efficiently handle an arbitrary p (either very large or not equal to a power of two).

---

[2] where a "time step" is the processor time needed at a stage of the network.

5.4. Conclusions

This Chapter presented three types of algorithms for the evaluation of X' X. These were labeled VBB, VTM, and 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.

We have shown that the number of page transfers grows quadratically in p (the number of active columns) for the VBB and VTM algorithms, and linearly in p for the ST algorithms. Therefore, in terms of the number of page transfers the ST algorithms are always optimal. On the other hand, the number of random seeks for the ST algorithms is significant.[3] The effect of the seek times is clearly demonstrated in the curves of the total I/O times as functions of p. The curves of the total I/O times for the VBB and VTM algorithms are quadratic in p, but the substantial percentage of seeks for the ST algorithms show as jumps in the linear curves.

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[4] then to be within f percent of the asymptote the memory size needs to be:

$$M = \frac{(1-f) \cdot k \cdot Tdac}{f \cdot Tio}$$

In comparing the total execution times,[5] we observed that with a fast CPU VBB 2 and VTM 2 perform better than the relational organization for p less or equal to 10 and 13 respectively. The ST 2 algorithm with the transposed organization performs better than the relational

---

[3] The VTM 1 strategy involves the least number of random seeks, although the number of page transfers is greater than the ST algorithms.

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

[5] for the I/O bound case (i.e. Tflop = 0.5 microseconds)

organization, provided the number of active columns is less than or equal to 51. Although the stripes algorithm always involves fewer page transfers,[6] the performance degradation for the larger values of p is again due to the cost of seek operations.

Considering the total execution times as functions of Tflop, we observed that the stripes algorithms become CPU bound for smaller values of Tflop. The VBB and VTM become CPU bound for Tflop $\geq$ 10 microseconds. On the other hand the ST algorithms become CPU bound for Tflop $\geq$ 1.2 microseconds. This provides another illustration of the extra I/O overhead of the VBB and VTM algorithms.

The good performance of the ST algorithms and a close investigation of the X' X operation suggested that in an optimal implementation only one pass over the active columns is needed. Therefore we proposed a multistage shuffle/exchange network (ZETA networks), with the property that in one pass through the network all the inner products of X (i.e. X' X) are accumulated. ZETA networks provide an efficient, pipelined evaluation of X' X, as well as an example of a specialized multiprocessor architecture for a particular computational operation.

---

[6] the number of page transfers for ST algorithms is p N which is always less than R

CHAPTER 6

The QR Decomposition

6.1. Introduction

For the QR decomposition of X, we apply p Householder transformations on X to reduce it to upper triangular form. Each Householder transformation is defined through a vector "u" and a scalar "c". In fact if H is a Householder transformation then

$$H = I - \frac{u \cdot u'}{c}$$

where I is the n x n identity matrix, u is an n dimensional vector and c is a constant; c is chosen such that $H' = H$ and $H' \cdot H = H \cdot H' = I$. The k^{th} Householder transformation zeroes the subdiagonal elements of the k^{th} column. The u of the k^{th} Householder transformation is:

$$u = e_k + \frac{x_k}{s \cdot ||x_k||}$$

where, e_k is the k^{th} n-dimensional unit vector, x_k is the k^{th} column of the transformed X and s is the sign of X (k,k). The c of the k^{th} Householder transformation is:

$$c = 1 + \frac{X(k,k)}{s \cdot ||X_k||}$$

After this transformation is applied, $X(k,k) = -s \cdot ||x_k||$ and $X(j,k) = 0$ for $j = (k+1), (k+2), \ldots, n$. The other columns are transformed according to:

(a) $t = \frac{u' \cdot X_j}{u(1)}$ and

(b) $X_j = X_j - t \cdot u$

Let H1, H2, ..., Hp be the p Householder transformations applied (on the left) to X. Then

$$Hp \cdots H2 \cdot H1 \cdot X = R$$

where R is an n x p upper triangular matrix. Let $Q = H1 \cdot H2 \cdots Hp$. Note that $Q' \cdot Q = Q \cdot Q' = I$ (that is Q is an orthogonal matrix). Therefore:

$$Q' \cdot X = R \text{ or } X = Q \cdot R$$

Since $n \gg p$, the last (n - p) rows of R are zeros. So for R, the storage requirement is of order $O(p^2)$. Q (or Q') can be recovered through the u's and c's of the Householder transformations. Therefore, rather than storing Q (which is of order $O(n^2)$) explicitly, we can store the u's in the zeroed lower triangular part of X. That is after the transformations are applied , the upper triangular part of X will contain the upper triangular part of R and the lower triangular part of X will contain the u's. The upper triangular part of R can also be kept in main store. Whenever Q is needed, the Householder transformations can be applied to the nxn identity matrix. In most applications only the first p columns of Q are needed. If that is the case the Housecholder transformations are applied to the first p columns of the nxn identity matrix. This is one scheme for accumulating Q or a submatrix of Q. In Chapter 7 we shall introduce a more efficient way for accumulating the transformations and constructing Q (or the first p columns of Q). As we shall see, our method will imply a substantial reduction in the I/O overhead.

Throughout, the column which determines the transformation is called the "pivot," and the remaining active columns are the non-pivotal columns. Our performance evlaution assumes a simple "pivoting scheme," where the pivot at the i^{th} iteration is the i^{th} column. A stabler pivoting scheme is to choose the pivot as the column with the largest norm, among the currently

active columns.

Note that we need the 2-norm of the pivot in order to define the transformation. A substantial savings in I/O and computation will occur if the 2-norms of the non-pivotal columns are updated (vs. recalculated). The formula for the update is:

$$||X_j^{k+1}|| = \left[||X_j^{k}||^2 - |r_{kj}|^2\right]^{1/2} \tag{6.1}$$

where $||X_j^{k}||$ is the 2-norm of the j^{th} column before the k^{th} iteration, $||X_j^{k+1}||$ is the 2-norm of the same column after the j^{th} transformation, and r_{kj} is the element in the k^{th} row and j^{th} column of R. However, as pointed in LINPACK [DONG79], if we are not careful, rounding errors and over (or under) flows can occur with this updating expression. LINPACK allows for recomputation whenever:

$$\lambda\, ||X_j^{k}|| \le \epsilon ||X_j^{1}|| \tag{6.2}$$

or

$$\lambda \frac{||X_j^{k}||}{||X_j^{1}||}\left[1 - \left(\frac{|r_{k,j}|}{||X_j^{k}||}\right)^2\right]^{1/2} \le \epsilon^{1/2} \tag{6.3}$$

where λ is a small constant, ϵ is the the rounding unit for the underlying processing subsystem, $||X_j^{1}||$ is the initial 2_norm of the j^{th} column of X, and $r_{k,j}$ is the element in the k^{th} row j^{th} column of X, updated in the k^{th} iteration. It is given by:

$$r_{kj} = X_{kj} - u'_k \cdot X_j \tag{6.4}$$

Therefore if the inner products of the k^{th} pivot with the remaining columns and the k^{th} row of the current X are available, by (6.4) and (6.1), it is possible to determine the 2-norms before actually applying the transformation.

In some cases the user will want to specify initial, final, and free columns. He will then force the initial and final columns to be "frozen" and the free columns to be permuted according to their largest norms during the execution of the decomposition. Since the choice of the columns is completely user dependent, we shall not consider this general case. Instead we shall study the performance of the simpler situation where in the i^{th} iteration, the i^{th} column is the pivot. We shall assume that the 2-norms are always updated. The decision for recomputing the 2-norms (according to (6.3)) is also data dependent. It is relatively easy to incorporate the fixed overhead of the worst case, where at each iteration all the 2-norms are recomputed. This will provide an upper bound for the actual overhead of recomputing some of the 2-norms when (6.3) is satisfied.

The decision to pivot on the i^{th} column in the i^{th} iteration, could easily be relaxed to the more general and more stable algorithm of pivoting on the column with the largest norm. This simply entails the permutation of some of the columns of X. The performance evaluation is essentially the same.

To minimize the I/O overhead, we have introduced a look-ahead scheme which eliminates the I/O references for accumulating the inner products. Our technique is simply this: retain in primary storage corresponding pages of the current pivot and the next pivot, then accumulate the dot products for the next transformation while applying the current transformation. In the usual implementation of the QR decomposition (with columns stored in a transposed organization), the current pivot is accessed in blocks and corresponding blocks of the remaining columns are read, processed, and written back. For the following iteration, a pass is made on the next pivot and the remaining columns to accumulate the inner products. Once the inner products are evaluated, the application of the transformation proceeds as before. With our scheme, we retain not only the current block of the pivot, but also the current block of the next pivot. Then pro-

cessing the corresponding blocks of the remaining columns involves two steps. First apply the current transformation and, second, accumulate the inner product of the transformed block with the transformed block of the next pivot. To make this scheme work, we need to know the index of the next pivot before applying the current transformation. If the the i^{th} pivot is the i^{th} column, this is trivial. However, if the algorithm is designed to pick the next pivot as the column with the largest norm, we must first determine the column which will have the largest norm after the current transformation is applied. This must be accomplished before actually applying the transformation. Therefore our scheme will not work if (6.3) is satisfied for any column.

6.2. Algorithms

Next we present three high level algorithms for the QR decomposition. These high level algorithms are used primarily to determine buffer management strategies. They are representative of the different orderings of the indices and operations.

The first algorithm, VBB, uses two vector building blocks. The second algorithm, VM, improves the first by using a vector-matrix approach for the inner product and AXPY operations. Finally, in LA (which is a direct implementation), the inner products of the next transformation are accumulated while applying the current transformation. Although in the performance evaluation we will always assume the 2-norms are updated, the algorithms given below test for recomputation (through the boolean function "Recompute").

VBB:

For i := 1 to p - 1 /* evaluate the "s" and the "c" of the next transformation */

$s_i := \text{sign}(X_{i,i})$

$c_i := 1 + \frac{X_{i,i}}{s \cdot ||X_i||}$

$r_{i,i} := -s \cdot ||X_i||$

For j := i+1 to p

For k := i to n /* accumulate the inner products */

$t_j := t_j + X_{k,j} \cdot X_{k,i}$

$t_j := \frac{t_j}{c}$

$X_{i,j} := X_{i,j} - t_j \cdot X_{i,i}$

$r_{i,j} := X_{i,j}$

For k := i to n /* apply the transformation */

$X_{k,j} := X_{k,j} - t_j \cdot X_{k,i}$

If Recompute (j) /* recompute the 2-norm */

$||X_j|| := 0$

For k := i+1 to n

$||X_j|| := ||X_j|| + X_{k,j} \cdot X_{k,j}$

$||X_j|| := ||X_j||^{1/2}$

else /* update the 2-norm */

$$||X_j|| := ||X_j|| \cdot \left[1 - \left(\frac{r_{i,j}}{||X_j||}\right)^2\right]^{1/2}$$

VM:

For i := 1 to p - 1

$s_i := \text{sign}(X_{i,i})$

$c_i := 1 + \frac{X_{i,i}}{s \cdot ||X_i||}$

$r_{i,i} := -s \cdot ||X_i||$

For j := i + 1 to p

$check_j := false$

$t_j := 0$

/* accumulate the inner products of pivot i with the remaining columns */

For k := i to n

For j := i + 1 to p

$t_j := t_j + X_{k,j} \cdot X_{k,i}$

For j := i + 1 to p

$t_j := \frac{t_j}{c}$

$X_{i,j} := X_{i,j} - t_j \cdot X_{i,i}$

$r_{i,j} := X_{i,j}$

If Recompute (j)

$check_j := true$

$||X_j|| := 0$

else

$$||X_j|| := ||X_j|| \cdot \left[1 - \left(\frac{r_{i,j}}{||X_j||}\right)^2\right]^{1/2}$$

For k := i to n /* apply the current transformation */

For j := i + 1 to n

$X_{k,j} := X_{k,j} - t_j \cdot X_{k,i}$

If $check_j$

$||X_j|| := ||X_j|| + X_{k,j} \cdot X_{k,j}$

LA:

$$s := \text{sign}(X_{1,1})$$

$$c := 1 + \frac{X_{1,1}}{s \cdot ||X_1||}$$

$$r_{1,1} := -s \cdot ||X_1||$$

For $j := 2$ to p $[t_j := 0]$

/* accumulate the inner products of the first column
with the remaining columns */

For $k := 1$ to n

Fo $j := 2$ to p

If $(k = 1)$ $[r_{1,j} := X_{1,j}]$

$$t_j := t_j + X_{k,j} \cdot \frac{X_{k,1}}{s \cdot ||X_1||}$$

For $j := 2$ to p

$$r_{1,j} := r_{1,j} - t_j$$

$$t_j := \frac{t_j}{c}$$

If Recompute (j)

$\text{check}_j := \text{true}$

$||X_j|| := 0$

else

$$||X_j|| := ||X_j|| \cdot \left[1 - \left(\frac{r_{1,j}}{||X_j||}\right)^2\right]^{1/2}$$

/* apply the current transformation and accumulate the inner
products of the next pivot with the remaining columns */

For $i := 1$ to $p - 1$

For $j := i + 1$ to p $[t_{j2} := 0]$

For k := i to n /* apply the transformation */

$X_{k,i+1} := X_{k,i+1} - t_{i+1} \cdot X_{k,i}$

If check $_{i+1}$

$||X_{i+1}|| := ||X_{i+1}|| + X_{k,i+1} \cdot X_{k,i+1}$

If (k = i+1)

$s2 := \text{sign}(X_{i+1,i+1})$

$$c2 := 1 + \frac{X_{i+1,i+1}}{s2 \cdot ||X_{i+1}||}$$

For j := i + 2 to p

$X_{k,j} := X_{k,j} - t_j \cdot X_{k,i}$

If check $_j$

$||X_j|| := ||X_j|| + X_{k,j} \cdot X_{k,j}$

If (k ≥ i+1)

/* accumulate the inner products for the next transformation */

$$t_{j2} := t_{j2} + X_{k,j} \cdot \frac{X_{j,i+1}}{s2 \cdot ||X_{i+1}||}$$

If (k = i+1)

$r_{i+1,j} := X_{i+1,j}$

For j := i+2 to p [check $_j$:= false]

For j := i+2 to p

$t_j := t_{2j}$

$r_{i+1,j} := r_{i+1,j} - t_j$

$$t_j := \frac{t_j}{c}$$

If Recompute (j)

$$check_j := true$$

/* recompute the two norms of the columns for which $check_j$ is set */

$$||X_j|| := 0$$

else

$$||X_j|| := ||X_j|| \cdot \left[1 - \left(\frac{r_{i+1,j}}{||X_j||}\right)^2\right]^{1/2}$$

$s := s2 ; c := c2$

In the next sections, buffer management algorithms for each of the VBB, VM, and LA are presented. These algorithms are for the transposed secondary storage organization. We shall also briefly describe a direct algorithm (labeled REL) for the relational secondary storage organization. To simplify the analysis, the algorithms and the cost functions are evaluated only for the main body of each algorithm. That is,, we have not incorporated the fixed overheads for evaluating the constants of a Householder transformation (that is "c" and "s"), the conditional branches etc. These overheads are insignificant and constant across all algorithms.

6.2.1. Vector Building Block

As mentioned earlier, two vector building blocks are used for the QR decomposition: (1) inner product and (2) AXPY. For the inner product any of the VBB algorithms for X'X can be used.

6.2.1.1. AXPY

For the AXPY operation, we divide the memory into five equal subdivisions: two subdivisions for reading the next two corresponding blocks of X_i and X_j (that is the i^{th} and j^{th} columns

of the matrix X); two subdivisions for operating on the current two corresponding blocks and; one subdivision which holds the updated block of X_j from the previous iteration.

AXPY:

Read ($X_i^{M/5,1}$, $X_j^{M/5,1}$)

For k := 1 to N / (M / 5)

$ If k < N / (M / 5)

Read ($X_i^{M/5,k+1}$, $X_j^{M/5,k+1}$)

$ $X_j^{M/5,k} := X_j^{M/5,k} - t_j \cdot X_i^{M/5,k}$

$ If k > 1

Write ($X_j^{M/5,k-1}$)

Write ($X_j^{M/5,N/(M/5)}$)

6.2.2. Vector-Matrix Algorithm

There are two main steps in this algorithm: (1) form he inner products of the pivot with the remaining columns; (2) apply the Householder transformation. For (1) we can use any of the strategies of the "vector-times-matrix" scheme of X'X. For (2):

-Divide the M pages of primary memory into four equal subdivisions.

-Read the next M / 4 pages of the pivot

-Iteratively read the corresponding M / 4 pages of each of the remaining columns and, concurrently, write the updated M / 4 pages of the column from the previous iteration. Also concurrently transform the M / 4 pages of the resident active column. If there are only two active columns use the algorithm of AXPY given above.

VM:

Read ($X_i^{M/4.1}$; $X_{i+1}^{M/4.1}$)

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

For j := i+ 1 to p

$ If j < p Read ($X_{j+1}^{M/4.k}$)

else If k < N / (M / 4) Read ($X_i^{M/4.k+1}$)

$ $X_j^{M/4.k} := X_j^{M/4.k} - t_j \cdot X_i^{M/4.k}$

$ If j > i+ 1 Write ($X_{j-1}^{M/4.k}$)

If k < N / (M / 4)

$ Read ($X_{i+1}^{M/4.1}$)

$ Write ($X_p^{M/4.k}$)

6.2.3. Look-ahead Algorithm

Only one iteration of the LA algorithm is presented. We have not included the details of the complete LA algorithm. More specifically, the algorithm below primarily describes the order of the referenced pages in applying a transformation and accumulating the inner products for the next pivot. For this algorithm main storage is divided into five subdivisions: (a) M / 5 pages of the current pivot; (b) the corresponding M / 5 pages of the next pivot; (c) the corresponding M / 5 pages of the current column being transformed; (d) the corresponding M / 5 pages of the previous column being written back to secondary storage; (e) the corresponding M / 5 pages of the previous column being read.

LA:

```
Read ( X_i^{M/5,1} , X_{i+1}^{M/5,1} )
For k := 1 to N / (M / 5)
    For j := i+1 to p
        $ If j < p Read ( X_{j+1}^{M/5,k} )
          else If k < N / (M / 5) Read ( X_i^{M/5,k+1} )
        $ X_j^{M/5,k} := X_j^{M/5,k} - t_j . X_i^{M/5,k}
          If j > i+1
              t_{2j} := t_{2j} + X_{i+1}^{M/5,k} . X_j^{M/5,k}
        $ If j > i+1 Write ( X_{j-1}^{M/5,k} )
    If k < N / (M / 5)
        $ Read ( X_{i+2}^{M/5,1} )
        $ Write ( X_p^{M/5,k} )
```

6.2.4. A Direct Implementation For the Relational Organization

For the relational secondary storage organization we used a "look-ahead" scheme identical to the LA algorithm. We have labeled this algorithm REL. Since the QR decomposition produces temporary files, a partially transposed file of the active columns is first constructed. This is identified as the projection step. Subsequently, the QR decomposition of X is evaluated through p passes. The first pass evaluates the inner products for the first pivot. During the remaining p - 1 passes the transformations are applied. However, while the current transformation is applied, the inner products for the next transformation are accumulated. With the relational secondary storage organization, the data pages contain corresponding elements of all the

active columns. Therefore, besides determining the next pivot prior to the application of the current transformation, no special buffering scheme is needed.[1] Hence, the main storage is divided into three subdivisions: M / 3 pages for inputing the next block of the file, M / 3 pages for applying the current transformation and accumulating the inner products for the next transformation, and M / 3 pages to hold the previously transformed block while it is written back to disk.

6.3. Cost Functions

In this section we shall give the cost functions for the QR algorithms. We shall present the expressions for the I/O costs and the total execution times. In the QR decomposition, some of the 2-norms might be recomputed. Since this is completely data dependent, we have considered cost functions only for the case where the 2-norms are updated.

6.3.1. I/O Costs

Below we list the expressions for the number of random seeks, the number of page transfers and the total I/O costs for VBB, VM, and LA. Throughout this discussion we have assumed the existence of two I/O subsystems. One I/O subsystem is dedicated to the "read" operations and the other to the "write" operations.

For VBB, the total number of random seeks is:

$$T_{RS-VBB} = p \cdot (p-1) \cdot \frac{9 \cdot N}{M} + \frac{p \cdot (p-1)}{2}$$

the total number of page transfers is:

---

[1] with the LA algorithm for the transposed secondary storage organization we retained in primary store the current pages of the next pivot

$$T_{PT-VBB} = 5 \cdot N \cdot \frac{p \cdot (p-1)}{2}$$

and the total I/O is:

$$T_{IO-VBB} = 4 \cdot p \cdot (p-1) \cdot \frac{N}{M} \cdot T_{r-io}(\frac{M}{4}) +$$

$$5 \cdot p \cdot (p-1) \cdot \frac{N}{M} \cdot T_{r-io}(\frac{M}{5}) + \frac{p \cdot (p-1)}{2} \cdot T_{r-io}(N)$$

For VM the total number of random seeks is:

$$T_{RS-VM} = \frac{N}{M} \cdot \left[3 \cdot (\frac{p \cdot (p+1)}{2} - 3) + 4 \cdot (p + 2 \cdot (\frac{p \cdot (p-1)}{2} - 1)) + 10\right] + 1$$

the total number of page transfers is:

$$T_{PT-VM} = N \cdot \left[2 \cdot (\frac{p \cdot (p+1)}{2} - 1) + \frac{p \cdot (p+1)}{2}\right]$$

and the total I/O is:

$$T_{IO-VM} = \frac{3 \cdot N}{M} \cdot (\frac{p \cdot (p+1)}{2} - 3) \cdot T_{r-io}(\frac{M}{3}) + \frac{10 \cdot N}{M} \cdot T_{r-io}(\frac{M}{5})$$

$$+ \frac{4 \cdot N}{M} \cdot (p + 2 \cdot (\frac{p \cdot (p-1)}{2} - 1)) \cdot T_{r-io}(\frac{M}{4}) + T_{r-io}(N)$$

The total number of random seeks for LA is:

$$T_{RS-LA} = \frac{N}{M} \cdot \left[5 \cdot (\frac{p \cdot (p+1)}{2} + \frac{p \cdot (p-1)}{2} - 7) + 21 + 3 \cdot p\right]$$

the total number of page transfers is:

$$T_{PT-LA} = N \cdot (p^2 + p - 1)$$

and the total I/O time is:

$$T_{IO-LA} = \frac{5 \cdot N}{M} \left[\frac{p \cdot (p+1)}{2} + \frac{p \cdot (p-1)}{2} - 7 \right] \cdot T_{r-io}(M/5) +$$

$$\frac{20 \cdot N}{M} \cdot T_{r-io}(M/4) + \frac{3 \cdot p \cdot N}{M} \cdot T_{r-io}(M/3) + T_{r-io}(N)$$

In order to compare the performance of the relational and completely transposed storage organizations, we have also computed the expression for the I/O times of the relational organization. The QR decomposition modifies the matrix X. Therefore, for the relational organization, a partially transposed file of the active columns is first constructed and the QR decomposition is subsequently applied to the partially transposed file. The u's of the Householder transformations are stored in the subdiagonal part of the partially transposed file. The total I/O cost, therefore, is the sum of two terms:

(1) the cost of constructing the partially transposed file of the active columns

(2) the cost of applying the transformations to the partially transposed file.

For (2) there are p passes. During the first pass the inner products for the first pivot are computed. During the remaining p - 1 passes the transformations are applied. The total number of random seeks is $2 \cdot p + 1$. With two I/O subsystems, the number of random seeks for the relational organization is independent of the primary memory size. The total number of page transfers is $R + 2 \cdot p^2 \cdot N$. The total I/O cost for the relational organization is:

$$T_{QR-IO-REL} := T_{proj} + T_{QR}$$

where,

$$T_{proj} = T_{r-io}(R) + T_{r-io}(p \cdot N)$$

and,

$$T_{QR} = (2 \cdot p - 1) \cdot T_{r-io}(p \cdot N)$$

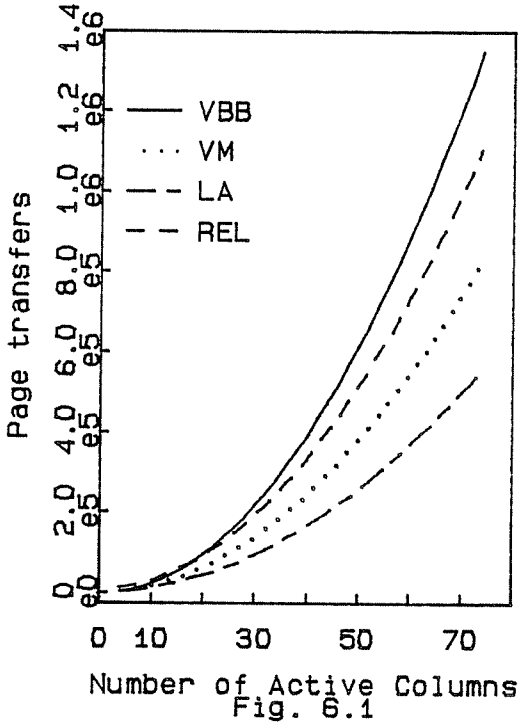


Fig. 6.1

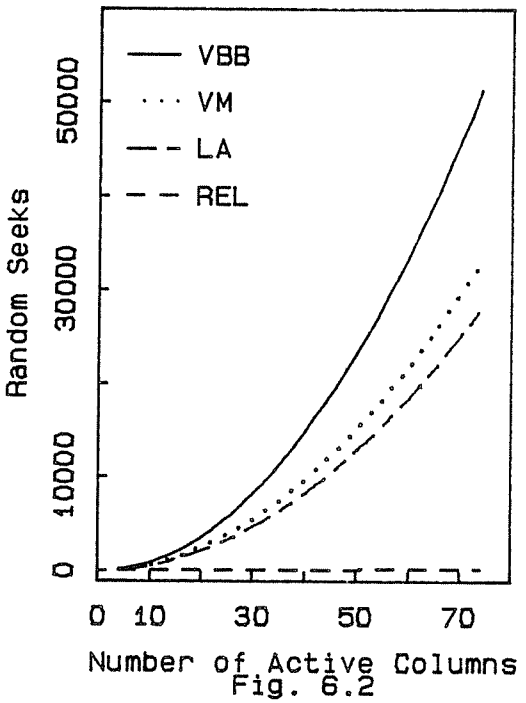


Fig. 6.2

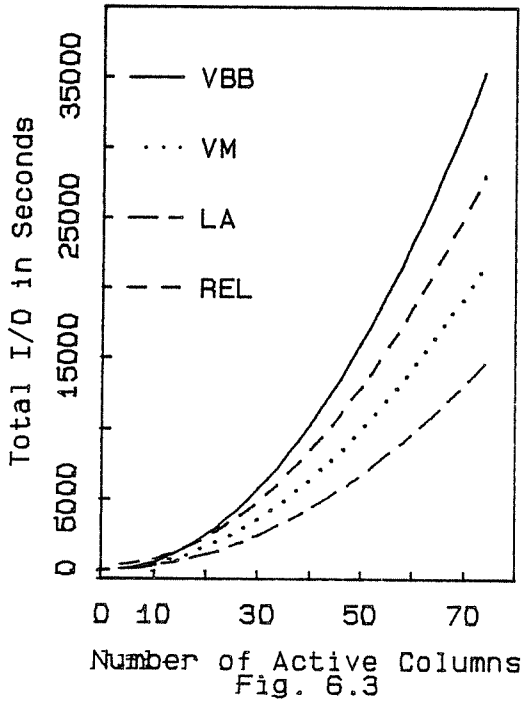


Fig. 6.3

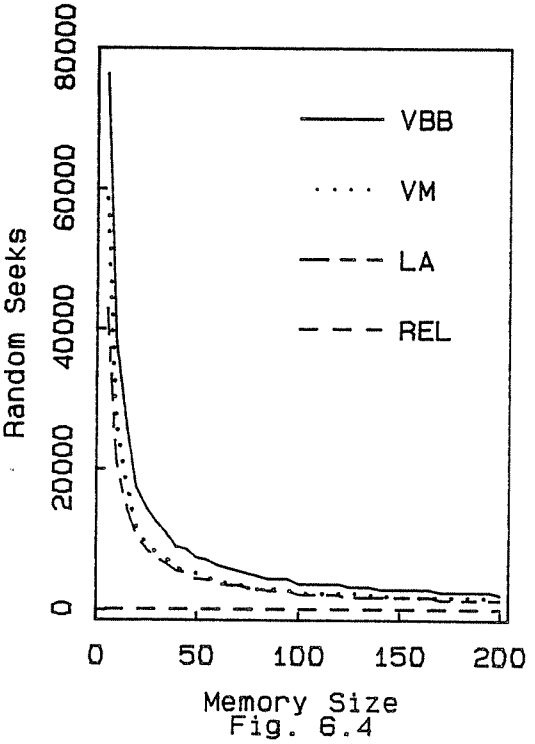


Fig. 6.4

In Figures 6.1, 6.2, and 6.3, we have plotted the number of page transfers, the number of random seeks, and the total I/O cost as a function of the number of active columns. The main storage size is held at 100 pages. The functions of the number of page transfers for VBB, VM, LA and REL are, approximately, N times $2.5 \cdot p^2$, $1.5 \cdot p^2$, p^2 and $2 \cdot p^2$. In the algorithms for the transposed organization, the number of currently active columns reduces by one at the end of each iteration. On the other hand, with the relational organization, each iteration involves $2 \cdot p \cdot N$ page transfers. Therefore, in page transfers, the VM and LA algorithms remain superior to the algorithm for the relational organization. However, with the vector building block (VBB) algorithm the pivot is referenced for each inner product and each AXPY operation. This extra overhead makes the VBB algorithm more I/O intensive than the algorithm for the relational organization when the number of active columns exceeds 16.

With the number of active columns held at 20, Figures 6.4 and 6.5 illustrate, respectively, the total number of random seeks and the total I/O time as a function of the size of the available buffer space. These curves are asymptotic of the general form

$$\frac{C_1}{M} + C_2$$

The values of these asymptotes are, approximately, the total times for the page transfers (that is the number of page transfers times Tio).

6.3.2. Total Execution Times

Next we present total execution times for the VBB, VM, and LA algorithms. The total execution time of the VBB algorithm is:

$$T_{QR-VBB} = T_{X'X-VBB2} + \frac{p \cdot p-1}{2} \cdot T_{ot-AXPY}$$

where $T_{X'X-VBB2}$ is the total execution time of the second algorithm with the vector building

block scheme of X'X, and $T_{ot-AXPY}$ is the total execution time of an AXPY operation given by:

$$T_{ot-AXPY} = 2 \cdot T_{r-io}(\frac{M}{5}) + (\frac{5 \cdot N}{M} - 1) \cdot Max(T_{AXPY}(q \cdot \frac{M}{5}), 2 \cdot T_{r-io}(\frac{M}{5}))$$

$$+ Max(T_{AXPY}(q \cdot \frac{M}{5}), \frac{M}{5} \cdot T_{io}) + \frac{M}{5} \cdot T_{io}$$

The total execution time of VM is:

$$T_{VM} = T_{X'X-VTM2} + T_{ot-AXPY} + \sum_{k=3}^{p} T_{HT}(k)$$

where $T_{X'X-VTM2}$ is the total execution time of the second algorithm with the vector times matrix scheme of X'X. $T_{HT}(k)$ is the total time to apply a Householder transformation, with k currently active columns (the pivot and k-1 remaining columns). This function is given by:

$$T_{HT}(k) = 2 \cdot T_{r-io}(\frac{M}{4}) + \frac{4 \cdot N}{M} \cdot \left[(k-1) \cdot Max(T_{AXPY}(q \cdot \frac{M}{4}), T_{r-io}(\frac{M}{4})) + T_{r-io}(\frac{M}{4})\right]$$

The total execution time of LA is:

$$T_{LA} = T_{VTM}(p) + \sum_{k=4}^{p} T_{LA}(k, 5) + T_{LA}(3, 4) + T_{ot-AXPY}$$

where, $T_{VTM}(p)$ is the time to form the inner product of the first pivot with the remaining columns, using the second algorithm of the vector times matrix scheme of X'X, and $T_{LA}(k, b)$ is the time of a look-ahead pass with k currently active columns. This function is given by:

$$T_{LA}(k, b) = \frac{b \cdot N}{M} \cdot \left[(k-2) \cdot Max(T_{AXPY}(q \cdot \frac{M}{b}) + T_{INN}(q \cdot \frac{M}{b}), T_{r-io}(\frac{M}{b}))\right]$$

$$+ 2 \cdot T_{r-io}(\frac{M}{b}) + \frac{b \cdot N}{M} \cdot \left[Max(T_{AXPY}(q \cdot \frac{M}{b}), T_{r-io}(\frac{M}{b})) + T_{r-io}(\frac{M}{b})\right]$$

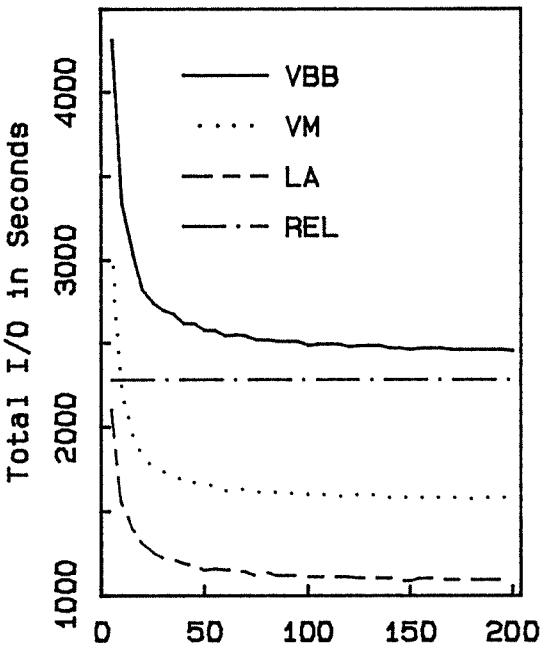


Fig. 6.5

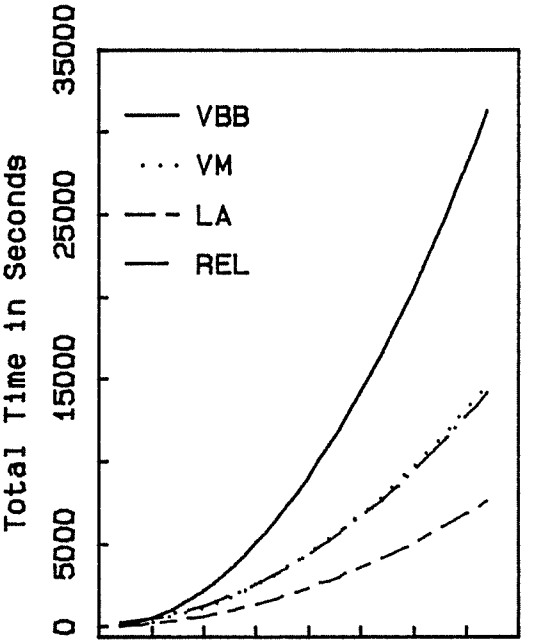


Fig. 6.6

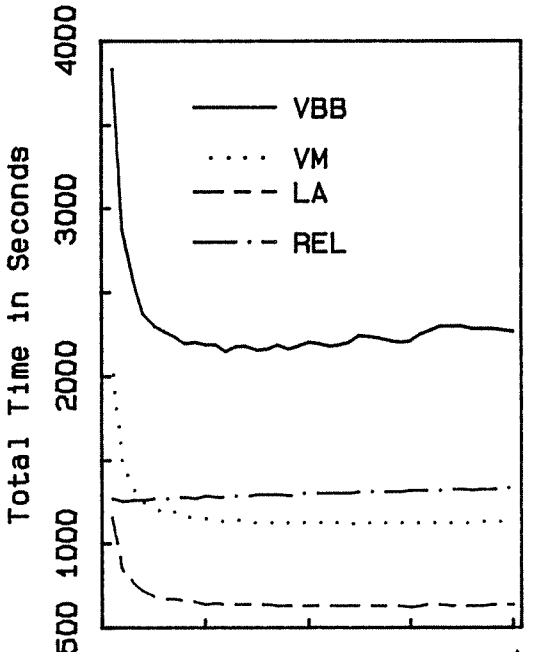


Fig. 6.7

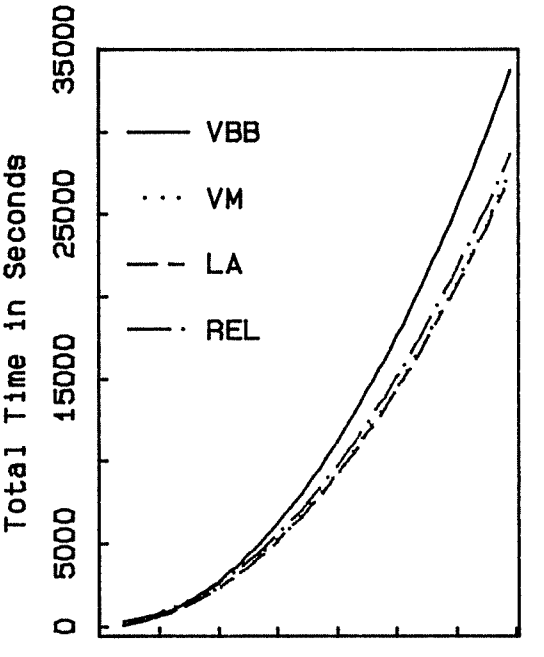


Fig. 6.8

For the relational organization, first the partially transposed file of the active columns is constructed and then p passes are made to apply the transformation, as well as accumulate the inner products for the next transformation. The total execution time is therefore:

$$T_{QR-Tot-REL} = T_{proj} + T_{PT-VTM} + \sum_{k=1}^{p-1} \left[2 \cdot T_{r-io}\left(\frac{M}{3}\right) + \frac{3 \cdot p \cdot N}{M} \cdot \mathrm{Max}\left(T_{opr}(k), T_{r-io}\left(\frac{M}{3}\right) \right) \right]$$

where T_{PT-VTM} is the time to accumulate the inner products of the first pivot with the remaining columns, and $T_{opr}(k)$ is the time of the arithmetic operations at the k^{th} iteration. This function is given by:

$$T_{opr}(k) = T_{AXPY}\left(\frac{M}{3} \cdot \frac{q}{p} \cdot (p-k)\right) + T_{INN}\left(\frac{M}{3} \cdot \frac{q}{p} \cdot (p-k-1)\right)$$

Figures 6.6 and 6.8 contain plots of the total execution times as a function of the number of active columns. In Figure 6.6 the time per flop is 0.5 microseconds and in Figure 6.8 the time per flop is 25 microseconds. In both cases the size of the primary memory is held at 100 pages. The LA algorithm is always a lower bound to all the other algorithms. The VBB algorithm outperforms the algorithm for the relational organization, when $p \leq 10$ in Figure 6.6 and when $p \leq 17$ in Figure 6.8. The VM algorithm outperforms the algorithm for the relational organization for $p \leq 41$ in Figure 6.6 and for all p in Figure 6.8.

Figures 6.7 and 6.9 present the total execution time for each algorithm as a function of main storage size. The time per flop is 0.5 microseconds in Figure 6.7 and 25 microseconds in Figure 6.9. The number of active columns is 20. Figure 6.7 is very similar to Figure 6.5, since with a fast CPU the operation is I/O bound. The curves for VBB, VM, and LA possess minimums in Figure 6.9. It is possible to obtain approximate parametric equations of M as a function of the system parameters, by solving minimization problems for each of VBB, VM and

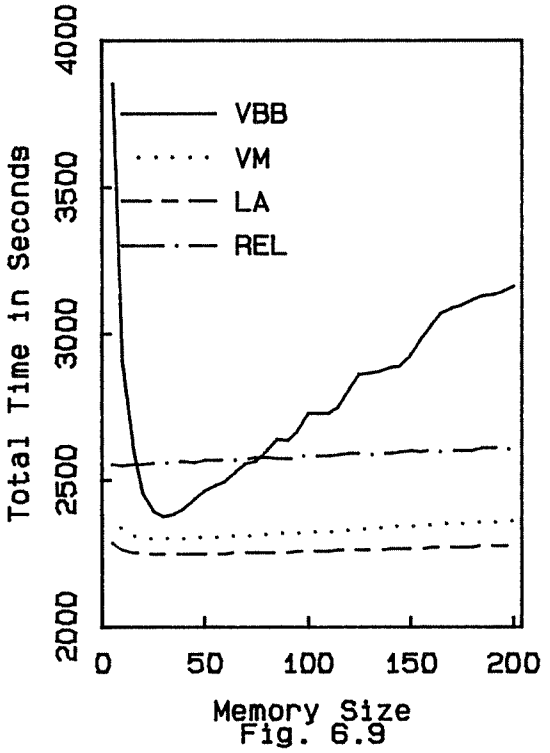


Fig. 6.9

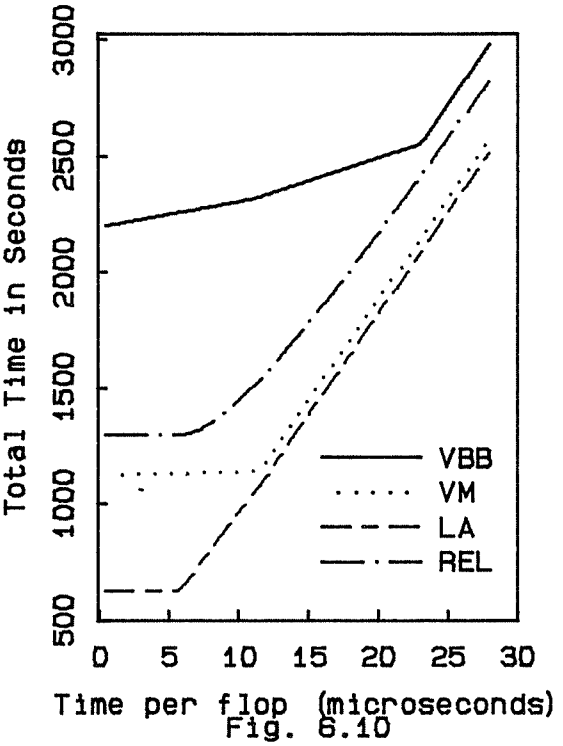


Fig. 6.10

LA. The minimization problem will determine the minimum value of the utilized main storage size M. To illustrate how this can be done, we shall solve the minimization problem for VBB. Throughout we shall ignore the cost of track-to-track seek operations and shall consider M and all the expressions continuous.

First note that the expression for the total execution time of VBB is not a differentiable function of M. The reason is the existence of the "Max" function, which evaluates to the maximum of two (approximately) linear functions in M. However, it is possible to express the optimization problem[2] as a linearly constrained non-linear problem.[3] For example since the coefficient of the term:

$$z = \mathrm{Max}\left(T_{AXPY}\left(q \cdot \frac{M}{5}\right),\ 2 \cdot T_{r-io}\left(\frac{M}{5}\right)\right)$$

is positive, this term can be replaced by the variable z by introducing the constraints

$$z \geq T_{AXPY}\left(q \cdot \frac{M}{5}\right)$$

and

$$z \geq 2 \cdot T_{r-io}\left(\frac{M}{5}\right)$$

Although this procedure will yield a precise solution, it will make the minimization problem much more complicated. We suggest an alternative method which, for a particular set of values of the system parameters, will give us an approximate value of M which minimizes the objective function.

Minimizing the objective function for VBB is equivalent to minimizing

---

[2] minimizing the total execution time as a function of M

[3] with a non-linear objective function

$$f(M) = C \cdot M + \left[\frac{4 \cdot N}{M} - 1\right] \cdot \text{Max}\left(q \cdot \frac{M}{4} \cdot \text{Tflop}, 2 \cdot (\text{Tdac} + \frac{M}{4} \cdot \text{Tio})\right)$$

$$+ \left[\frac{5 \cdot N}{M} - 1\right] \cdot \text{Max}\left(q \cdot \frac{M}{5} \cdot \text{Tflop}, 2 \cdot (\text{Tdac} + \frac{M}{5} \cdot \text{Tio})\right)$$

where

$$C = \frac{11}{10} \cdot \text{Tio} + \frac{q \cdot \text{Tflop}}{4} + \frac{\text{Max}(q \cdot \text{Tflop}, \text{Tio})}{5}$$

We shall ignore the existence of an upper bound on M and note that 5 is a lower bound on M. In other words, the minimization problem is constrained with $M \geq 5$. The linear terms of:

$$\text{Max}\left(q \cdot \text{Tflop} \cdot \frac{M}{4}, 2 \cdot (\text{Tdac} + \frac{M}{4} \cdot \text{Tio})\right)$$

and

$$\text{Max}\left(q \cdot \text{Tflop} \cdot \frac{M}{5}, 2 \cdot (\text{Tdac} + \frac{M}{5} \cdot \text{Tio})\right)$$

are equal for:

$$M_1 = \frac{8 \cdot \text{Tdac}}{q \cdot \text{Tflop} - 2 \cdot \text{Tio}}$$

and

$$M_2 = \frac{10 \cdot \text{Tdac}}{q \cdot \text{Tflop} - 2 \cdot \text{Tio}}$$

respectively. Note that $M_2 > M_1$, when $q \cdot \text{Tflop} > 2 \cdot \text{Tio}$.

For the minimization of f(M), we need to consider two cases:

Case 1: $q \cdot \text{Tflop} \leq 2 \cdot \text{Tio}$. This corresponds to an I/O bound case and minimizing f(M) is equivalent to minimizing:

$$f_1(M) = \frac{18 \cdot N \cdot Tdac}{M} + M \cdot \left[\frac{q \cdot Tflop}{4} + \frac{Max\,(\,(q \cdot Tflop,\, Tio\,) + Tio}{5} \right]$$

The minimum of the function (convex over $M > 0$) is obtained at:

$$M_{o1} = Sqrt\,(\,\frac{360 \cdot N \cdot Tdac}{5 \cdot q \cdot Tflop + 4 \cdot (\,Max\,(\,q \cdot Tflop,\, Tio) + Tio\,)}\,)$$

If $M_{o1} \geq 5$ then the optimum is achieved at M_{o1}, otherwise the optimum is achieved at $M = 5$.

Case 2: $q \cdot Tflop > 2 \cdot Tio$. Here we distinguish three subcases:

I $\quad 5 \geq M_2$

II $\quad M_1 \leq 5 < M_2$

III $\quad 5 < M_1$

Case I corresponds to the CPU bound case and minimizing the objective function is equivalent to minimizing:

$$f_2(M) = \frac{11}{10} \cdot Tio \cdot M$$

over $M \geq 5$ and, therefore, the minimum is achieved at $M = 5$.

For II, we minimize the function f(M) over the sets $(5, M_2)$, (M_2, ∞), $\{5\}$, and $\{M_2\}$. Over $(5, M_2)$, minimizing f(M) is equivalent to minimizing

$$f_3(M) = \frac{7 \cdot Tio + 2 \cdot q \cdot Tflop}{10} \cdot M + \frac{10 \cdot N \cdot Tdac}{M}$$

This function is convex over $(0, \infty)$ and achieves its minimum at:

$$M_{o3} = 10 \cdot \mathrm{Sqrt}\left(\frac{N \cdot Tdac}{7 \cdot Tio + 2 \cdot q \cdot Tflop} \right)$$

If $M_{o3} < 5$ we take $M_{o3} = 5$; if $M_{o3} > M_2$ we take $M_{o3} = M_2$. Over (M_2, ∞), minimizing f(M) is equivalent to minimizing $f_2(M)$. Therefore, for case II, the minimum is achieved at the value of M which minimizes:

$$\min(f(5), f(M_2), f(M_{o3}))$$

For III, we need to consider the sets $(5, M_1)$, (M_1, M_2), (M_2, ∞), $\{5\}$, $\{M_1\}$, and $\{M_2\}$.

Over $(5, M_1)$, the objective function is f_1. If $M_{o1} < 5$ we take $M_{o1} = 5$. If $M_{o1} > M_1$, we take $M_{o1} = M_1$. Over (M_1, M_2) the objective function is f_3. If $M_{o3} < M_1$ we take $M_{o3} = M_1$; if $M_{o3} > M_2$, we take $M_{o3} = M_2$.

Finally, over (M_2, ∞), minimizing f(M) is equivalent to minimizing f_2. Therefore, for case III, the minimum is achieved at the value of M which minimizes:

$$\min(f(5), f(M_1), f(M_2), f(M_{o3}), f(M_{o1}))$$

The following example will illustrate how an approximation to the optimum M can be obtained through this type of analysis. Let Tflop = 25 microseconds. Then $M_1 \approx 27$, $M_2 \approx 33$, $M_{o1} \approx 38$ and $M_{o3} \approx 29$. Therefore we are in case III and the optimum is achieved at M_2.

The total execution time as a function of the time per flop for each algorithm is plotted in Figure 6.10. The critical points in these curves are points where "Max" terms in the expressions for the total execution times become CPU bound. For VBB, the inner product term consumes about 50% of the total execution time. The expression for the inner product term includes the term

$$\frac{p \cdot (p-1)}{2} \cdot T_{INN}\left(q \cdot \frac{M}{4}\right)$$

The contribution of this term to the total execution time is substantial. That is why the curve for VBB increases sharply for the smaller values of Tflop. The two critical points in the curve for VBB, are the points where the terms

$$\text{Max}(q \cdot \text{Tflop}, \text{Tio})$$

and

$$\text{Max}\left(\text{Tflop} \cdot q \cdot \frac{M}{b}, 2 \cdot T_{r-io}\left(\frac{M}{b}\right)\right)$$

become CPU bound. For the second max-function b = 4 for the inner products and b = 5 for the AXPY. For the first max-function the value of Tflop is approximately 11 microseconds and for the second 22.

The critical point of VM corresponds to the value of Tflop which makes the term

$$\text{Max}\left(T_{AXPY}\left(q \cdot \frac{M}{4}\right), T_{r-io}\left(\frac{M}{4}\right)\right)$$

CPU bound, namely Tflop $\approx$ 11 microseconds. The critical point of LA corresponds to the value of Tflop which makes the term

$$\text{Max}\left(T_{AXPY}\left(q \cdot \frac{M}{5}\right) + T_{INN}\left(q \cdot \frac{M}{5}\right), T_{r-io}\left(\frac{M}{5}\right)\right)$$

CPU bound, that is Tflop $\approx$ 6 microseconds.

6.4. Multidisk Algorithms for QR

The performance evaluation of the statistical building blocks assumed one I/O subsystem for operations involving only reads, and two I/O subsystems for operations which produce temporary files. With the current trends in computer technology it is reasonable to assume that in the foreseeable future, very fast CPU's will be available at affordable costs. Moreover, the rate of increase in CPU speeds is much greater than the rate of increase for secondary storage

modules. It is to be expected that in the near future even computationally intensive operations, such as X'X and the QR decomposition, will become I/O bound if the underlying data sets are large. Therefore, proposing algorithms and access schemes which only try to render an operation CPU bound is not sufficient. With I/O bound operations it is reasonable to consider multidisks or parallel-readout disks.

It is argued in [BORA83] that although there exists some processor-per-head parallel-readout disks, the improvements in disk technology have been primarily due to increases in track capacity. With the increase in track capacity, the hardware problems of aligning the disk heads for parallel I/O become formidable. The same report also shows that processor per-head disks are not presently cost effective. Boral and DeWitt conclude that "although parallel-readout disk drives could form the basis of high performance, ..., changes in disk technology have rendered this approach questionable and other approaches must be developed to provide high bandwidth mass storage devices" [BORA83]. Therefore, we found it more reasonable to consider secondary storage layouts and parallel algorithms for multidisk configurations.

We shall compare two secondary storage layouts and parallel algorithms for the QR decomposition. As mentioned earlier, the optimal algorithm for the QR decomposition involves O(p) passes over the active columns. At each pass the current Householder transformation is applied while the inner products for the next transformation are accumulated. It is necessary to dedicate some of the disk drives to "read" operations and some to "write" operations. With $2 \cdot k$ parallel disk drives, we shall dedicate k disk drives to read and write operations respectively. The multidisk configuration will, therefore, be logically subdivided into two subsystems. During a pass all the input is going to be accessed from the same subsystem and all the processed pages written to the other. After the current transformation is applied, the two subdivisions will interchange their roles. Figure 6.11 shows such an architecture.

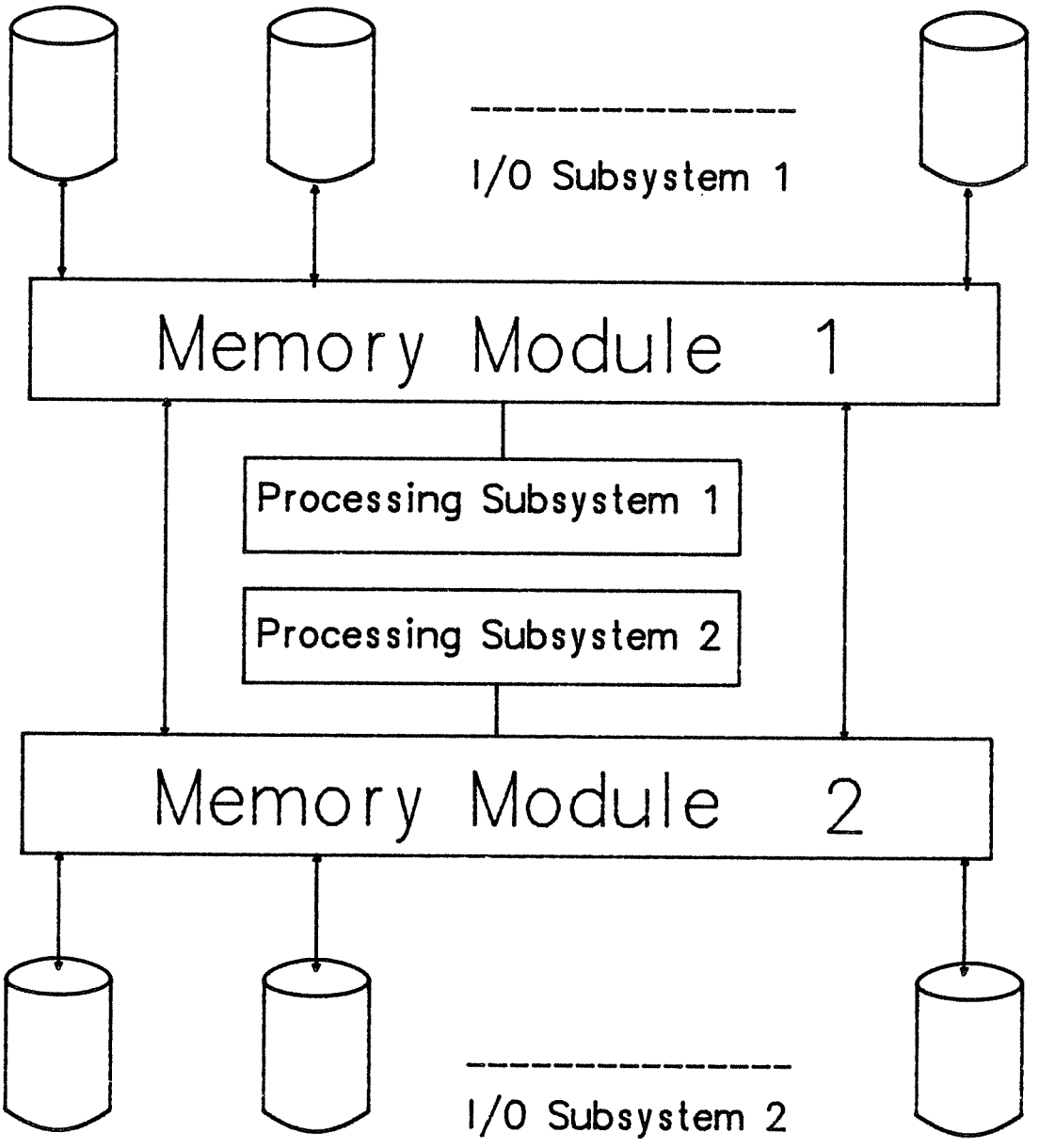

I/O Subsystem 1
Memory Module 1
Processing Subsystem 1
Processing Subsystem 2
Memory Module 2
I/O Subsystem 2

Figure 6.11

The multiprocessor architecture has two processing subsystems. During a run, one processing subsystem will be dedicated to the application of the transformation and the other processors will be dedicated to the accumulation of the inner products for the next transformation. Furthermore, we shall assume $3 \cdot k$ "blocks" of main storage for each processing subsystem. We leave the size of a "block" unspecified, but assume it spans several pages (or tracks). The reason we have chosen blocks rather than pages as units of transfer will be apparent in a moment. Let B be the block size.

The first strategy we have considered is very similar to the two I/O subsystem case. Here the transposed files are assumed to be stored across k of the disk drives. In other words, block i of a column is stored on disk drive $(i-1) \bmod k + 1$ of an I/O subsystem. The $3 \cdot k$ blocks of the main storage for each processing subsystem are divided into three units. The first unit is for holding pages of either the pivot or the next pivot. The second is for holding corresponding pages of a column on which either the transformation is applied or the inner product is accumulated. Finally, the third is for inputing or outputting the next column. The transposed files are processed in k block units in parallel. In a steady state, the system concurrently:

- Reads k blocks of column i
- Applies the transformation to column (i-1) and, when done, transfers it o the second set of processors for accumulating the inner products.
- Accumulates the inner product of column (i-2) with the next pivot
- Writes column (i-3) to secondary store.

It can easily be shown that this algorithm involves approximately, $\frac{N \cdot p}{B \cdot k}$ I/O operations, in which k blocks are read/written in parallel during each I/O operation. The primary storage size must be at least $6 \cdot k$ blocks, for this algorithm to work. However, an advantage of this scheme is the simplicity of the interconnection between the I/O and processing subsystems. The

main storage subsystem could be subdivided into $2 \cdot k$ units, allocating three blocks to each disk drive. Each memory unit will then be interconnected to one and only one disk drive throughout the execution of the program. Another advantage is that we do not need to worry about the relative position of the columns with respect to one another. We shall see below why this is an important consideration.

With the second strategy each transposed file is stored in its entirety on one disk unit. In particular, column i is stored on disk unit $(i - 1) \bmod k + 1$ (the current pivot being stored on unit 1). The main storage module for each of the processing subsystems is divided into two subdivisions. One subdivision is dedicated to the computation and the other to I/O. Unlike the previous scheme, the main storage must be shared among the processors of the same processing subsystem. In order not to be a bottleneck, the bandwidth of the shared memory modules must be approximately k times higher than the previous scheme. The transposed files are processed in $U = \frac{3 \cdot k}{2 \cdot k + 1}$ block units. For the first main processing subsystem, U blocks of the pivot reside in main storage. For the second, U blocks of the next pivot reside in main storage. At each iteration, the transformation is applied to the U blocks of the k columns currently residing in main storage. To see how the algorithm proceeds, we next present the flow of control which specifies the page replacement policy and the concurrent execution. Let $B2 = \text{ceil}(\frac{N}{U})$, $q = \text{ceil}(\frac{p}{k})$, and $X_{i,j}$ be the i^{th} U-block of column X_j.

```
For i := 1 to B2
        For j := 1 to q
                $ Read X_{i,(j-1)·k+1} ··· X_{i,(j-1)·k+k}

                $ Apply the transformation to
                        Case j > 1: X_{i,(j-2)·k+1} ··· X_{i,(j-2)·k+k}
                        Case j = 1:
                                Case i > 1: X_{i-1,(q-1)·k+1} ··· X_{i-1,(q-1)·k+k}
                                Case i = 1: no-op

                $ Accumulate Inner Products of
                        Case j > 2: X_{i,(j-3)·k+1} ··· X_{i,(j-3)·k+k}
                        Case j = 2:
                                Case i > 1: X_{i-1,(q-1)·k+1} ··· X_{i-1,(q-1)·k+k}
                                Case i = 1: no-op
                        Case j = 1:
                                Case i > 1: X_{i-1,(q-2)·k+1} ··· X_{i-1,(q-2)·k+k}
                                Case i = 1: no-op

                $ Write
                        Case j > 3: X_{i,(j-4)·k+1} ··· X_{i,(j-4)·k+k}
                        Case j = 3:
                                Case i > 1: X_{i-1,(q-1)·k+1} ··· X_{i-1,(q-1)·k+k}
                                Case i = 1: No-op
                        Case j = 2:
                                Case i > 1: X_{i-1,(q-2)·k+1} ··· X_{i-1,(q-2)·k+k}
                                Case i = 1: no-op
                        Case j = 1:
                                Case i > 1: X_{i-1,(q-3)·k+1} ··· X_{i-1,(q-3)·k+k}
                                Case i = 1: no-op
```

This algorithm involves approximately $\frac{2 \cdot N \cdot p}{3 \cdot k \cdot B}$ I/O operations. During each I/O operation, $k \cdot U$ blocks are read or written in parallel. Therefore, from the standpoint of the number of I/O operations, it performs better than the previous scheme. Moreover, the minimum size of the main storage is $4 \cdot k + 2$ blocks (vs. $6 \cdot k$ blocks for the first scheme). However, there are a number of serious drawbacks to this algorithm. The first is the relative positioning of the columns. In the worst case, columns 1 through p for the first pass are all stored on the same disk unit. Even if a more complex algorithm is introduced and the columns are rearranged for

the next pass, the scheme could still involve extra I/O if a stable version of the QR algorithm is chosen (for example pivoting on the columns with the largest norm). As mentioned earlier, the second drawback is the high memory bandwidth. Finally, a third disadvantage is that, unlike the first scheme, the interconnection between the memory modules and the I/O subsystems is not static. More specifically, the transformed columns are written back shifted one disk unit with respect to their previous position. When the same blocks are read for the next pass, the interconnection is straight.

In summary, laying out the transposed files across the disk units provides a simpler, more coherent and less restrictive strategy for the QR decomposition. Moreover, if an operation involves only very few of the columns (for example AXPY), the first strategy substantially reduces the access time.

6.5. Conclusions

We have presented vector building block, vector-matrix, and direct algorithms for the QR decomposition. For the transposed organization these were labeled VBB, VM, and LA, respectively. The direct algorithm for the relational organization was labeled REL. One basic observation is that with the QR decomposition there is an iterative decrease in the number of active columns. Therefore, in general, the algorithms with the transposed secondary storage organization performed better than the direct algorithm with relational secondary storage organization. In fact the LA algorithm with transposed secondary organization performed better than the direct implementation with the relational secondary storage organization, for all values of the number of active columns, the main storage size, and the time per flop.

For the number of page transfers, the coefficients of p^2 for VBB, VM, LA, and REL were, respectively, 2.5, 1.5, 1, and 2. This shows the optimality of LA and also the fact that

the VM algorithm involves fewer page transfers than REL. Therefore, for all values of p, the total I/O time of VM is less than the total I/O time of REL. However, with respect to the total execution times, and with an I/O bound case[4] VM per forms better only for $p \leq 41$. The reason is that the VM implementation uses only one I/O subsystem in the "vector times matrix" steps. On the other hand, with the look-ahead scheme for the relational secondary storage organization both I/O subsystems are utilized during each pass.[5]

With the CPU bound case both VBB and VM perform relatively better than the relational organization. This occurs as computation cost dominates the total execution time, and the initial projection step is a substantial portion of the I/O startup time for the the direct algorithm REL.

For the total execution time as a function of main storage size, we introduced a methodology for predetermining the main storage size which minimizes the cost function. As indicated earlier, the curves for VBB, VM, and LA possess minima and our method enables preallocation of the optimum (or near optimum) number of pages. We should also observe that, unlike X'X, only a constant number of main storage subdivisions (and hence pages) are needed for the optimal LA algorithm with the transposed secondary storage organization.

Finally, as in the case of X'X, the LA algorithm becomes CPU bound with smaller values of Tflop. Rendering the execution of an operation on a large data set CPU bound is one way of solving the I/O problem.

The discovery and optimality of the LA algorithm suggested a multiprocessor/multidisk architecture with two I/O subsystems and two processing subsystems. In this special purpose architecture, the I/O subsystems are dedicated to the input and output of the data pages of the

---

[4] that is time per flop = 0.5 microseconds

[5] except the first (for the inner products of the first pivot)

currently active columns during a pass. The processing subsystems are dedicated respectively to the application of the current transformation and the accumulation of the inner products for the next transformation. We compared two alternative secondary storage strategies for storing the transposed files across the parallel disk drives. With the first strategy, the data pages of the transposed files are stored across the parallel disk drives. With the second strategy each column is stored in its entirety on one disk drive. The columns are then distributed across the disk drives. We came to the conclusion that although the second strategy involves, on the average, fewer I/O operations and requires a smaller main storage, the first strategy might be preferable because of the simplicity of the interconnections and the layout. Furthermore, in some cases the second strategy results in a considerable degradation of performance.

CHAPTER 7

The Singular Value Factorization

7.1. Introduction

The Singular Value Factorization attempts to compute an n by p matrix W, a p x p orthogonal matrix V, and a p x p diagonal matrix D such that

$$X = W \cdot D \cdot V'$$

The p columns of W are the eigenvectors associated with the p largest eigenvalues of XX'. The columns of V are the orthonormalized eigenvectors of X'X. The non-ascending ($s_1 \geq s_2 \geq s_3 \geq \cdots \geq s_p$) diagonal elements of D are the non-negative square roots of the eigenvalues of X'X (also called the "singular values" of X). Both D and V are of order O (p^2), and therefore are stored in primary memory. W, however, is of order O($n \cdot p$), and should be stored on disk. With the Golub-Reinsch algorithm [WILK71], the decomposition precedes in two main steps:

Step 1 - **Reduction to Bidiagonal Form:** This is accomplished by premultiplying X with p Householder transformations and postmultiplying it with p - 2 Householder transformations. The left transformations zero subdiagonal column elements. The right transformations zero superdiagonal row elements. The left and right transformations are applied consecutively. Each of the first p - 2 left transformations are followed by a right transformation. The k^{th} left transformation zeroes elements k+1 to n of the k^{th} column. The k^{th} right transformation zeroes elements k+2 to p of the k^{th} row. The last two left transformations zero the subdiagonal elements of the last two columns. When these transformations are applied X will be reduced to

upper bidiagonal form.

Step 2 - **Decomposition of the Bidiagonal Matrix:** This is done through the implicitly shifted QR algorithm [WILL71, STEW73]. The details of this algorithm will not be discussed here. It is an infinitely iterative algorithm which tries to render the superdiagonal elements of the bidiagonal matrix negligible. Each iteration consists of the pre and post multiplication of the bidiagonal matrix with Givens rotations. At the end of each iteration the resulting matrix remains bidiagonal. If B is the bidiagonal p x p matrix, this algorithm results in orthogonal matrices J and T (both p x p) such that

$$J \cdot B \cdot T' = D \quad (\text{or } B = J' \cdot D \cdot T)$$

Let the n by p matrix S be given by:

$$S = \begin{bmatrix} J' \\ O \end{bmatrix}$$

where O is the (n-p)xp zero matrix. Then W is:

$$W = L_1 \cdot L_2 \cdots L_p \cdot S$$

where the L_i's are the left Householder transformations. V' is:

$$V' = T \cdot R_{p-2} \cdots R_1$$

where the R_i's are the right transformations.

Next, we will describe two approaches which introduce modifications to the Golub-Reinsch algorithm for large matrices.

The first is due to Cuppen [CUPP81]. Cuppen suggests that W and V not be updated and stored explicitly while applying the transformations and the rotations. He proposes storing the transformations which form these matrices and then, after the matrix is accepted as diagonal, the transformations can be applied in the correct order to, say, an identity matrix, yielding W and V. For the Householder transformations this approach is not new. It is suggested, for example, in [STEW73] and we used it in the QR decomposition. However, Cuppen also suggests storing the rotations.[1] Each rotation is characterized by three numbers: the angle and the two row (column) indices of the rotation vectors. The implicitly shifted QR algorithm which is used in the bidiagonalization phase of the conventional GR algorithm is an infinitely iterative algorithm. In the worst case this could involve a large number of rotations. However, experience has shown that the average number of sweeps to reduce a superdiagonal element to zero (or acceptably near zero) is approximately two. Therefore, in most cases an O (p) storage will be sufficient for the rotations. But Cuppen points out that if the "ultimate shift" strategy is used instead of the implicit shift, the actual number of sweeps per eigenvalue is reduced to one. The ultimate shift involves two passes. In the first pass the eigenvalues of the bidiagonal matrix are evaluated using the implicit shift, but without storing the rotations. Of course this pass will involve several sweeps. In the second pass, the eigenvalues are used to determine the shifts and in this case the rotations are stored (or accumulated). It can be shown mathematically [PARL80] that each eigenvalue will involve exactly one sweep. Therefore, theoretically, we can guarantee that the amount of storage for the rotations will be O (p). However, due to roundoff errors, the actual number of sweeps per eigenvalue could be more (in most cases two).

---

[1] In the conventional GR algorithm, each rotation is immediately applied to the current W and V.

The second approach which also attempts to achieve substantial savings in computation and secondary storage references was proposed by Chan [CHAN82]. His approach is to first triangularize the matrix X using Householder transformations and then apply the Golub-Reinsch algorithm to the n by n upper triangular matrix.[2] Therefore if:

$$Q' \cdot X = \begin{bmatrix} R \\ O \end{bmatrix}$$

and

$$R = L \cdot D \cdot V'$$

then

$$W = Q \cdot \begin{bmatrix} L \\ O \end{bmatrix}$$

Although Chan's motivation is primarily optimizing the computation, this approach also results in minimizing the I/O overhead.[3] W can be obtained through either: (1) applying the left transformations[4] to the first p columns of the identity n by n matrix and then performing a matrix multiplication with L, or (2) applying the transformations to L (augmented by an (n - p) by p zero matrix) in the correct order.

Both of these approaches suggest that for $n \gg p$, it is better to postpone the construction of W until all the transformations and rotations have been determined. To determine the left

---

[2] To take advantage of the special form of the triangular matrix in the bidiagonalization step, Chan proposes using Givens rotations instead of Householder transformation. However, this method will be computationally more intensive.

[3] since the rotations are not applied to the current n by p matrix W.

[4] whose u's are stored in the lower triangular part of the transformed X

Householder transformations, the u's of the transformations are stored in the zeroed subdiagonal part of X. The rotations are either stored explicitly (where three numbers determine a rotation) or accumulated in a p by p matrix. Both require at most $O(p^2)$ storage.

An interesting point is that the QR decomposition could itself be used as a building block to the SVF. Moreover, to determine the first p columns of the product of the left transformations, it can be shown that through one sweep we can obtain this product, provided the first p elements of each u and the inner product of the u's are available. With some implementation strategies, the inner product of the u's can be accumulated while applying the transformations.[5] On the other hand, we saw in the previous chapter that the most efficient algorithm of the QR decomposition is the look-ahead (LA) scheme. However, with this algorithm it is not possible to accumulate all the inner products of the u's. The reason is that only the corresponding pages of the pivot, the next pivot, and one of the active columns are resident in primary store. Therefore, there needs to be another pass over the transposed columns which store the u's to accumulate the inner products. This means that $X'X$ can also be used as a building block to SVF.

Next, let us illustrate how it is possible to accumulate the left Householder transformations and W, if the inner product of the u's and the first p elements of each u are available. Suppose we want to evaluate:

$$Q = (I - \rho_1 u_1 \cdot u_1').(I - \rho_2 u_2 \cdot u_2') \cdots (I - \rho_p u_p \cdot u_p')$$

It can easily be shown that:

---

[5] for example, if the matrix is stored partially transposed, or if a horizontal-stripes scheme is used with the completely transposed organization.

$$Q = I + \sum_{i=1}^{p} \sum_{j=i}^{p} c_{i,j}\, u_i \cdot u_j'$$

where the $c_{i,j}$'s are in terms of the ρ_i's and the inner products of the u's. The first p elements of the k^{th} row of $u_i \cdot u_j'$ is given by:

$$u_{i,k} \cdot [\, u_{j,1}, u_{j,2}, \cdots, u_{j,p} \,]$$

where the first (j-1) elements of u_j are zero. Let U be the matrix of the u's and Q_p the first p columns of Q. Let T be given by:

$$T_{i,j} = \sum_{k=i}^{j} c_{i,k} \cdot u_{k,j} \;;\; 1 \le i \le j \le p$$

Then the rows k through m of Q_p can be obtained by accessing the rows k through m of U, and performing a matrix multiplication of the (m-k)+ 1 by p submatrix of U with the upper triangular matrix T. If $k \le p$ we need to add 1 to the diagonal elements to this matrix product. Therefore,

$$W = Q_p \cdot L = (I_p + U \cdot T) \cdot L = I_p \cdot L + U \cdot (T \cdot L)$$

and W could be obtained basically through the product of the n by p matrix U with the p by p matrix $T \cdot L$.

For the performance evaluation we shall not specify any particular scheme for the diagonalization step. However we shall use Chan's approach in first performing a QR decomposition of X, and then applying the GR algorithm (or a modification) to the upper triangular matrix. For the transposed organization the total execution time is:

$$T_{SVF} = T_{QR} + T_{U'U} + T_{MPT}$$

where, T_{QR} is the total time for the QR decomposition step, $T_{U'U}$ is the time for the "U'U"

operation to evaluate the inner product of the u's. T_{MPT} is the time to perform the matrix multiplication of an n by p matrix with a p by p matrix. We shall consider vector building blocks, vector-matrix, and direct implementations for the three main steps of forming the QR decomposition, the U'U operation, and the matrix product step.

If X is processed stripe-wise, accumulating the inner products of the u's will save a sweep.[6] Therefore, another strategy is to process the transposed files in horizontal stripes. In other words, it is feasible to divide the primary memory into O (p) logical subdivisions, and always reference a horizontal stripe of the active columns. This horizontal access is performed for every left transformation. That is for the k^{th} transformation, if the current stripe consists of rows i through j, the elements i through j of u_1 through u_{k-1} will be accessed. In addition, rows i through j of columns k through p of the transformed X will also be accessed. The u's are accessed to accumulate their inner products with u_k. Besides requiring a substantial number of random seeks, this strategy will also require more page transfers. In fact the number of page transfers for the look-ahead scheme, including the extra pass to accumulate the inner products is:

$$N \cdot (2 \cdot p + \frac{p \cdot (p+1)}{2} - 1 + \frac{p \cdot (p-1)}{2})$$

For a stripe-wise algorithm the number of page transfers is:

$$N \cdot (p + p \cdot (p-1) + \frac{p \cdot (p-1)}{2})$$

Therefore, ignoring the linear terms, the look-ahead strategy involves $N \cdot O(p^2)$ page transfers whereas the stripe-wise strategy will involve $N \cdot O(\frac{3}{2} \cdot p^2)$ page transfers.

---

[6] by performing the "U'U" operation while applying the transformations

For the relational secondary storage organization, we must first construct a partially transposed file of the active columns. Since the u's of the Householder transformations are accessed during the QR decomposition step, it is possible to accumulate the inner products of the u's while applying the transformations. The total execution time for the relational organization is:

$$T_{REL} = T_{QR-Tot-REL} + T_{MP-REL}$$

$T_{QR-Tot-REL}$ is the total execution time for the QR decomposition step (see Chapter 6). This term includes the execution time to construct the partially transposed file of the active columns, as well as the computation time for accumulating the inner products of the u's. T_{MP-REL} is the time to perform the matrix product of the partially transposed file (n by p) with a p by p matrix.

7.2. Algorithms

The implementation of the SVF consists of the following steps:

(1) The QR decomposition

(2) The inner products of the u's

(3) Forming W: After the left rotations are accumulated in a p by p matrix and the matrix product $T \cdot L$ is evaluated, the final phase is to form W.

We have already analyzed in Chapters 5 and 6 the QR decomposition and the U'U step (for accumulating the inner products of the u's). In the following discussion we shall primarily concentrate on the matrix product step. Let us emphasize however, that an implementation of SVF at an abstraction level implies all the three steps are implemented at that level. For example, a vector buiding block implementation of SVF implies that the QR step, the X' X, and the matrix product step are all implemented through vector operations (i.e. inner products, AXPY

operations etc.).

There are six alternative algorithms for the matrix product step.

```
I: For k := 1 to n
       For i := 1 to p
            For j := 1 to p
                 W_kj := W_kj + U_ki · (TL)_ij

II: For i := 1 to p
       For k := 1 to n
            For j := 1 to p
                 W_kj := W_kj + U_ki · (TL)_ij

III: For i := 1 to p
        For j := 1 to p
             For k := 1 to n
                 W_kj := W_kj + U_ki · (TL)_ij

IV: For k := 1 to n
       For j := 1 to p
            For i := 1 to p
                 W_kj := W_kj + U_ki · (TL)_ij

V: For j := 1 to p
       For k := 1 to n
            For i := 1 to p
                 W_kj := W_kj + U_ki · (TL)_ij

VI: For j := 1 to p
       For i := 1 to p
            For k := 1 to n
                 W_kj := W_kj + U_ki · (TL)_ij
```

Algorithms III and VI can be characterized as vector building block (VBB) implementations of the matrix product. There are two types of vector operations used in the VBB

implementation: (1) scaling a vector - invoked p times; (2) AXPY - invoked $p^2 - p$ times.

Algorithms II and V can be characterized as vector-matrix (VM) implementations. In algorithm V a pass is made over U to produce a column of W. Therefore U is accessed p times and W is accessed once. In algorithm II a pass is made over each of the columns of U to produce the partial sums of W. Therefore W is accessed p times and U is accessed once.

Algorithms I and IV process the matrices in horizontal stripes and characterize more efficient and direct implementations (DIR) of the matrix product. Here both U and W are accessed only once. This is the obvious strategy for the relational secondary storage organization. Let us note that IV is perhaps the most natural way of implementing this matrix product.

We also propose another direct and dynamic algorithm for the matrix product. This algorithm is based on algorithm II. Let the main storage be divided into $2 \cdot (p + 1)$ equally sized buffers. Furthermore, assume the transposed columns of U are accessed in $B = \frac{M}{2 \cdot (p + 1)}$ page blocks. The algorithm IIn is:

```
IIn: For b := 1 to N/B
         For i := 1 to p
             For k := 1 to q · B
                 For j := 1 to p
                     W_kj := W_kj + U_ki · (TL)_ij
```

In this algorithm U and W are both accessed once. The strategy here is to process the blocks of each column, adding the "contributions" of the column blocks to every element of W in which they appear as a term. During a run $p \cdot B$ pages are allocated to the current stripe of W. At any given instant the processing subsystem is accumulating the partial contribution for B pages of a column to the current stripe of W. When all the p columns are processed, the current stripe of W is written to the I/O subsystem dedicated to the write operation. This algorithm is

similar in approach to the ST 1 and ST 2 algorithms of X' X.

In the next sections, buffer management algorithms for each of the VBB, VM, and direct implementations are presented.

7.2.1. Vector Building Block

As indicate earlier, there are two vector operations which are used in the vector building block approach. We have already presented the algorithm for the AXPY operation in Chapter 6. For scaling a vector, the main storage is divided into three subdivisions. The first subdivision is allocated to reading the next block of U_i. The second subdivision is allocated to the scaling of the current block, and the third subdivision is for writing the previously scaled block.

SCALE:

$B = \frac{M}{3}$

Read ($U_i^{B.1}$)

For k := 1 to N/B

$ If k < N/B then Read ($U_i^{B.k+1}$)

$ $Y^{B.k} := a \cdot U_i^{B.k}$

$ If k > 1 then Write ($Y^{B.k-1}$)

7.2.2. Vector Matrix

For the vector-matrix strategy we specify an algorithm based on V. The main storage is divided into $2 \cdot (p + 1)$ subdivisions. p subdivisions are allocated to each of the next stripe of U which is being read and the current stripe of U which is being processed. One block subdivisions are allocated to the current block of the column of W being processed and to the block of

the column of W which is being currently written to disk.

VM:

$$B = \frac{M}{2 \cdot (p + 1)}$$

Read ($U_1^{B.1} \ldots U_p^{B.1}$)

For k := 1 to N/B

$ If k < N/B then Read ($U_1^{B.k+1} \ldots U_p^{B.k+1}$)

$ $W_j^{B.k} := [U_i^{B.k} \ldots U_p^{B.k}] \cdot (TL)_j$

$ If k > 1 then Write ($W_j^{B.k-1}$)

7.2.3. Direct Implementations

The first strategy corresponds to IV. Here we divide the M pages of primary store into three units, and each unit into p subdivisions (for the p u's). The first unit is for reading the next M / (3 · p) pages of the u's. The second unit is for accumulating the current stripe of W. The third unit is for holding the previously processed stripe of W which is being transferred to secondary store. With this scheme, the main storage is divided into 3 · p subdivisions. Therefore, there should be at least 3 · p pages of primary store and the overhead for the random seeks will be substantial. However, since the third primary storage unit contains corresponding elements of all the columns of W, W could be stored in a relational organization.

DIR 1:

$$B = \frac{M}{3} \cdot p$$

Read ($U_1^{B.1} \cdots U_p^{B.1}$)

For k := 1 to N/B

$ If k < N/B then Read ($U_1^{B.k+1} \cdots U_p^{B.k+1}$)

$ $W^{B.k}$:= [$U_i^{B.k} \cdots U_p^{B.k}$] · (TL)

$ If k > 1 then Write ($W^{B.k-1}$)

With the second strategy (which corresponds to IIn) the main storage subsystem is divided into $2 \cdot (p + 1)$ equal subdivisions. Let $B = \frac{M}{2 \cdot (p + 1)}$. Initially p B - page blocks are allocated to the first stripe of W. There are three concurrent processes.

(1) The "Read"" process finds a free block (of size B pages) and reads the next B pages of the next column of U. The function "Find" does not return until it finds B free pages. Once the B pages of a column are read, the process sets the corresponding entry in the two dimensional boolean array A. Therefore A [i,k] is set if and only if the k^{th} B - page block of column U_i is read.

(2) The "Accumulate" process first checks if A [i,k] is set. If so, it accumulates the contributions of the corresponding pages of U_i to the current stripe of W. If k = 1 and $U_i^{B.1}$ is processed, the B pages of U_i are allocated to the accumulation of the next stripe of W through the function "Alloc." Otherwise (that is $k \neq 1$), the B pages of U_i are freed. When stripe k of W is constructed, the corresponding entry in the boolean area "Avail" is set.

(3) The "Write" process first allocates the pages of stripe k of W to the accumulation of stripe k+1 of W. Next it checks if the current (or k^{th}) block of W is available to be written, and, if so, writes it to secondary storage.

DIR 2:

$$B = \frac{M}{2 \cdot (p + 1)}$$

/* allocate $p \cdot B$ pages of main store to the first stripe of W */

Alloc ($M^{p \cdot B}$, $W^{p \cdot B, 1}$)

For k := 1 to N/B

$ For i := 1 to p /* the Read process */

If Find (B) then Read ($U_i^{B,k}$)

set (A [i,k])

$ For k := 1 to N/B /* the Accumulate Process */

For i := 1 to p

Check (A [i,k])

For j := 1 to p

$W_j^{B,k} := W_j^{B,k} + U_i^{B,k} \cdot (TL)_{i,j}$

If (k = 1) then Alloc ($U_i^{B,k}$, $W^{p \cdot B, 2}$)

else Free ($U_i^{B,k}$)

Set (Avail [k])

$ For k := 1 to N/B /* Write process */

If k < (N/B − 1) then

Alloc ($W^{p \cdot B,k}$, $W^{p \cdot B,k+2}$)

Check (Avail [k])

Write ($W^{p \cdot B, k}$)

7.3. Cost Functions

For the performance evaluation we shall not consider the overhead of the diagonalization step or the formation of TL. All the processed matrices for these steps are of order O (p^2). This overhead is the same for the transposed and the relational organizations, and it is dependent on the particular schemes that are used (for example, the diagonalization could be performed using Givens rotations or Householder transformations, the diagonalization step could use the conventional "implicit shift" or the "ultimate shift" etc.).

7.3.1. I/O Costs

As indicated earlier, the SVF consists of three main steps: (1) the QR decomposition, (2) the evaluation of U'U, and (3) the accumulation of W. We identified the third step as a matrix product step (the product of an n by p matrix with a p by p matrix). With the transposed secondary storage organization, we shall incorporate corresponding cost functions from the QR, U'U, and matrix product operations for the vector building block, vector-matrix, and, the direct implementations. In other words, the cost function of, for example, the vector building block implementation is the sum of the cost functions of the vector building block implementations for the QR decomposition step, the U'U step and the matrix product step. Here we shall indicate only the cost functions of the matrix product step, since the cost functions of the U'U and QR decomposition steps are available in Chapters 5 and 6.

For the vector building block implementation, the the SCALE building block is invoked p times and the AXPY building block $p^2 - p$ times. The AXPY operation for the matrix product step, unlike the AXPY for the QR decomposition, reads the columns X and Y from two I/O subsystems. It writes the updated Y to the subsystem it was read from. The cost functions below take this into consideration. The number of random seeks for the matrix product step is:

$$2 \cdot p + (p^2 - p) \cdot \left(1 + 10 \cdot \frac{N}{M}\right)$$

the number of page transfers is: $2 \cdot p \cdot N + 3 \cdot (p^2 - p) \cdot N$

and the I/O cost is:

$$2 \cdot p \cdot T_{r-io}(N) + (p^2 - p)\left(T_{r-io}(N) + 10 \cdot \frac{N}{M} \cdot T_{r-io}(\frac{M}{5})\right)$$

For the vector-matrix algorithm, a pass is made over U per column of W. The number of random seeks is: $p^2 \cdot (2 \cdot (p + 1) \cdot \frac{N}{M} + p$

the number of page transfers is: $N \cdot p \cdot (p + 1)$

the I/O cost is:

$$p \cdot \left(p \cdot (2 \cdot (p + 1) \cdot \frac{N}{M} \cdot T_{r-io}(\frac{M}{(2 \cdot (p + 1)}) + T_{r-io}(N)\right)$$

For the direct algorithms we have described two schemes. The first is based on algorithm IV and the second on algorithm IIn. Since with either scheme just one pass is made over U and W, the number of page transfers is $2 \cdot p \cdot N$. For the first direct implementation (based on algorithm IV), the number of random seeks is: $2 \cdot \frac{3 \cdot p^2 \cdot N}{M}$

and the I/O cost is: $2 \cdot \frac{3 \cdot p^2 \cdot N}{M} \cdot T_{r-io}(\frac{M}{3 \cdot p})$

For the second direct implementation (based on algorithm IIn), the number of random seeks is: $2 \cdot p \cdot (2 \cdot (p + 1)) \cdot \frac{N}{M}$

and the I/O cost is: $2 \cdot p \cdot (2 \cdot (p + 1)) \cdot \frac{N}{M} \cdot T_{r-io}(B)$

With the relational secondary storage organization, the partially transposed file of the active columns is first constructed prior to the QR decomposition step. The only difference from the algorithm for the QR decomposition of Chapter 6 is that the inner products of the u's should also be accumulated while applying the transformations. Note that this does not effect the I/O. For the matrix product step the scheme is based on algorithm IV. Therefore, the total number of random seeks is $2 \cdot p + 3$. The total number of page transfers is:

$$R + p \cdot N + (2 \cdot p - 1) \cdot p \cdot N + 2 \cdot p \cdot N$$

and the total I/O is:

$$T_{QR-IO-REL} + 2 \cdot T_{r-io}(p \cdot N)$$

Figures 7.1, 7.2, and 7.3 illustrate, respectively, the number of page transfers, random seeks, and total I/O as functions of the number of active columns. Here the main storage size is held at 200 pages. The vector building block algorithm involves fewer page transfers than the algorithm with the relational organization, only if the number of active columns is less or equal to 5. The vector-matrix algorithm, on the other, involves fewer page transfers for $p \leq 10$. The curves of the total I/O in Figure 7.3 follow very closely the page transfer curves. Note that in Figure 7.2 the VM algorithm generally involves the largest number of random seeks. The reason is that for this algorithm the main storage is divided into O(p) subdivisions. Although this is also the case with the direct algorithms, the number of columns referenced with the direct algorithms is much smaller than the number of columns referenced by VM. The two strategies for the direct implementations performed very similarly, although the strategy based on IIn always performed better.

Figure 7.4 illustrates the total I/O times as a function of the main storage size. Here the number of active columns is held at 10. Note that although the VM strategy involves fewer

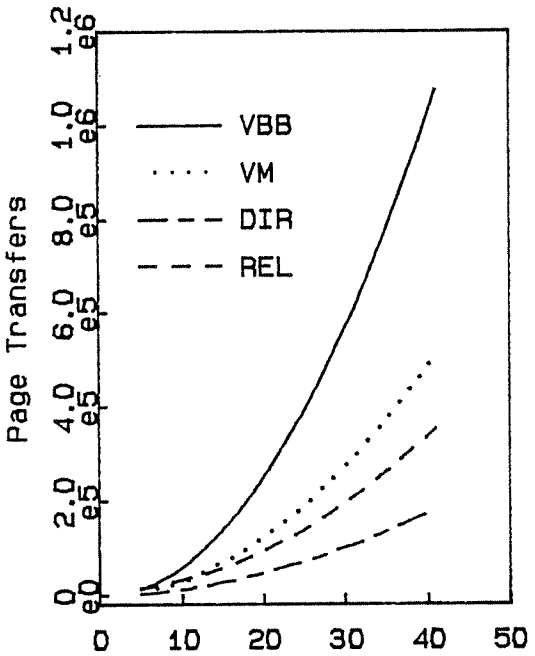


Fig. 7.1

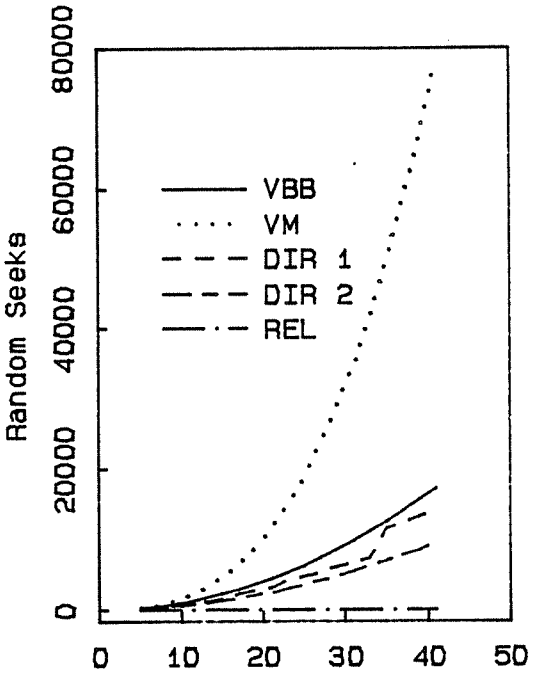


Fig. 7.2

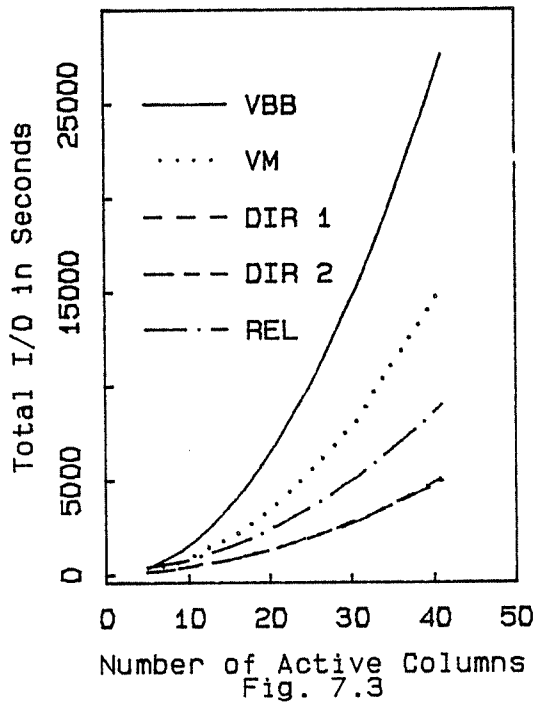


Fig. 7.3

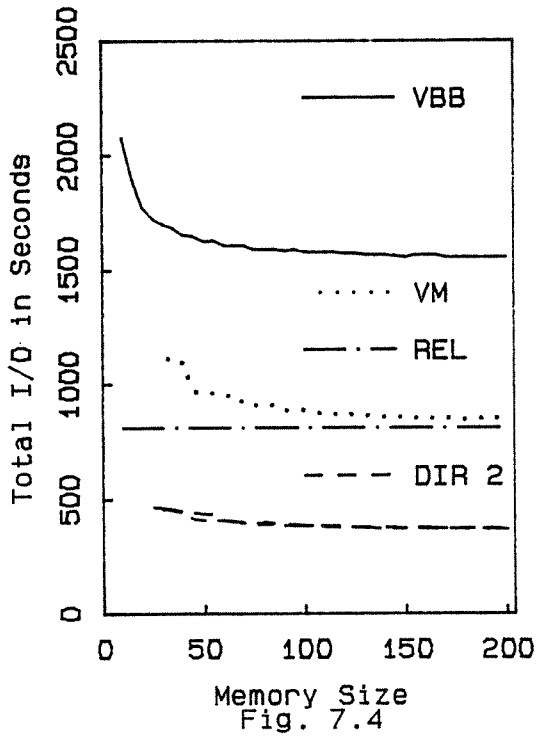


Fig. 7.4

page transfers than REL,[7] the REL curve is a lower bound to the VM curve. This demonstrates the effect of the large number of seeks on the performance of the VM algorithm.

7.3.2. Total Execution Times

For the total execution times we will only present the cost functions for the matrix product step.

The total execution time of the matrix product step with the vector building block implementation is:

$$p \cdot T_{SC} + (p^2 - p) \cdot T_{ot-AXPY}$$

where T_{SC} is the total execution time of the "SCALE" building block given by:

$$\left(T_{r-io}(\frac{M}{3}) + 3 \cdot \frac{N}{M} \cdot T_{VBB-ol1} + Tio \cdot \frac{M}{3} \right)$$

where

$$T_{VBB-ol1} = Max(T_{AXPY}(q \cdot \frac{M}{3}), Tio \cdot \frac{M}{3})$$

$T_{ot-AXPY}$ is the total execution time for the AXPY building block, with the scheme of reading X and Y from two different I/O subsystems. It is given by:

$$T_{r-io}(\frac{M}{5}) + 2 \cdot T_{VBB-ol2} + (\frac{5 \cdot N}{M} - 2) \cdot T_{VBB-ol3} + Tio \cdot \frac{M}{5}$$

where

$$T_{VBB-ol2} = Max(T_{AXPY}(\frac{M}{5}), Tio \cdot \frac{M}{5})$$

and

---

[7] For p = 10, VM involves 31700 page transfers and REL 32000 page transfers

$$T_{VBB-ol3} = Max\left(T_{AXPY}\left(\frac{M}{5}\right),\ 2 \cdot T_{r-io}\left(\frac{M}{5}\right)\right)$$

For the vector-matrix algorithm, the execution time for the matrix product step is p times:

$$p \cdot T_{r-io}\left(\frac{M}{(2 \cdot (p+1)}\right) + \left(\frac{N \cdot (2 \cdot (p+1)}{M} - 1\right) \cdot T_{VM-oll}$$

$$+ T_{VM-ol2} + T_{io} \cdot \frac{M}{(2 \cdot (p+1)}$$

where

$$T_{VM-oll} = Max\left(p \cdot T_{r-io}\left(\frac{M}{(2 \cdot (p+1)}\right),\ T_{INN}\left(p \cdot q \cdot \frac{M}{(2 \cdot (p+1)}\right)\right)$$

and

$$T_{VM-ol2} = Max\left(Tio \cdot \frac{M}{(2 \cdot (p+1)},\ T_{INN}\left(p \cdot q \cdot \frac{M}{(2 \cdot (p+1)}\right)\right)$$

For the first direct algorithm (based on algorithm IV), the total execution time for the matrix product step is:

$$2 \cdot p \cdot T_{r-io}\left(\frac{M}{3 \cdot p}\right) + \frac{3 \cdot p \cdot N}{M} \cdot T_{DIR-ol}$$

where

$$T_{DIR-ol} = Max\left(T_{INN}\left(q \cdot \frac{M}{3} \cdot p\right),\ p \cdot T_{r-io}\left(\frac{M}{3 \cdot p}\right)\right)$$

For the performance evaluation of the second direct algorithm (based on algorithm IIn), we need to distinguish between the I/O and CPU bound cases. Let $B = \frac{M}{2 \cdot (p+1)}$. If

$$T_{AXPY}\left(B \cdot q \cdot p\right) > T_{r-io}(B)$$

then the algorithm will be CPU bound, otherwise it will be I/O bound. The total execution time for the I/O bound case is:

$$p \cdot (2 \cdot (p + 1)) \cdot \frac{N}{M} \cdot T_{r-io}(B) + p \cdot T_{r-io}(B) + T_{AXPY}(B \cdot q \cdot p)$$

and for the CPU bound case is:

$$(p + 1) \cdot T_{r-io}(B) + T_{AXPY}(N \cdot p^2 \cdot q)$$

The total execution time for the direct algorithm for with the relational organization is:

$$T_{QR-Tot-REL} + T_{r-io}(\frac{M}{3}) + \frac{3 \cdot p \cdot N}{M} \cdot Max(T_{INN}(q \cdot \frac{M}{3} \cdot p), Tio \cdot \frac{M}{3}) + Tio \cdot \frac{M}{3}$$

where $T_{QR-Tot-REL}$ is the total execution time for the QR decomposition step. However, as indicated earlier, this term also incorporates the CPU times for accumulating the inner products of the u's.

Figures 7.5 and 7.6 present the curves of the total execution times as functions of the number of active columns. The main storage size is held at 200 pages. In Figure 7.5, the time per flop is 0.5 microseconds and in Figure 7.6 it is 25 microseconds. As was expected, the shape and behavior of the curves in Figure 7.5 are quite similar to that of Figure 7.3. VBB and VM outperform the direct algorithm for the relational organization for $p \leq 5$ and 7 respectively. However, in Figure 7.6, VBB and VM perform better than the relational organization if the number of active columns are less than 6 and 15 respectively. The reason is that with slower CPU's (hence computation bound execution), the fixed "start-up" I/O overhead for the relational organization[8] offsets the relative gains in I/O access times. In other words, for the CPU bound case, the total execution time consists of two main terms: (1) an overhead term - due to I/O start up, and (2) the dominanting computation time. Therefore, for smaller values of p, the

[8] in other words the step forming the partially transposed file

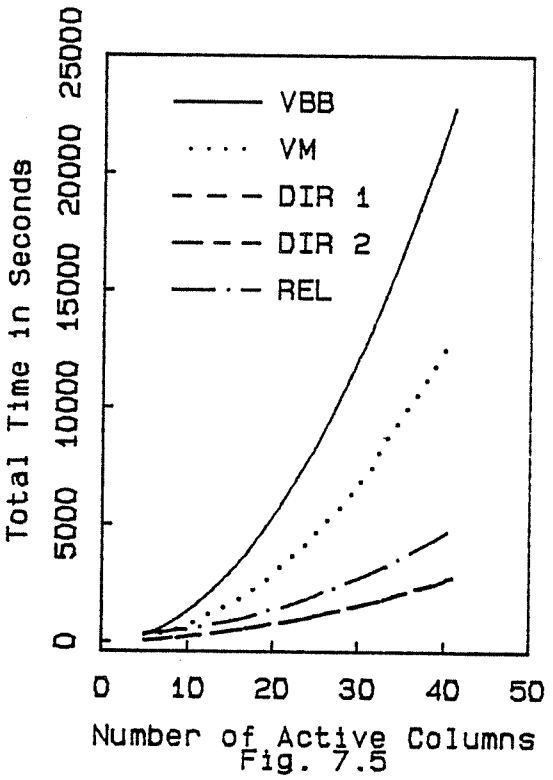

Total Time in Seconds
0
5000
10000
15000
20000
25000
VBB
VM
DIR 1
DIR 2
REL
0
10
20
30
40
50
Number of Active Columns

Fig. 7.5

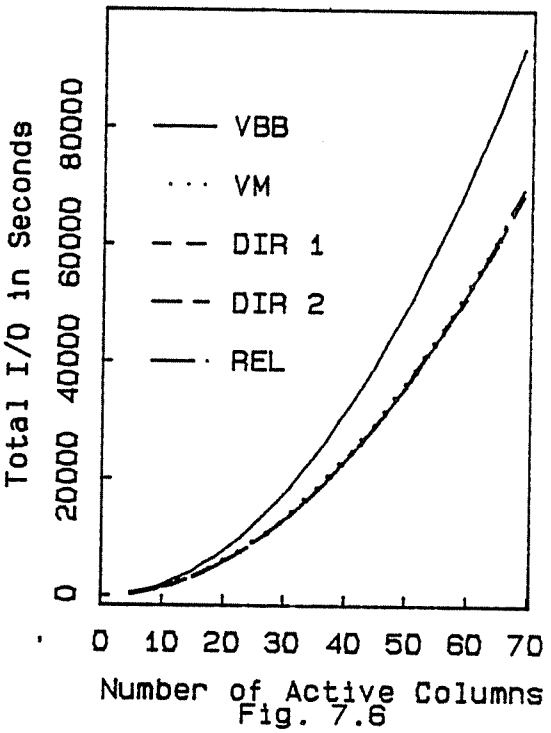

Total I/O in Seconds
0
20000
40000
60000
80000
VBB
VM
DIR 1
DIR 2
REL
0
10
20
30
40
50
60
70
Number of Active Columns

Fig. 7.6

projection step is a significant percentage of the overhead term for the relational organization.

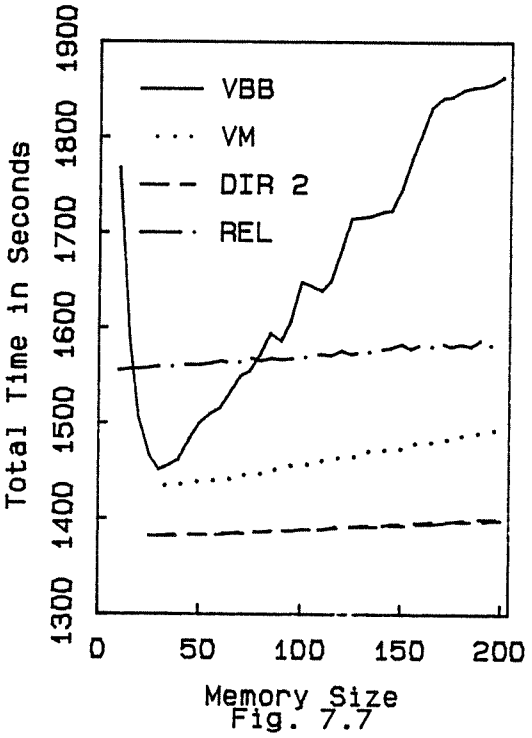
In Figure 7.7 the total execution time as a function of main storage size is presented. The number of active columns is 10 and the time per flop is 25 microseconds (CPU bound case). Similar to the VBB curve for the QR decomposition (Figure 6.9), the VBB curve for SVF possesses a sharp minimum. For the other curves we observe that the main storage size does not effect the performance significantly.

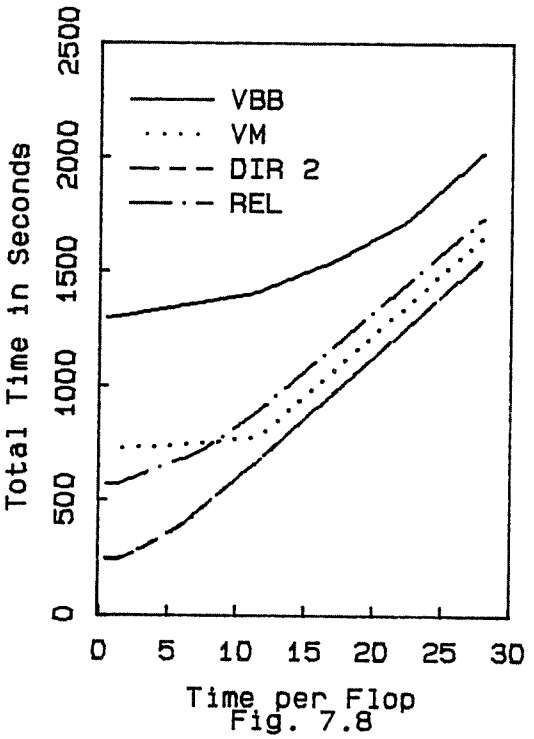
In Figure 7.8 we have the curves of the total execution times as functions of the time per floating point operation. The direct implementations and the algorithm for the relational organization become CPU bound when the time per floating point operation exceeds, approximately, 1.2 and 1.1 microseconds per flop, respectively. The corresponding values for the matrix product step of VBB and VM are 11 and 12 microseconds per flop. Therefore, the direct implementation will become CPU bound for much smaller values of the time per floating point operation.

In Chapters 5 and 6 we proposed a parallel interconnection network for X'X (ZETA networks) and a multiprocessor/multidisk organization for the QR decomposition. Here we shall propose a cyclic shift interconnection network for the matrix product step of SVF. This will complete the set of special purpose multiprocessor architectures for the three main steps of SVF.

7.4. An Interconnection Network for Matrix Products

The interconnection we propose for the matrix product step is based on algorithm VI. The approach here is to let each column of U apply its contribution in producing a column of W. In other words, if columns one through p of U accumulate their contributions to the construction of column j of W (in any order), then when the last column accumulates its contribution, the j^{th} column of W is obtained. Therefore it is clear that we need p stages. However, at


Total Time in Seconds
1300 1400 1500 1600 1700 1800 1900
VBB
VM
DIR 2
REL
0 50 100 150 200
Memory Size
Fig. 7.7


Total Time in Seconds
0 500 1000 1500 2000 2500
VBB
VM
DIR 2
REL
0 5 10 15 20 25 30
Time per Flop
Fig. 7.8

each stage it is possible to process all the columns of W in parallel. That is, if there are p processors at each stage, then it is possible to let the p processors at each stage apply the contributions of the p columns of U to a column permutation of W in parallel. The problem here is how to interconnect the p processors of one stage with the p processors of the next stage so that all the columns of U pair up with all the columns of W. This should sound familiar since we solved a similar problem with ZETA networks. However there are some important differences. First, we are not interested in forming all the possible set of pairs for a string of p inputs. We are only interested in interconnections which will pair every element in a set of p inputs (the columns of U) with every element of another set of p inputs (the columns of W). Second, and as a result of the first observation, the shuffling is required for only one set of columns (the columns of U or the columns of p). Therefore, as we shall show, all that is needed is a multistage cyclic shift network, similar to the δ networks characterized by Lenfant [LENF78]. We should also note that the multistage network we propose for the matrix product step, works for any even p and not just for values of p that are powers of two.

An example of the multistage cyclic shift network for $p = 4$ is shown in Figure 7.9. The main property of this network is that U_k is paired with $W_{(j + k - 2) \bmod p + 1}$ at stage j. Clearly, if this property holds for every k, all the columns of W will be constructed in p stages.

For the cyclic shift network, the intermediate processors each have two inputs and two outputs. One output-input connection is for transferring pages of the same column of U, and hence each column operates in a row of the interconnection. Therefore, one of the interconnections between two adjacent processors of the same row is straight. In other words the first output of processor k at stage j is connected to the first input of processor k at stage $j + 1$, and the interconnection is used to transfer the pages of column U_k. In order to satisfy the criteria of the cyclic order of operations, the second output of processor k is interconnected to the second input

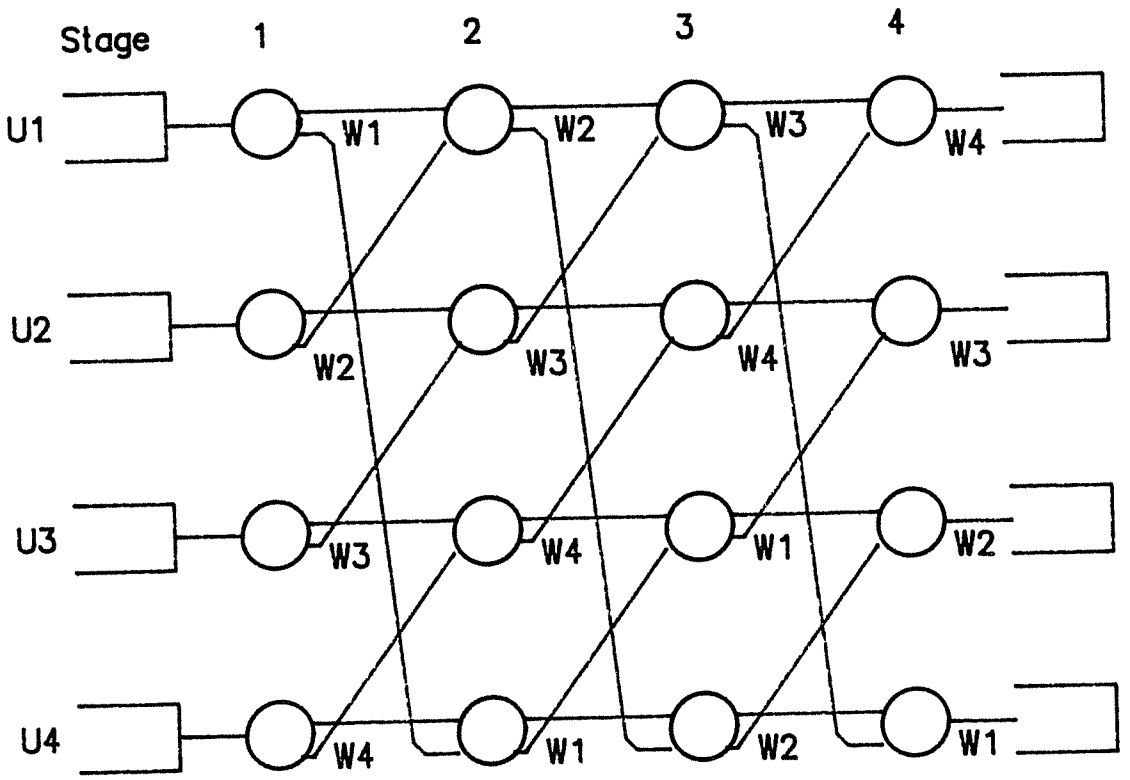


Figure 7.9

of processor ((k - 2 + p) mod p) + 1 at stage j + 1.

In an ideal situation, if there are $2 \cdot p$ disk drives so that p disk drives contain the p columns of U, the interconnection network will act as a pipeline. Thus when a "stripe" of W passes through the pipeline the output can be written in parallel to the other p disk drives. Although it is not reasonable to assume the availability of p drives, if p^2 processors are available and interconnected in p stages as specified above, the scheme could easily be used with the relational organization. With two I/O subsystems, the pages of U are accessed from one subsystem, transposed in a pre-processing step, and processed in the pipeline. After p stages the pipeline will start yielding the (transposed) pages of W. In a post-processing step the transposed pages of W could be combined in order to store W relational. As in the case with ZETA networks, p processors would suffice to emulate the p - stage network, since the interconnections are identical at each stage.

7.5. Conclusions

We have presented a new method for evaluating the SVF of a data matrix X, which consists of three main steps: (1) the QR decomposition of X, (2) the evaluation of $U'U$, and (3) the matrix product step for forming W. An important observation here is that with direct implementations the construction of W requires only O (1) passes over an n by p matrix. This constitutes a substantial savings in data accesses compared to the more conventional method of constructing W. The usual approach is to apply the Householder transformations to the first p columns of the n by n identity matrix (forming Q_p) and then apply the left rotations to Q_p. This would require at least O (p) passes over an n by p matrix. With the relational secondary storage organization, the accumulation of the inner products of the u's (the $U'U$ step), was incorporated in the QR decomposition step. Consequently, the construction of W required just

one pass over the n by p matrix U.

With the transposed secondary storage organization, it is important to analyze the contribution of the QR decomposition and U' U steps to the cost functions. For the percentage of the total I/O as a function of the number of active columns, the QR decomposition step involved about 38% percent of the total I/O and the U' U step about 15%, with the VBB strategy. The corresponding percentages for the VM strategy were, respectively, 45% and 15%. However, for the direct implementations, the percentage of the QR decomposition step went up from about 65% to 93% when the number of active columns was varied from 5 to about 70. The percentage of the U' U step went down from about 11% to 2%. Table 7.1 summarizes these percentages.

Strategy	QR	U' U
VBB	38%	15%
VM	45%	15%
DIR	65% to 94%	11% to 2%

Table 7.1

Therefore, the QR step constitutes a substantial percentage of the total I/O for the algorithms with the transposed organization. For the direct implementations it is at least 65% of the total I/O. This means a multiprocessor/multidisk architecture which enhances the performance of the QR decomposition step will also substantially enhance the performance of SVF.

For the percentages of the total execution times, Table 7.1 is more or less representative of the I/O bound cases. However, for the CPU bound cases all three strategies showed similar behavior, with the QR step constituting approximately 40% of the total execution time and the U' U approximately 20%. This was to be expected since the total number of floating point

operations for U' U is approximately $0.5 \cdot p^2 \cdot N \cdot q$. On the other hand, the number of floating point operations for the QR and matrix product steps are, approximately, $p^2 \cdot N \cdot q$.

For the number of page transfers the coefficients of p^2 for VBB, VM, DIR, and REL were, respectively, $6.5 \cdot N$, $3 \cdot N$, N and $2 \cdot N$. Therefore, unlike the case with the QR decomposition, the VM algorithm involves more page transfers than REL. In the I/O bound case, the vector-matrix scheme outperformed the direct algorithm with relational organization, only for the number of active columns less or equal to 7. For the CPU bound case, the initial projection start-up cost for the relational organization had a significant impact on the difference between the cost functions. This resulted in relatively better performance of the vector building block and vector matrix schemes and, in fact, VM performed better for $p \leq 15$.

Another important observation is that with the Singular Value Factorization we observed the cumulative effect of the implementations at the three abstraction levels. This was due to the implementation of the QR decomposition step, the U' U step, and the matrix product step all at the same abstraction level. Table 7.2 summarizes the threshold values of p for X'X, QR and SVF.

Algorithm	X'X	QR	SVF
Vector Building Block	10	10	5
Vector-Matrix	13	41	7
Direct Implementation	51	all p	all p

Table 7.2

The threshold values are the maximum number of active columns for which the algorithms with the transposed organization outperform the direct implementations with the relational organization. These values are for the I/O bound total execution times. Notice the optimality of the

direct implementations with the transposed organization, especially for the Singular Value Factorization and the QR decomposition. The cumulative effect of implementations at levels one (VBB) and two (VM) is significant. One reason for the poor performance is the large number of page transfers in the matrix product steps of these algorithms. For VBB, the matrix product step involved approximately $3 \cdot p^2 \cdot N$ page transfers. VM the matrix product step involved approximately $p^2 \cdot N$ page transfers. The direct implementations, on the other hand, involved only $2 \cdot p \cdot N$ page transfers. Another reason for the poor performance of VBB and VM is that with the relational secondary storage organization we were able to suggest an efficient algorithm which eliminated the I/O overhead for the construction of U' U.

Finally, we proposed a multiprocessor architecture for the matrix product step. In Chapters 5 and 6 we had proposed parallel architectures for X'X and the QR decomposition, respectively. These two operations constitute the first two main steps of the direct algorithms with the transposed secondary storage organization. The multiprocessor architecture for the matrix product step was proposed to have a complete set of architectures for all the major computational steps of the Singular Value Factorization. The multiprocessor architecture for the matrix product step is a pipeline of p stages, with p processors at each stage. The processors at one stage are interconnected to the processors at the next stage through a cyclic shift.

CHAPTER 8

CONCLUSIONS

The data management issues of statistical databases have received considerable attention in recent years. Although several interesting solutions to the characteristic problems of large statistical databases have emerged, the data management issues of the computational methods have been generally ignored. This dissertation endorsed an integrated approach to statistical databases. With an integrated system, the database management supports both the data intensive and computational queries. Since issues regarding data retrieval, sampling, and aggregation type queries have already been carefully examined, we chose to concentrate on the data management issues of the computational queries.

The main emphasis in this dissertation has been the comparative performance evaluation of three important computational methods: X' X, the QR decomposition, and the Singular Value Factorization. The alternative algorithms were with respect to two basic types of secondary storage organizations (relational and transposed) as well as three abstraction levels corresponding to vector building block, vector-matrix, and direct implementations. The basic contributions of our research can be summarized as follows:

(1) we developed closed form equations, for the I/O costs and the total execution times in terms of the system and data parameters. The variable parameters were the number of active columns, main storage size, and time per floating point operation.

(2) we analyzed the effects of the transposed and relational secondary storage organizations on the total I/O and the total execution time.

(3) we performed a comparative evaluation of alternative algorithms for vector building block, vector-matrix and direct implementations. The algorithms explicitly specify the buffer management strategy that provides optimal performance.

(4) we proposed special purpose multiprocessor architectures for each of X' X, QR and SVF.

The performance evaluation assumed a simple and general system architecture. The generality of the system architecture enables the designers of the database management system to modify the parameter values to suit their particular architectures. This, in turn, would facilitate choosing between the transposed and relational secondary storage organizations and also the abstraction level of the algorithms to be supported.

Below we present three main conclusions which we deduced from our extensive performance evaluation of X' X, the QR decomposition, and the Singular Value Factorization.

8.1. Building Blocks For An Integrated System

An important performance issue in the development of a special purpose database management system, is the specification of the basic data structures and algorithms which implement the set of supported operations. For the computational methods, we investigated alternative implementations at three abstraction levels: vector building blocks, vector-matrix, and direct implementations. The vector building blocks and vector-matrix implementation strategies permit the implementation of all the computational methods through a small, well-defined set of vector and vector-matrix operations. Moreover, alternative and novel computational methods will, most probably, be supportable through existing vector or vector-matrix building blocks. Unfortunately, however, the vector building-block and vector-matrix implementations of the computational methods performed considerably poorer than the direct implementations. For example, with X' X and the matrix product step of SVF, vector building block and vector matrix strategies involved $O(p^2 \cdot N)$ page transfers. However, with the direct implementations, only $p \cdot N$ page transfers are needed for these operations. Therefore, an integrated system supporting the computational methods through vector buiding blocks and vector-matrix operations will perform considerably poorer than an integrated system which supports the computational methods

through direct implementations.

Therefore, the options to building blocks integrated statistical database systems, are either integrated systems where the computational methods are supported through direct implementations, or interface systems.

The former option implies an implementation per computational method. As we discussed in Chapter 2, the problem with this approach is that the set of statistical computational methods is continually expanding. Whenever, a new computational method is introduced, the database management system software must be extended to include a direct implementation of the new computational method.

Unfortunately the interface system approach is not likely to yield a satisfactory solution as most statistical and linear algebra packages implement the computational methods through vector building blocks.[1] Furthermore, these packages utilize the global buffer management strategy of the underlying operating system. An interesting research project would be to compare the performance of the buffer management algorithms for the three computational methods with implementations which use a global buffer management strategy.

8.2. Transposed and Relational Organizations

The second conclusion concerns the relative performance of the transposed and relational storage organizations. The direct implementations of each algorithm using the transposed secondary storage organization outperform the direct implementations using the relational organization, when the number of active columns is less than or equal to 50 in the case of $X'X$, and for all ranges of active columns in the case of the QR decomposition and the Singular Value

[1] for example LINPACK [DONG79] implements most of the matrix operations through the Inner Product, AXPY, and 2-Norm vector building blocks.

Factorization. In fact, in the case of the QR decomposition and the Singular Value Factorization, increasing the number of active columns also increased the difference between the cost functions[2] of the transposed and relational direct algorithms, in favor of the transposed organization. Therefore, our results suggest that for those statistical computational methods whose algorithms involve an iterative decrease in the number of currently active columns, the underlying storage structure must be the fully transposed secondary storage organization.

Although the transposed and the relational organizations are the main secondary storage layouts commonly utilized in most existing statistical database management systems,[3] it might be worthwhile to investigate other alternative storage structures. As a simple example, assume that the users of the system (i.e. the analysts) are capable of predetermining the attribute subsets which they will analyze. If it is feasible to convey this suggestive information to the database management system, then it becomes possible to construct a clustering of the attribute sets and store the data sets as partially transposed files. This will reduce the total I/O time for X' X type operations, as well as the overhead of the initial projection step for those operations that produce temporary files (e.g. the QR decomposition).

There are more challenging research endevours which could be pursued in order to efficiently structure and access the statistical databases. The first step here is to characterize the data structures and the access patterns utilized by a given set of operations. The second step is to incorporate identified structures into the database management system. The third step is to implement efficient and direct algorithms for each of the operations.

---

[2] both I/O and total execution times

[3] for example SEEDIS [McCA83] uses variable length records, whereas RAPID [TURN79] uses fully transposed files

An interesting research effort along these lines is currently underway at LBL, University of California [SHOS84]. By interviewing various scientists, the database research group was able to categorize the access patterns and the topology of scientific databases. Both experimental (raw) and associated data (derived or meta-data) were categorized. Although this research investigated scientific databases, the approach of determining partition structures could also be applied to statistical operations.

8.3. Statistical Database Machine Architectures

The dissertation proposed a multiprocessor architecture for X' X (the ZETA networks), a multiprocessor/multidisk architecture for the look-ahead scheme of the QR decomposition, and a multiprocessor architecture for the matrix product step of the Singular Value Factorization. Each architecture was designed with respect to a direct algorithm for the corresponding computational method. For these pipelined architectures, during the execution of an operation, and at a given instant, there are three basic types of data-blocks:

(1) input blocks from the input I/O subsystem

(2) blocks being processed by the multiprocessor

(3) updated pages written to the output I/O subsystem

The system continuously accesses the input blocks from the input I/O subsystem, and pipes the blocks through the multiprocessor. The multiprocessor updates the blocks, and pipes the updated pages to the output I/O subsystem. It is our contention that such pipelined architectures are ideal for the concurrent execution of the computational methods. Unfortunately, since the access pattern of each operations is distinct, the multiprocessor connection topology was different in each architecture.

An extension of our research is the development of a general purpose statistical database machine architecture which supports both data intensive and computational statistical operations.[4] It would be interesting to compare the performance of a general purpose architecture with the performance of our special purpose multiprocessor architectures which were designed for optimal algorithms of the computational methods.

Finally, two other avenues of future research appear fruitful. The first, is the development of a comprehensive list of computational methods.

The computational methods analyzed in this dissertation were those of multiple linear regression. Although the QR decomposition and the Singular Value Factorization are used in other important statistical techniques (e.g. canonical correlation and principal component analysis), there are a number of other statistical methods, with corresponding computational methods, whose data management problems need to be explored and analyzed. The goal here is to provide the analyst with a comprehensive list of alternative statistical techniques, which have high performance and which also have the capability of manipulating large data sets. To this end we need to pursue the development of a list of the most frequently used statistical tools and, correspondingly, the most useful computational methods.

To date, research has mostly been "algorithm" directed. The underlying system architecture for each of the performance evaluations was simple and the emphasis was on alternative algorithms for this general type of architecture. On the other hand, the proposed multiprocessor architectures were determined by the direct algorithms of the computational methods. Therefore, another avenue of research would be to propose secondary storage organizations and paral-

---

[4] The statistical database machine architectures which have been proposed thus far (e.g. [FARS83, HAWT82]) consider only data intensive statistical operations.

lel algorithms for existing loosely coupled multiprocessor architectures.[5] In other words, it could be worthwhile to investigate the performance issues of distributed statistical databases. A novel area here is the analysis of distributed algorithms for a number of computational operations (e.g. X'X, QR, SVF etc.), over large and distributed data sets.

[5] An example of such a multiprocessor is the CRYSTAL multicomputer [COOK83] developed at the Computer Science Department of the University of Wisconsin - Madison.

APPENDIX

In Chapters 5, 6, and 7 we presented several buffer management algorithms for vector building block, vector-matrix, and direct implementations of X'X, the QR decomposition and the Singular Value Factorization. Below we briefly describe the syntax of the language as well as the set of functions which were utilized in the algorithms.

(1) Block Reference: $X_i^{B.k}$ stands for the k^{th} B - page block of columns X_i. If B is missing the block size is 1.

(2) Concurrent Execution: the statement lists which are preceded by a "$", execute concurrently. The general form is:

$ <statement list>

$ <statement list>

There could be two or three concurrent statement lists in an algorithm. With two concurrent statement lists one statement list is executed by the I/O subsystem and the other by the processing subsystem. With three concurrent statement lists the I/O subsystems consists of an input and an output subsystem, and, therefore, the three concurrent statement lists are executed by the processing subsystem and each of the input as well as output subsystems.

(3) Read ($X_{il}^{Bl.kl} \cdots X_{ij}^{Bj.kj}$): means the indicated blocks are read from the same I/O subsystem.

(4) Write ($X_{il}^{Bl.kl} \cdots X_{ij}^{Bj.kj}$): means the indicated blocks are written to the same I/O

subsystem.

(5) For Loops: For a := b to c <statement list>: b is the initial value of the loop index a, ci is the final value, and the increment is 1.

(6) If Statements: If <condition> <statement list> [else <statement list>].

(7) Computational Statements: there are two basic types of computational statements- Inner Products or AXPY. For Inner Products, the statement:

$$a := a + X_i^{B.k} \cdot X_j^{B.k}$$

indicates the current value of the scalar a is the old value of a plus the inner product of the vectors which are the k^{th} B - page blocks of columns X_i and X_j.

For AXPY, the statement:

$$Y := Y + a \cdot X$$

indicates the i^{th} element of the transformed Y is equal to the i^{th} element of Y plus a (scalar) times the i^{th} element of X.

There were two other computational steps which were utilized in the Singular Value Factorization.

(a) Matrix Times Vector:

$$V := [\, V_1 \cdots V_p \,] \cdot Y$$

vector V is the product of the matrix whose k^{th} columns is V_k and the vector Y. This step is used in the VM implementation of the SVF.

(b) Matrix Product:

$$W^{B.k} := [\, U_i^{B.k} \cdots U_p^{B.k} \,] \cdot M$$

the k^{th} B - block of matrix W is equal to the product of the matrix whose j^{th} column is $U_j^{B.k}$ and the matrix M. This step is used in the DIR 2 implementation of the SVF.

Next we briefly explain the functions and the procedures which were used in the algorithms.

(8) Free ($X_i^{b.k}$): when this procedure is invoked, the B pages of primary store currently occupied by $X_i^{B.k}$ become available.

(9) Find (B): this boolean function does not return until B pages of primary store are allocated to the calling process.

(10) Set (b): sets the boolean variable b to TRUE.

(11) Alloc (X^B, Y): allocates the B pages of X^B to Y.

(12) Check (b): this procedure checks for the value of b and it returns when b is et to TRUE.

(13) (i, j) := FindPair (A, D) : this function accesses the one dimensional boolean array A, and the two dimensional boolean array D and does not return until it finds a pair (i, j) such that A [i] is TRUE and D [i, j] is FALSE. It returns the pair (i, j). This function was invoked in ST 1 and ST 2 algorithms of X' X.

(14) clear(A):with A a boolean array this procedure resets all the entries in A to FALSE.

BIBLIOGRAPHY

[BAKE76] Baker, M., "Users's Guide to the Berkley Transposed File Statistical System: PICKLE," Technical Report No. 1, 2nd ed., University of California, Berkleym Survey Reserach Center, 1976.

[BATO79] Batory, D. S., "On Searching Transposed Files," ACM Transactions on Database Systems, Vol. 4, No. 4, December, 1979.

[BELL83] Bell, J., "Data Structures For Scientific Simulation Programs'" Proceedings of the Second LBL Workshop on Statistical Database Managment, September, 1983.

[BIRK70] Birkhoff, G., and Thomas, B. C., Modern Applied Algebra, McGraw-Hill, Inc., 1970.

[BORA82] Boral, H., DeWitt, D. J., and Bates, D. M., "Framework for Research in Database Management for Statistical Analysis," University of Wisconsin-Madison, Computer Sciences Tech. Report #465, February, 1982.

[BORA83] Boral, H., and Dewitt, D. J., "Database Machines: An Idead Whose time Has Passed? A Critique of The Future of Database machines," University of Wisconsin-Madison, Computer Sciences Technical Report #504, July, 1983.

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

[CHAN81] Chan, P., and Shoshani, A., "SUBJECT: A Directory Driven System for Large Statistical Databases," Proceedings of the First LBL Workshop on Statistical Database Managment, December, 1981.

[CHAN82] Chan, T. F., "An Improved Algorithm for Computing the Singular Value Decomposition," ACM Transactions on Mathemetical Software 8, 1, 1982.

[CHUA81] Chuan, W., and Tse-Yun, F., "The Universality of the Shuffle-Exchange Network" IEEE Transactions on Computers, vo. c-30, No. 5, pp.324-331.

[CODD70] Codd, E. F., "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM, Vol. 13, No. 6, June, 1970.

[COOK83] Cook, R., Finkel, R., Gerber, B., DeWitt, D. J., and Landweber, L., "The Crystal Nuggestmaster," University of Wisconsin-Madison, Computer Sciences Dept. Tech. report #500, April, 1983.

[CUPP81] Cuppen, J. J. M., "The Singular Value Decomposition in Product Form," Mathematisch Institut, Amesterdam, pp. 81-106, April, 1981.

[DENN83] Denning, D., Nicholson, W., Sande, G., and Shoshani, A., "Research Topics in Statistical Database Management," Proceedings of the Second International Workshop on Statistical Database Management, September, 1983.

[DONG79] Dongarra, J., Moler, C., Bunch, J., and Stewart, G., 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, 1983.

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

[FISH82] Fishburn, J. P., and Finkel, R. A., "Quotient Networks," IEEE Transactions on Computers, Vol. c-31, No. 4, April 1982.

[FARS82] Farsi, H., and Tartar, J., "A Relational Database Machine for Efficient Processing of Statistical Queries," Proceedings of the Second LBL Workshop on Statistical Database Managment, September, 1983.

[FLYN66] Flynn, J., "Very High-Speed Computing Systems," Proceedings of IEEE, 1966.

[GEY 83] Gey, F., McCarthy, L. J., Merill, D., and Holmes, H., "Computer-Independent Data Compression for Large Statistical Databases," Proceedings of the Second LBL Workshop on Statistical Database Managment, September, 1983.

[GOLD83] Goldman, I. A., "Model For a Clinical Research Database," Proceedings of the Second LBL Workshop on Statistical Database Managment, September, 1983.

[GOLO67] Golomb, S. W., Shift-Register Sequences, Holden-Day, Inc., San Fransisco, 1967.

[GOLU73] Golub, G., and Styan, G., "Numerical Computations for Univariate Linear Models," Journal Statistical Computing, Vol. 2, 1973.

[HAWT82] Hawthorn, P., "Microprocessor Assisted Tuple Access, Decomposition and Assembvly for Statistical Database systems," Proceedingd of the 8th International Conference on VLDB, September, 1982.

[HEXT82] Hext, G. R., "A Comparison of Types of Database System Used in Statistical Work," COMPSTAT 82, 1982.

[HOFF76] Hoffer, A. J., "An Integer Programming Formulation of Computer Data Base Design Problems," Information Sciences 11, pp. 29-48, 1976.

[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.

[KENN80] Kennedy, W., and Gentle, J., Statistical Computing, Marcel Dekker, Inc., 1980.

[KLUG81] Klug, A., "Abe: A Qery Language by example," Proceedings of the First LBL Workshop on Statistical Database Managment, December, 1981.

[KNUT81] Knuth, D. E., The Art of Computer Programming, Volume 2, Seminumerical Algorithms Addison-Wesley, 1981.

[LANG76] Lang, T., "Interconnections Between Processors and Memory Modules Using the Shuffle-Exchange Network," IEE Transactions of Computers, vol. c-25, No. 5, May, 1976.

[LAWR75] Lawrie, D. H., "Access and Alignment of Data in an Array Processor," IEEE Transactions on Computers, vol. c-24, No. 12, December, 1975.

[LENF78] Lenfant, J., "Parallel Permutations of Data: A Benes Network Control Algorithm for Frequently Used Permutations," IEEE Transactions on Computers, vol. c-27, No. 7, July, 1978.

[LEV 81] Lev, G. L., Pippenger, N., and Valiant, L. G., "A Fast Parallel Algorithm for Routing in Permutation Networks," IEEE Transactions on Computers, vol. c-30, No. 2., February, 1981.

[MAND82] Mandel, J., "Use of Singular Value Decomposition In Regression Analysis," The American Statistician, Vol. 36, No. 1, February, 1982.

[McCA82] McCarthy, J. L., "The SEEDIS Project: A summary Overview," Lawrence Berkley Lab., LBL-14083, 1982.

[McCU83] McCullagh, P., Nelder, J. A., Generalized Linear Models, New York, Chapman and Hall, 1983.

[MEYE69] Meyers, E. D. Jr., "Project IMPRESS: Time Sharing in the Social Sciences," AFIPS Conference Proceedings of the Spring Joint Computer Conference, 1969.

[NASS81] Nassimi, D., and Sahni, S., "A Self-Routing Benes Network and Parallel Permutation Algorithms," IEEE Transactions on Computers, vol. c-30, No. 5, May, 1981.

[PARL80] Parlett, B. N., The Symmetric Eigenvalue Problem, Prentice Hall, 1980.

[PARK80] Parker, S. D., "Notes on Shuffle/Exchange-Type Switching Networks," IEEE Transactions on Computers, vol. c-29, No. 3, March 1980.

[PERR81] Perry, J. R., "Secondary Storage Methods for Solving Symmetric, Positive Definite, Banded Linear Systems," Yale University Computer Science Research Report #201, 1981.

[PETE72] Peterson, W. W., and Weldon, E. J. Jr., Error-Correcting Codes, The MIT Press,

1972.

[RAFA83] Rafanelli, M., and Ricci, F. L., "Proposal of a Logical Model For Statistical Database," Proceedings of the Second LBL Workshop on Statistical Database Managment, September, 1983.

[ROWE83] Rowe, N. C., "Top-Down Statistical Estimation on a Database," Proceedings of SIGMOD 83, 1983.

[RYAN76] Ryan, T. A., Joiner, B. L., and Ryan, B. F., MINITAB Student Handbook, Duxbury press, Massachusetts, 1976.

[SEBER77] Seber, G., Linear Regression Analysis, John Wiley & Sons, 1977.

[SHOS82] Shoshani, A., "Statistical Database: Characteristics, Problems, and Some Solutions," Proceeding of the Eight International Conference on VLDB, 1982.

[SHOS84] Shoshani, A., Olken, F., and Wong, H. K. T., "Characteristics of Scientific Databases," LBL-17582

[STAN83] Stanley, S. Y. W., Navathe, S. B., and Batory, D. S., "Logical and Physical Modeling of Statistical/Scientific Databases," Proceedings of the Second International Workshop on Statistical Database Management, 1983.

[STEW73] Stewart, G., Introduction to Matrix Computations, Academic Press, 1973.

[STON71] Stone, H. S., "Parallel processing with the perfect shuffle," IEEE Trans. Comput., C-20, Feb 1971.

[THOM83] Thomas, J. J., and David, H. L., "ALDS Projec: Motivation, Statistical Databse Management Issues, Perspectives, and Directions," Proceedings of the Second International Workshop on Statistical Database Management, 1983, pp. 82-88.

[TUKE77] Tukey, J. W., Exploratory Data Analysis, Addison-Wesley, Massachusetts, 1977.

[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.

[WILL71] Wilkinson, J. H., and Reinsch, C., Linear Algebra, Springer-Verlag, 1971.