PERFORMANCE EVALUATION OF THE

STATISTICAL AGGREGATION BY CATEGORIZATION IN THE SM3 SYSTEM

C. K. Baru

S. Y. W. Su

Database Systems Research and Development Center

University of Florida, Gainesville, FL. 32611

Abstract

To perform a statistical aggregation operation over a large file often requires that the records of the file be divided into categories based on the values of the attribute(s) over which some statistical computation is to be performed. It is rather inefficient to perform the necessary data transfer, categorization and statistical computation using a single processor. Parallel algorithms designed for multiprocessor systems have been proposed and their performance improvement over the conventional systems has been demonstrated. It is shown in this paper that three to four times performance improvement can be further gained by using a dynamically partitionable multicomputer system with switchable main memory modules (SM3).

1. INTRODUCTION

In the past decade, we have witnessed a tremendous progress in database management technology. Many commercial database management systems (DBMSs) have been made available for large corporations and enterprises. However, these DBMSs are mainly designed for business-oriented applications in which data are assumed to be formatted and can be represented by a few primitive data types such as integer, real, character, string, and bit. They are not particularly suited for statistical applications which have many characteristics that are different from business applications. These differences are well-documented in [HAM78, CHA81, SHO82, BOR82]. Most notably, statistical applications regularly deal with complex data types such as matrices, time series, set, vector, variable length text strings, data, etc., which are generally not recognized and supported by the existing DBMSs. The operations required in statistical applications are also quite different from those of business data processing. For example, statistical aggregations (e.g., aggregation of petroleum products by countries, states/provinces, and petroleum types) and disaggregations, modification of data to produce the needed periodicity for statistical analysis of time series, and conversion of data to suit statistical packages are a few needed operations, in addition to the regular database management operations such as retrieval, update, insertion and deletion. Due to the differences in both data representation and operations, the need for new techniques for the modeling, design and implementation of statistical databases and statistical database management systems have been recognized by the research community [STW81, STW83]. Works in data modeling [CHA81, JOH81, SU83a], physical design [BAT82, TUR79, EGG81], systems for statistical processing [SAS79, LEM80, BEC78, HID81], security in statistical databases [CHI81, DEN80, YU77] and language interfaces [BRO82, WON82] are but a few examples of the current efforts.

This paper addresses the problem of statistical aggregations and their implementation. Two types of statistical aggregations are commonly used in statistical applications. The first type is called scalar aggregation which takes a set of values as input and computes a single value such as the computation of Sum, Count, Average, Maximum, Minimum, or Median. The second type shall be called "statistical aggregation by categorization" which aggregates records of a file into subsets based on the various value combinations of some attributes and applies scalar aggregation functions over these subsets to obtain summary data for the subsets. For example, the aggregation of petroleum data records by countries, states/provinces and petroleum types followed by the computation of total production for each category is an example of statistical aggregation by categorization. Both aggregation types require that every record of a file be processed to obtain the result. They are very time-consuming operations, especially when the involved data files are large. Efficiency in statistical aggregations is definitely a problem in systems used for statistical applications.

A possible approach to this problem is to design better algorithms and hardware architectures for the efficient processing of statistical aggregations. Several architectures and algorithms have been studied. They include the multiprocessor cache system [DEW81], the multiprocessor hashing system [JOH82]

and the common-bus network system [SU82]. Based on the analyses of algorithms presented in these works, the operation of moving data among the processors accounts for a large portion of the total execution time required for statistical aggregations. Considerable performance improvement can be obtained if data transfer time (typically through a communication bus) can be drastically reduced. This idea motivates our present work in seeking a novel architecture to perform statistical aggregation efficiently.

This paper presents an architecture of a multicomputer system called SM3 which contains a number of main memory modules that are shared by the processors. In this system, the usual data transmission through a communication bus is replaced by switching of main memory modules between processors, thus reducing the usual network transmission time to memory switching time. The algorithms described in [DEW81] and [JOH82] are adapted to this system. The adapted algorithms are evaluated analytically and compared with the results of the original algorithms and architectures. It is shown that 1) it is possible to achieve a time reduction 3-4 times less than the original algorithms for statistical aggregations by categorization and 2) the two adapted algorithms represent two extremes, each with some advantages over the other and there is a crossover point at which one becomes less appealing than the other.

Section 2 of this paper defines the problem of aggregation by categorization and reviews the algorithms and architectures proposed in [DEW81, JOH82]. Section 3 describes the architecture of the Switchable Main Memory Module System (SM3) and two algorithms which are the adaptations of DeWitt's and Johnson's algorithms. Section 4 presents an analysis of the adapted algorithms and a comparison of their execution times with the results obtained by DeWitt, et al., and Johnson.

2. THE AGGREGATION PROBLEM AND PROPOSED SOLUTIONS

In this section, we define the statistical aggregation problem addressed in this paper and review two existing architectures and their associated algorithms.

2.1 Statistical Aggregation by Categorization

Given a set of record occurrences of a record type defined by a set of attributes, a statistical aggregation by categorization is an operation which partitions the set of record occurrences into subsets, each of which contains record occurrences having the same value over a single or multiple attribute(s). The attribute(s) by which the subsets are formed is/are called category attribute(s). The subsets are henceforth called categories. The data of the record occurrences in each category are subject to some statistical summarizations to produce summary values for some summary attributes that describe or characterize the category. For example, the set of record occurrences of the following record type can be partitioned into categories by the values of the category attributes, STATE, COUNTY, RACE, and SEX.

POPULATION

STATE	COUNTY	RACE	SEX	NAME	INCOME	AGE

Each category contains a subset of the original set of records whose values over the attributes STATE, COUNTY, RACE, and SEX are identical. A category is therefore an aggregation of records having the same values over some designated category attributes. One can apply some scalar aggregation functions such as AVERAGE and SUM over the attributes AGE and INCOME, respectively, to obtain values for the summary attributes AVG_AGE and TOTAL_INCOME for each category. The new set of record occurrences of the following record type is called the summary set [JOH81].

POPULATION_CATEGORY

STATE	COUNTY	RACE	SEX	AVG_AGE	TOTAL_INCOME

A statistical aggregation by categorization can be further applied on a summary set. For example, if the average ages and total incomes of populations categorized by state and county are of interest, further aggregation applied on POPULATION_CATEGORY would result in the following record type and its summary set:

POPULATION_BY_STATE_COUNTY

STATE	COUNTY	AVG_AGE	TOTAL_INCOME

The values of the summary attributes AVG_AGE and TOTAL_INCOME in the above record type are re-computed over the new categories. From the above discussion, it is clear that an aggregation operation can accept either a regular set of records or a summary set as input to carry out the summarization operations.

Having defined the aggregation operation, let us consider some problems involved in performing this operation in a conventional computer. First, we note that the size of a summary set (i.e., the number of record occurrences) depends on the number of distinct categories formed which can be rather large in some statistical applications. Second, all record occurrences have to be accessed and processed sequentially to compute the summary values. When the size of a file is large, this can be a very time-consuming operation. Third, as pointed out by Johnson [JOH81], some aggregations may involve a "one-pass" scan of the input set while others may involve a "multi-pass" scan. A one-pass aggregation allows the summary values to be computed as the data values in the records are scanned and processed. The input values can be discarded once they are used in the computation. The computation of AVG_AGE and TOTAL_INCOME is an example. The AGE and INCOME values are scanned and processed once only. A multi-pass aggregation, on the other hand, requires the set of data values to be stored and made accessible for computing the summary values. An example of this aggregation type is one involving the computation of a median as the summary data. It can be expected that the processing strategies and time required for these two types of aggregations can be quite different. Both aggregation types require an excessive amount of computation time, and the second one requires also a considerable amount of storage space.

The above observations and problems have motivated a couple of recent research efforts to investigate better architectures for supporting aggregation operations. The first architecture represents a general class of database machines called multi-processor-cache (MPC) [DEW81]. The second architecture called MPH uses hashing techniques in a multiprocessor system and is specifically designed for supporting aggregations [JOH82]. Since both architectures support the relational data model, we shall in the following review use the standard relational terminology. Thus, the record type mentioned above is henceforth called a relation and record occurrences are called tuples.

2.2 The MPC System

The MPC system represents a class of database machines which includes systems like DIRECT [DEW79], INFOPLEX [MAD79], and RDBM [HEL81]. The general configuration, shown in Figure 1, consists of a common secondary storage connected to a set of disk caches. These disk caches are connected via an inter-connection device to a set of processors. The inter-connection device allows one-to-one and one-to-all communication between the disk caches and the processors. The processors are small, general-purpose computers which carry out data management commands given by a host computer. For the purpose of describing the aggregation algorithm proposed for the architecture, a simple model used in [JOH82] and shown in Figure 2, should suffice.

In the first phase of the MPC algorithm, each of the N processors is loaded with a block of data from the disk. The data are assumed unsorted and, in the worst case, each processor can receive tuples from each of C distinct categories of the database. All processors perform aggregation in parallel by selecting the proper category, based on the category attribute values, and updating the proper summary attributes of the category. At the end of this phase each processor creates, at worst, a relation with C tuples. The primary memory is assumed to be large enough to hold this relation. For example, suppose a query stating, "What is the average income of females and males in each state?", was issued on the relation POPULATION_CATEGORY of section 2.1. This query requires an aggregation over category attribute values STATE and SEX in order to compute the summary values for AVG_AGE and TOTAL_INCOME. The attribute sex has two values, male and female, and the attribute STATE has 50 values. Hence, the number of categories is 100 and each processor needs to maintain a memory-resident table of size 100. As the tuples are read in, the values of AVG_AGE and TOTAL_INCOME are updated in this table, based on the values of STATE and SEX.

For the purpose of I/O time computation, it is important to compute the size of each relation in terms of the number of disk blocks that it occupies. We shall assume that C categories translate to G disk blocks, where G = (C x output tuple size in bytes)/ Disk block size in bytes. If there are N processors in the system then, at the end of the first phase the system contains N * G blocks of data. In the second phase, each processor writes back the data to disk and performs a merge/sort. This phase merges and aggregates the individual results, thus reducing the N runs of G blocks each into a single run of G blocks.

The MPC algorithm assumes that (i) each processor receives tuples from all the C categories of the data and, as a result, (ii) the local memory of each processor is large enough to hold all categories. If the number of categories is very large, G will also be very large. It has been observed that for a given hardware configuration, there exists a value X such that if G > X * R (where R is the size of the input relation in number of disk blocks), the performance of MPC is worse than a conventional system [JOH82]. The factor X is purely hardware dependent and in the particular configuration employed in [DEW81], X has the value 0.1. A major reason for this degradation is, of course, the worst case assumption employed above which states that each processor receives tuples from all the C categories. Since there are N processors, the global relation created at the end of phase 1 is of size N * G blocks, which may be larger than the source relation itself. On the other hand, the advantage of having N processors in the MPC system is that it ensures a high degree of parallelism and good processor utilization, since all N processors work together on the data during both the aggregation phase and the merge-sort phase.

2.3 The MPH System

The MPH database machine [JOH82] combines the multiple processor structure of MPC and the hardware hashing techniques of the Relational Database Machine (RDM) [SHA79]. The system can be represented as in Figure 3. The processors P_1, P_2. ... P_N are special-purpose, low-cost, medium performance machines with limited local, high-speed RAM. H is a high-speed device capable of performing selections and computing hash functions on tuples, as they come off the common secondary storage.

All tuples belonging to the same category are always mapped to the same processor. A processor can receive tuples belonging to more than one category if there are more categories than processors. Each of the N processors is assigned C/N categories and loaded with the appropriate aggregation operation. Using our previous query as an example, if there are 20 processors in the system, each processor would receive tuples from 100/20 = 5 categories. Tuples from the source relation (assumed unsorted) arrive at H which in turn directs them to the appropriate processor based on the category attribute values. At the end of this phase, each processor contains the final aggregated results for C/N categories. No merging of results is necessary since all the tuples belonging to a particular category are always mapped to the same processor. The final result is obtained by merely collecting the individual results from each processor.

The advantages of the MPH system are (i) since the MPH algorithm assumes that each processor has enough local memory to hold C/N categories if equivalent processors are used, the MPH system can tolerate N times more categories than MPC, and (ii) no final merging of results is required. On the other hand, if the distribution of tuples in the source relation is uneven or if one category contains an unusually large number of tuples, then the MPH system would suffer due to low processor utilization. A single processor may become the bottleneck due to overloading.

A performance analysis of MPH was carried out by Johnson and Thompson [JOH82]. The analysis used disk rotation times and processing rates to determine the buffer sizes required at the processors in order to keep the idle-time low. Also, an effort was made to study the effect of uneven loading of processors by assuming a Gaussian distribution for the number of tuples per processor. Detailed analysis using timing equations was not carried out for the MPH system. Hence, our analysis of the MPH-like algorithm in the SM3 system will also help to provide a rough benchmark for the MPH algorithm itself.

In the next section, we give a brief description of the SM3 system and show how algorithms which are naturally applicable to the system are indeed variations of the MPC and MPH algorithms. Since both algorithms are equally easy to implement in the SM3 system, it is possible to select the "better" of the two algorithms for any given query.

3. THE SM3 SYSTEM AND ITS ALGORITHMS

3.1 The SM3 System

The Switchable Main Memory Modules (SM3) System (Figure 4) is a partitionable multicomputer system primarily designed for non-numeric processing. Each node of the system is a general-purpose computer with its own local, primary memory, secondary storage and CPU. The computers also possess some main memory modules which can be switched between each other, mainly for the purpose of data transfer. Data can be loaded by one processor, P_i, into a main memory module which can be switched to another processor, P_n, for data access. The network is connected together by two different buses - the Switchable Memory Bus (SMB) and the Cluster Control Bus (CCB). The Switchable Memory Bus allows communication with the Control Computer, CC, which can transfer data to and from the Switchable Memories (SM's) via this bus. The Cluster Control Bus is connected via switches S1, S2, ... Si, which are controlled directly by CC. The CC is primarily responsible for compiling global queries, forming clusters by setting/resetting the Si switches, and collecting the results of global queries whenever they are required at CC.

The construction of CCB is identical to that of the bus used in the MICRONET system [SU78, NIC80, SU83b]. It contains address lines, data lines, and "global control lines" which are used for synchronization and establishment of a high-level protocol in the network. The control lines are wire-ANDed at each node and are, hence, called global-AND lines. They are used extensively for synchronizing the various phases of a given algorithm and contribute greatly towards reduction of interrupt and message transfer times.

As mentioned above, the CC can dynamically partition the system into independent clusters. One of the cluster processors is designated by the CC as the Cluster Control Processor (CCP). The CCP can manipulate all the switches inside a cluster except the Si's. It also has the capability of assigning any other processor of the cluster as a CCP, thereby losing its own privileged position. Note that in the extreme case when the whole network forms a single cluster, the CC takes on the responsibilities of a CCP.

The Switchable Memory Switch (SMS), controllable both by CCP and CC, allows the Switchable Main Memory module of a processor to be in one of three states - local to the processor (local mode), local to the cluster (cluster mode) or global to CC (global mode). The Cluster Control Processor Switch (CCPS), also controllable by CCP and CC, is used to assign CCP status to a processor. If processor Pi is to be the CCP, then CCPSi (normally open) is closed; thereby allowing Pi to perform one-to-one read/write and one-to-all write through the CCB.

Access to the switchable memories is regulated and synchronized in order to avoid problems like illegal accesses, loss of data and deadlock. The synchronization of accesses to the switchable memories, the setting of the Switchable Memory Switch, the specification of the access mode (read or write) for accessing the switchable memory and other related operations are all performed via Status Words assigned to each processor. A detailed description of the operation of these status words is given in [FEI84]. At this point, we stress again that the SM3 system allows sharing of memory modules, partitioning of the multicomputer system and one-to-one and one-to-all communication within a cluster and with CC.

The data (source relations) in the SM3 system are spread over many processors and stored in their local secondary stores. For a given query, all processors that contain the required source relation are grouped into a cluster by CC. All these processors are loaded with the same command and process the relation in parallel. The global control lines are used to synchronize the processors at the end of each phase. If data needs to be moved around inside the cluster during processing, the switchable memories are used. The final results are transferred via the switchable memories to either a node processor or to the CC. For our calculation purposes, we shall assume that the results are always transferred to the CC. This assumption gives results compatible with those of [DEW81].

Preliminary analysis of the SM3 system was carried out for operations like SELECT and JOIN [SU84]. To enhance performance, two SM modules mapped to the same address space (not shown in Figure 4) are actually provided. They are independently controllable but operate in exactly the same manner. In the analysis to follow, it will be assumed that data between disk and main memory is transferred via dual, independently accessible I/O buffers.

3.2 Algorithm I

This algorithm is the adaptation of the one used in MPC. As in MPC, it is assumed that the number of categories C is small enough to be held (in a sorted sequence) in the primary memory of each processor. The algorithm starts with each of the N processors of the cluster reading a block of data into one of its two I/O buffers. When one buffer is full, it is switched to the CPU while the disk controller continues to fill the other buffer with the next data block.

Meanwhile, the processors scan the first block of data and, based on the category attribute values, update the summary value in a table which resides

in the switchable memory. This processing time is overlapped with the disk read-out time. This sequence of operations is carried out until the last block of data has been read. In determining the operation time, only the maximum of the CPU and I/O operations is considered except for the last block for which processing and I/O operations cannot be overlapped. The results of the entire operation are stored in the switchable memory module, unless an overflow occurs into the regular main memory. At the end of this phase, the CCP switches all switchable memory modules to itself (i.e., to cluster mode) and reads the data from each SM in turn, at the speed of regular main memory access. It forms the final result by merging and aggregating the individual results from each processor into its own memory-resident table. This second phase is repeated more than once if the switchable memory has overflowed. Finally, assuming that the results are required at CC, they are moved from CCP to CC via the dual shared memories. If the SM3 system operates as a single cluster then, since the CC is itself the CCP, this last step of moving data would not be necessary.

Unlike in MPC and MPH, the source relation in SM3 is distributed across many processors. Hence, along with CPU operations, the I/O operations are also performed in parallel. The second phase of MPC requires N separate runs of G blocks each to be written back to the disk in order to merge/sort them to form the final result. In SM3, since the switchable memory modules can be accessed in the cluster mode by the CCP, the partial results are read directly from the memory to form the final result without having to write data back to the disk.

3.3 Algorithm II

Algorithm II is akin to the MPH algorithm where each processor is assigned a fixed sub-set of the C categories. At query compilation time, CC can assign categories to each processor such that the C categories are divided among the N processors. The algorithm starts with each processor transferring its first block of data from disk to the I/O buffer. At this point, only CCP enters into the broadcast (one-to-all) mode of communication and copies the data from its I/O buffer to the shared memories of all processors, including itself. The processors then proceed to form aggregates as before, in their local memories. During this time, the next data block is being read from the CCP's disk into its I/O buffer. The CPU and I/O operations are overlapped at the CCP and the I/O operation stops when the last block of data is read. At this point, the next processor is given CCP status (by the current CCP) and the above sequence repeats itself. In effect, the whole relation is scanned block-by-block and presented to each of the N processors which form aggregates by picking tuples relevant to them. Finally, each processor contains a non-intersecting subset from the set of all C categories, which need to be combined at the node where the output is required in order to form the final result.

The major advantage of this algorithm is that each processor needs to accommodate only C/N categories. Two major concerns in MPH were (i) the overloading of a processor due to an unusually large number of tuples mapping to a single category, and (ii) improper utilization of processors due to uneven distribution of tuples in the source relation. These problems could be solved in the SM3 system by minor alterations in the algorithm and better planning at query compilation time. For example, if a particular category has too many tuples, then it can be assigned to more than one processor. In this case, the final phase requires some aggregation to be performed along with merging as in Algorithm I.

From the above discussion, it is clear that the SM3 system lends itself to both types of algorithms. It is an easy matter to switch from one algorithm to the other. Algorithm I would be used in instances where the number of categories is not very large and the source relation is fairly equally distributed among the network nodes. Under such conditions, Algorithm I offers a great amount of parallelism and always performs much better than Algorithm II. A special case arises when the number of categories is large but the relation is naturally distributed across processors into non-intersecting subsets. For example, consider a case where an enterprise maintains information on its employees in the form of a distributed relation. Let us assume that the company has 20 locations where the data is locally maintained. Assuming that there are 4000 employees in the company, each location has 200 employees. If an aggregation is performed over the "Employee-Number" attribute, the total number of categories will be 4000 with each node containing 200 categories. Since the number of categories formed at each node is small, we can still use Algorithm I in this case.

Algorithm II is used when the number of categories is large. The loss of parallelism in this algorithm is compensated for by the last phase where, unlike in Algorithm I, no further aggregation is required since the categories in the different processors are non-intersecting. Instances may arise when one category is much larger in size than others. In such a case, as mentioned before, a category can be assigned to more than one processor and some aggregation can be performed in the last phase of the algorithm.

Hence, the criteria for deciding between the two algorithms can be specified quite clearly and one can switch from one algorithm to the other, thereby gaining the advantages of both without their disadvantages.

4. ANALYSIS AND EVALUATION

An analytical evaluation of the system, using timing equations, has been carried out in order to identify bottlenecks and to compare with other algorithms. A list of all parameters employed in this analysis is provided in Table I. The values chosen attempt to reflect a typical environment of operation for the SM3 system and they are also as close as possible to the values used in [DEW81], so that one may carry out a reasonable comparison. The absolute values would change depending on implementation.

4.1 Parameter Specifications

4.1.1 I/O Device Parameters

The secondary storage is assumed to be a stand-

ard, moving-head disk device. Specifically, parameter values of the IBM 3330 disk drive have been used. This device has 404 cylinders with 19 recording surfaces per cylinder. Each track on a surface holds 13,030 bytes. The size of a single track will be assumed as the unit of data transfer between secondary storage and primary memory and, also, between switchable memories. The time for a single rotation of this disk is 16.7 msec. The average direct-access time is 38.6 msec. and the track-to-track seek time is 10.1 msec.

4.1.2 CPU Parameters

The processing unit is modeled as a 1 MIP processor. Operation times are specified at a block level. The size of a block is equal to that of a track on a disk, i.e., 13,030 bytes. Assuming that the length of tuples in the relation is 100 bytes, we can accommodate approximately 130 tuples per block. Based on the fact that the CPU is a 1 MIP processor and using some results from [DEW81], we can calculate the times required for performing some general operations on a block of data. Scanning a block of data in order to perform simple operations like selection requires about 12 msec. This value represented by TSCAN, allows more than 90 instructions to be performed per tuple. More complex operations like those required for sorting or joining require a time of TPROC per block where TPROC has been approximated to 95 msec. The time required for moving a block of data within the primary memory is represented by TMOVE, for which a conservative estimate of 20 msec. is used.

4.1.3 SM3 System - Specific Parameters

Some basic operations which need to be accounted for in the SM3 system are query compilation and code generation, processor clustering, switching of shared memories and buffers, message transfer times, etc. Query compilation and code generation involve steps of parsing, optimization, and the generation of code. A detailed study of such operations was done in [CHAM81]. Based on the results of [CHAM81] and [DEW81], we can assign the time for code generation as TCODE = 200 msec. This figure would be higher if more extensive query optimization is carried out as suggested in section 3.3. Once the code has been generated, there is an overhead on the control computer CC to cluster the required processors. This step involves at least two table look-ups, one to determine which processors are involved in the operation and another to check if they are all free. Having determined that all the required processors are free for clustering, the proper switches need to be set/reset. The time allotted for clustering TCLUS is 50 msec.

It is assumed that all the switches in the network - Si's, SMSi's, CCPSi's, etc. - can be set/reset by reading a status register and issuing a hardware interrupt. The time required for this switching is TSWITCH = 2 msec. Also, the time required for switching the I/O buffer between the disk and CPU is TSWBUFF = 2 msec. A one-to-all broadcast from one switchable memory to the others requires that the sending processor ensure that all receiving switchable main memory (SM) modules are free. If so, the SM modules can be switched to the one-to-all broadcast mode. The time for this operation is represented by TBRDCST = 4 msec. The time taken to send simple interrupt messages between processors via the CCB is TMSG = 5 msec.

The track size of 13,030 bytes will be called a block and will be considered to be the unit of data transfer. The I/O buffers and the switchable main memory modules altogether comprise at least 52 Kbytes of special-purpose primary memory in each processor. In our calculations, the size of the global relation is taken to be r = 50,000 tuples (100 bytes each) and the size of a cluster is N = 19 processors (= number of heads per disk).

It is more meaningful to express the relation size in terms of blocks rather than tuples. The source relation of 50,000 tuples can be accommodated in R blocks, where R = 384.6. Since there are 19 processors per cluster, the number of blocks occupied by the relation in each processor is R/N = 384.6/19 = 20.24. This quantity is rounded up to RRn = 21. Similarly, since each category gives rise to a single tuple in the output relation, the number of categories C is an indication of the size of the output relation in tuples. The output relation can be expressed in number of blocks as follows: G = C/130. This value of G is again rounded off to GG. In the case of Algorithm I, each processor creates an output relation of size GG, whereas in Algorithm II each processor creates an output relation of size GG/N.

We shall now present the timing equations for each of the two algorithms discussed in sections 3.2 and 3.3.

4.2 Algorithm I

Initially, the CC needs to compile the query and generate code for it. This is followed by the clustering operation where CC forms a cluster of the relevant processors. This overhead time is

$$TOVHD = TCODE + TCLUS.$$

Next, each processor in the cluster accesses its first block and reads the block into the I/O buffer. The dual buffers are then switched, one to the processor and the other to the disk controller. The initial data access time is

$$TINIT = TACCESS + TREAD + TSWBUFF.$$

The reading of the remaining (RRn-1) blocks involves a track read time for each block plus a cylinder-to-cylinder seek time, i.e., a total of

$$(RRn-1) * TREAD + CEIL ((RRn/T)-1) * TSEEK$$

(where CEIL(X) is the smallest, non-negative integer greater than X.)

The time per block required to compute the aggregates is TPROC. Hence, the total processing time for (RRn-1) blocks is

$$(RRn-1) * TPROC.$$

The reading of the last (RRn-1) blocks is overlapped with the processing of the first (RRn-1) blocks, so we need to consider only the maximum of these two

times, i.e.:

MAX{((RRn-1)*TREAD + CEIL((RRn/T)-1) * TSEEK), ((RRn-1) * TPROC)}, RRn ≥ 1.

Finally, we have to account for the time required for switching the buffers from the CPU to the disk controller (and vice-versa) at the end of the processing/reading of each block. The time here is

(RRn-1) * TSWBUFF.

Therefore, the total elapsed time in this stage is

TEXEC = MAX{((RRn-1) * TREAD + CEIL((RRn/T)-1) * TSEEK), ((RRn-1) * TPROC)} + (RRn-1) * TSWBUFF.

For the very last block the buffer switching time and the processing time are non-overlapped operations. Hence, they contribute to the overall time:

TFIN = TSWBUFF + TPROC.

The final phase requires the CCP to switch all switchable memories to itself and then perform a global aggregation. (Note that the synchronization of the final state is achieved via the global control lines. Each processor sets its local control line when it reaches the end of its operation. Since the global line is a logical AND of each local line, the setting of the global line implies that all processors have completed their respective operations. Almost no time is lost in this synchronization stage.) The CCP performs global aggregation essentially by polling each SM in turn. The time required is TSWBUFF + (N-1) * TPROC. If the result relation cannot be accommodated in a single switchable memory module, the CCP may have to poll each processor more than once. After processing data from a particular processor, the CCP has to step through (N-2) processors before returning to the original processor. During this time, each processor can fill its SM with the next block of output. Hence, the total time required for forming global aggregates is

GG * TSWBUFF + G * (N-1) * TPROC.

Once the global aggregates have been computed at CCP, they need to be moved to CC for final output. The global results are again moved block by block via the switchable memories. The time for this operation is

GG * TSWBUFF + G * TMOVE.

Hence, the total time for performing global aggregation and collecting results at CC is

TTRAN = (GG * TSWBUFF + G * (N-1) * TPROC) + (GG * TSWBUFF + G * TMOVE).

Therefore, the total time involved in Algorithm I is the sum of its individual components:

TAGGCASE1 = TOVHD + TINIT + TEXEC + TFIN + TTRAN.

4.3 Algorithm II

As in Algorithm I, there is an overhead involved for compiling the query and forming a cluster of processors.

TOVHD = TCODE + TCLUS.

Each processor reads its first block of data from disk to I/O buffer and switches the two I/O buffers around.

TINIT = TACCESS + TREAD + TSWBUFF.

The reading and processing of the remaining (RRn-1) blocks of data are again overlapped operations. At this point, only the CCP goes ahead with reading and processing of its local relation. The time to read (RRn-1) blocks is

(RRn-1) * TREAD + CEIL((RRn/T)-1) * TSEEK.

Processing each block involves reading the block from the I/O buffer and broadcasting it to all processors, and performing aggregation on the block. The processing time required for the (RRn-1) blocks is

(RRn-1) * (TMOVE + TBRDCST + TPROC).

Only the maximum of processing and I/O times will eventually effect the operation time. The time required for switching buffers for each block of data is (RRn-1) * TSWBUFF. Finally, the last block of data needs to be processed and this processing cannot be overlapped with any other operation. The time required for processing the last block is

TMOVE + TBRDCST + TPROC.

After the CCP completes processing its local relation, it has to switch the next processor to CCP status so that the same sequence of steps may be repeated by it. The time required for switching a processor to CCP status is modeled as a buffer switching time plus an interrupt processing time, i.e., TSWBUFF + TMSG.

Since all the operations stated above have to be repeated for each of the N processors in the cluster, the total time contributed in this stage by all the processors is

TEXEC = N * <MAX[{(RRn-1) * TREAD + CEIL((RRN/T)-1) * TSEEK}, {(RRn-1) * (TMOVE + TBRDCST + TPROC)}] + {(RRn-1) * TSWBUFF} + {TMOVE + TBRDCST + TRPOC} + {TSWBUFF + TMSG}>.

In the final phase, all the results have to be transferred from each processor to the CCP, via the switchable memories. They may then be routed from there to the CC. The operations involved in this stage are similar to those in the final phase of Algorithm I, except that in this case each processor has G/N blocks of output rather than G blocks. Therefore, the final transfer time is

TTRAN = CEIL (G/N) * TSWBUFF + (G/N) * (N-1) * TPROC + GG * TSWBUFF + G * TMOVE.

The total time involved in Algorithm II is then

TAGGCASE2 = TOVHD + TINIT + TEXEC + TTRAN.

The individual results from each processor may have to be merged in order to maintain the global order; hence, a time of TPROC per block has been assumed. In all the above calculations, the time required for performing aggregation on one block (i.e., scan tuple, maintain memory-resident table in sorted order and update its values) is also equal to TPROC (95 msec.). If the aggregation operation to be performed is more complex, a higher value for TPROC may be chosen.

4.3 Evaluation

Results from the evaluation of Algorithms I and II are shown in Table II. They have been shown alongside the results from [DEW81] for the purpose of comparison. Similar analytical results were not available for the MPH System since no detailed formulas were provided in [JOH82]. Algorithm II should be a good representation of the MPH algorithm. Algorithm II should provide lower times than the MPH algorithm because of the advantages of the switchable memory and other architectural features of SM3. The equations in [DEW81] use the time involved in a particular operation as an indication of the amount of work done by the system. No assumptions regarding overlapping operations were made. In order to present an equivalent comparison, we re-calculated the timing for the MPC system assuming that I/O and CPU operations can be overlapped. The results obtained are listed under the column entitled "Modified MPC." Figure 5 gives a graphical representation of the performance of these algorithms.

Algorithm I performs much better than II for smaller values of C as expected since Algorithm II always scans the entire relation sequentially, thereby increasing the total time of the operation. As the number of categories increases, the parallel scan feature of Algorithm I is overshadowed by the time required to merge/aggregate results in the last phase. In essence, Algorithm I is sensitive to the output relation size, whereas Algorithm II is more sensitive to the input relation size. Hence, as the number of categories increases, the time required for Algorithm I exceeds that required for Algorithm II. Given the parameter values, it is feasible in the SM3 system to decide in advance which algorithm should be adopted for a given query.

The analysis carried out in [JOH82] approximated the distribution of the number of tuples/processor to a Gaussian distribution. Based on this, the authors have shown that in a large number of cases the degradation in performance due to uneven distribution is held to under 5%. The implementation of Algorithm II in the SM3 system has not violated any of the basic MPH assumptions. Hence, this analysis should hold true in the SM3 system also.

4.4 Further Enhancements

By illustrating the feasibility of the MPC and MPH type of algorithms for data aggregation, we have shown the flexibility of the SM3 system in adopting the most optimal algorithms in a given situation. The timing equations help us to identify further areas of improvement and possible optimization.

By estimating the number of possible tuples that will map to each category, a decision could be made whether to select Algorithm I, Algorithm II, or a combination of both. For example, suppose we expect the number of categories for a given query to be very large and at the same time we know that a large number of tuples will be mapped to one of the categories. In such a case, Algorithm II could be used to handle the large number of categories. Also, the category that has a large number of component tuples could be split across more than one processor. Let c be the category that is shared by p processors. The number of tuples that map to c is represented by $|c|$. Each processor that shares category c should receive $|c|/p$ tuples for this category. This can be easily accomplished by allowing each of the p processors to count the number of tuples of c that have already gone by, and by making use of the implicit ordering of the processors in a cluster. As soon as a processor receives $|c|/p$ tuples for the category, it stops collecting further tuples and the next processor in line starts collecting from tuple number $|c|/p + 1$ onwards. In this case, the total number of categories (for calculation purposes) will be $(C-1) + p$. This method is a combination of Algorithms I and II since up to the last phase the steps followed are as in Algorithm II. In the final phase, the global sorting has to be combined with some aggregation in order to arrive at the final result for category c. Hence, the extent of aggregation is not as much as in Algorithm I but is more than in II.

Further efficiency can be attained if the CC can maintain a comprehensive directory of all the files in the system, including information on number of tuples and boundary attribute values for each local segment of a global relation. Using this information, the CC can judiciously assign categories to processors so that the time involved in the final stage for merging/aggregating local results can be reduced considerably. Hence, it is possible to issue a highly optimized query to the system. Since the code generation time is a very small component of the overall operation time, the increased overhead required for optimizing the query has very little impact.

Since Algorithm II broadcasts all the blocks of the source relation, any processor can listen in and form aggregates for a given set of categories. This broadcast feature of the algorithm allows CC to assign an arbitrary number of processors to the cluster, thereby speeding up the operation.

The last phase of operation in both algorithms can be modified to use the broadcasting key sorting algorithm suggested in [SU82]. In Algorithm I, since a given category can be present in more than one processor, a global merge/sort is required in order to form the final result. In Algorithm II, the categories in each processor are distinct but they may not arrive in a global sequence at the CCP. Hence, a final global sort may be required in order to arrange all categories in their sorted sequence. The algorithm presented in [SU82] performs a global sort by using the Cluster Control Bus (CCB) rather than the switchable memories. The sort keys and tuples are transferred via this bus while the local memories are used to maintain "sort arrays." These arrays contain the top-most key from each processor. The processor with the lowest key (or set of keys), in the case of an ascending sort, is called the Successful Computer (SC). While

the SC transfers its key(s) and related tuple(s) to the Control Computer, the remaining processors perform computations to determine which of them will be the next SC. Hence, both the CCB and the memories are used simultaneously and, also, there is a high degree of parallelism in the algorithm. At the end of this stage, the globally sorted relation is available at the processor that has been designated as the Control Computer (CC). In the SM3 system, this corresponds to the CCP. It has been shown that this sorting algorithm compares favorably with several other sorting algorithms [SU83b]. The algorithm can be used to speed up the sorting operation required in Algorithms I and II.

REFERENCES

[BAT82] Batory, D.S., "Index Encoding: A Compression Technique for Large Statistical Databases," CIS Tech. Rep. 8182-9, Dept. of CIS, Univ. of Florida, 1982.

[BEC78] Becker, R.A. and Chambers, J.M., "Design and Implementation of the S System for Interactive Data Analysis," Proc. of IEEE COMPSAC 78, (1978), pp. 626-629.

[BOR82] Boral, H., DeWitt, D., and Bates, D., "A Framework for Research in Database Management for Statistical Analysis," Proc. of ACM/SIGMOD 1982 Conference, June 1982.

[BRO82] Brown, V.A., Navathe, S.B., and Su, S.Y.W., "Computer Data Types and a Data Manipulation Language for Scientific and Statistical Databases," CIS Tech. Report, Database Systems Research and Developemnt Center, Univ. of Florida, June 1982.

[CHA81] Chan, P. and Shoshani, A., "SUBJECT: A Directory Driven System for Organizing and Accessing Large Statistical Databases," Proc. of the 7th VLDB, 1981, pp. 553-563.

[CHAM81] Chamberlain, D.D., et al., "Support for Repetitive Transactions and ad hoc Queries in System R," ACM TODS, Vol. 6, No. 7, March 1981, pp. 70-94.

[CHI81] Chin, F.Y. and Ozsoyoglu, G., "Auditing and Inference Control in Statistical Databases," IEEE Trans. on Software Eng., SE-8, 3, May 1982, pp. 223-234.

[DEN80] Denning, D.E., "Secure Statistical Databases with Random Sample Queries," ACM TODS, Vol. 5, No. 3, Sept. 1980, pp. 291-315.

[DEW79] DeWitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Trans. on Computers, C-28(6), June 1979, pp. 395-406.

[DEW81] DeWitt, D.J. and Hawthorn, P.B., "A Performance Evaluation of Database Machine Architectures," Proc. of 7th VLDB, Sept. 1981.

[EGG81] Eggers, S., Olkin, F., and Shoshani, A., "A Compression Technique for Large Statistical Databases," Proc. of 7th VLDB, Sept. 1981, pp. 424-434.

[FEI84] Fei, T.H., Baru, C.K., and Su, S.Y.W., "SM3: A Dynamically Partitionable Multicomputer System with Switchable Main Memory Modules," IEEE International Conference on Data Engineering (COMPDEC), April 24-27, 1984, Los Angeles, CA.

[HAM78] Hampel, V.E. and Ries, D.R., "Requirements for the Design of a Scientific Data Base Management System," in a special report on Generalized Data Management Systems and Scientific Information, OECD Nuclear Energy Agency, Paris, 1978, pp. 111-131.

[HEL81] Hell, W., "RDBM - A Relational Database Machine: Architecture and Hardware Design," Proc. of 6th Workshop on Comp. Arch. for Non-Numeric Processing, June 1981.

[HID81] Hideto, I. and Kobayashi, Y., "Additional Facilities of a Conventional DBMS to Support Interactive Statistical Analysis," Proc. of 1st LBL Workshop on Statistical Database Management, Dec. 1981, pp. 25-36.

[JOH81] Johnson, R.R., "Modelling Summary Data," Proc. of ACM/SIGMOD Intl. Conf. on Management of Data, 1981, pp. 93-97.

[JOH82] Johnson, R.R. and Thompson, W.C., "A Database Machine Architecture for Performing Aggregations," Tech. Report UCRL - 87419, June 1982.

[LEM80] Lemon, J.S. and Knowles, J.S., "Data Management Facilities in Statistical Packages," COMPSTAT '80, pp. 108-114.

[MAD79] Madnick, S.E., "The INFOPLEX Database Computer: Concepts and Directions," Proc. IEEE Computer Conference, Feb. 1979.

[NIC80] Nickens, D.O., Genduso, T.B., and Su, S.Y. W., "The Architecture and Hardware Implementation of a Prototype MICRONET," Proc. of 5th Conf. on Local Computer Networks, Oct. 1980, pp. 56-64.

[SAS79] SAS Institute, Inc., SAS User's Guide, 1979 Edition, Raleigh, N.C., 1979.

[SHA79] Shaw, D.E., "A Relational Database Machine Architecture," Proc. of 5th Workshop on Comp. Arch. for Non-Numeric Processing, March 1980.

[SHO82] Shoshani, A., "Statistical Databases: Characteristics, Problems, and Some Solutions," Proc. of 8th VLDB, Mexico City, Sept. 1982, pp. 208-222.

[STW81] Proceedings of the 1st LBL Workshop on Statistical Database Management, Dec. 2-4, 1981.

[STW83] Proceedings of the 2nd Intl. Workshop on Statistical Database Management, Sept. 1983, Los Altos, CA.

[SU78] Su, S.Y.W., et al., "MICRONET - A Microcomputer Network System for Managing Distributed Relational Databases," Proc. of

4th VLDB, Berlin, W. Germany, Sept. 1978, pp. 288-298.

[SU82] Su, S.Y.W. and Mikkilineni, K.P. "Parallel Algorithms and their Implementation in MICRONET," Proc. of 8th VLDB, Mexico City, Sept. 1982.

[SU83a] Su, S.Y.W., "SAM*: A Semantic Association Model for Corporate and Scientific-Statistical Databases," Information Sciences, 29, 1983, pp. 151-199.

[SU83b] Su, S.Y.W., "A Microcomputer Network System for Distributed Relational Databases: Design, Implementation, and Analysis," Journal of Telecommunication Networks, 1983.

[SU83c] Su, S.Y.W. and Mikkilineni, K.P., "An Evaluation of Sorting Algorithms for Common-Bus Local Networks," manuscript, 1983.

[SU84] Su, S.Y.W. and Baru, C.K., "Dynamically Partitionable Multicomputers with Switchable Memories," to appear in the Journal of Parallel and Distributed Computing, 1984.

[TUR79] Turner, M.J., Hammond, R. and Cotton, P., "A DBMS for Large Statistical Databases," Proc. 5th VLDB, Oct. 1979, pp. 319-327.

[WON82] Wong, H.K.T. and Kuo, I., "GUIDE: Graphical User Interface for Database Exploration," Proc. of 8th VLDB, Mexico City, Sept. 1982, pp. 22-32.

[YU77] Yu, C.T. and Chin, F.Y., "A Study on the Protection of Statistical Databases," ACM/SIGMOD Intl. Conf. on Management of Data, 1977, pp. 169-181.

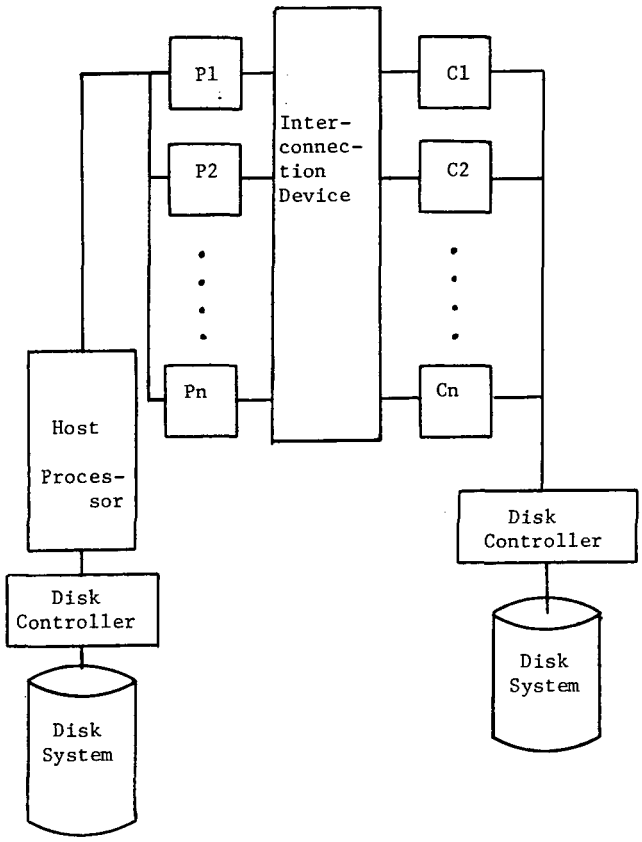

Fig. 1 MPC System Architecture

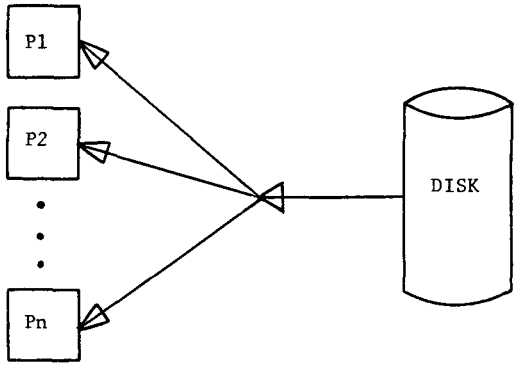

Fig. 2 The MPC System

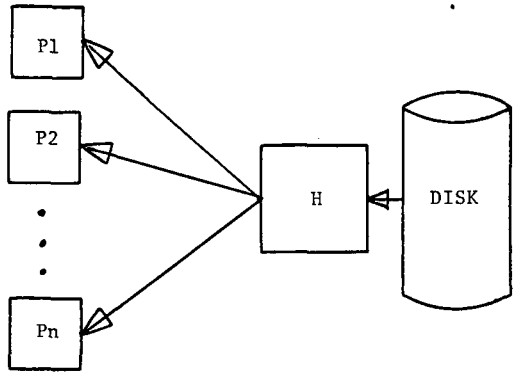

Fig. 3 MPH System

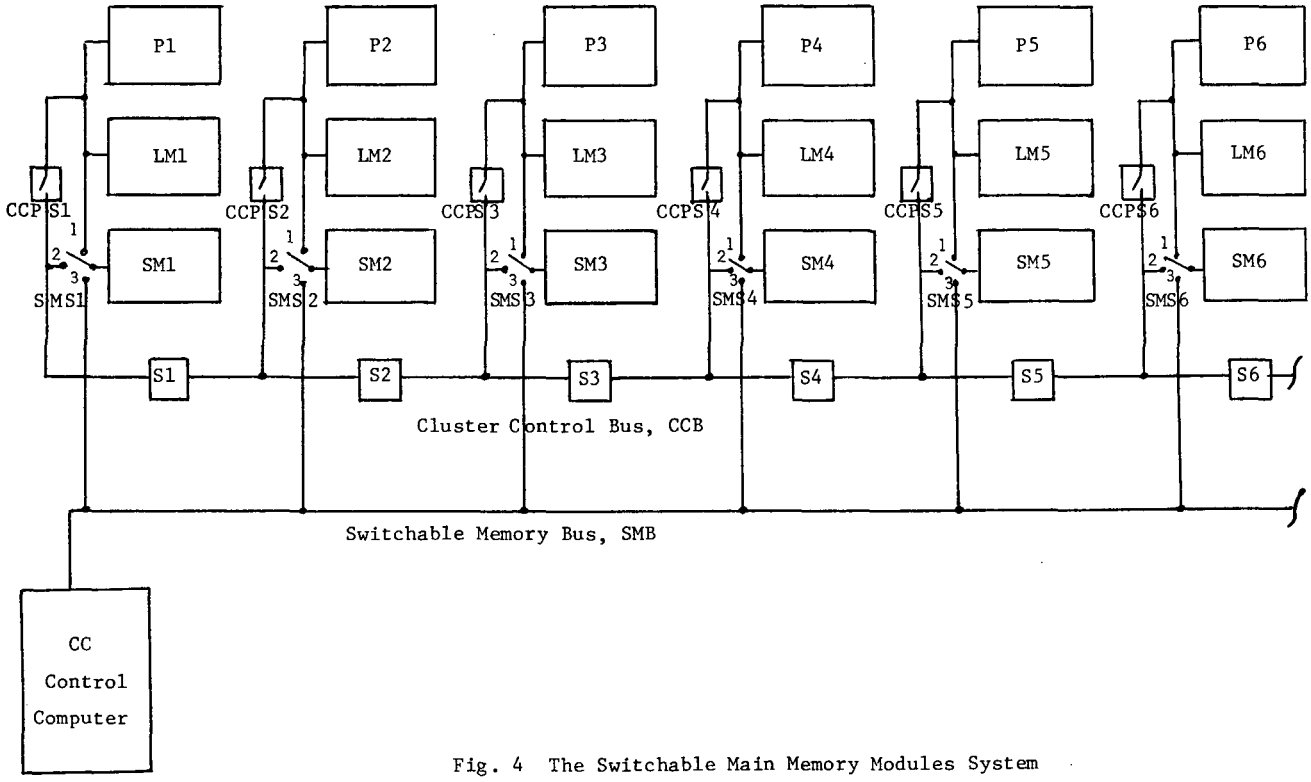

Fig. 4 The Switchable Main Memory Modules System

TABLE I

PARAMETERS AND VARIABLES USED

	DISK PARAMETERS	
BLOCK_SIZE	Size of a single track	13,030 bytes
T	Number of tracks/cylinder	19
TACCESS	Direct-access time	38.6 msec
TREAD	Track (block) read time	16.7 msec
TSEEK	Track seek time	10.1 msec
	CPU PARAMETERS	
TSCAN	Time to perform simple operations on a block of data	12 msec
TMOVE	Time to move a block of data within memory	20 msec
TPROC	Time to perform complex operation on a block of data	95 msec
	SM3 SYSTEM PARAMETERS	
TCODE	Code generation time	200 msec
TCLUS	Cluster formation time	50 msec
TMSG	Interrupt service time	5 msec
TBRDCST	Time to do one-to-all broadcast via shared memories	4 msec
TSWBUFF	I/O buffer switching time	2 msec
TSWITCH	Time to set/reset SM system switches	2 msec
N	Number of processors per cluster	19
M	Size of buffers and switchable memories in number of blocks	1
TUPLE_SIZE	Size of individual tuples in source/output relation	100 bytes
RECS	Number of tuples in source relation	50,000
	VARIABLES AND DERIVED VALUES	
C	Number of categories formed out of relation	Variable
R, A, B	Source relation in number of blocks	$\frac{\text{RECS*TUPLE\_SIZE}}{\text{BLOCK\_SIZE}}$
RR	Rounded value of R	CEIL(R)
RRn	Rounded value of number of blocks of the relation per processor	CEIL(R/N)
G	Number of categories expressed in blocks	$\frac{\text{C*TUPLE\_SIZE}}{\text{BLOCK\_SIZE}}$
GG	Rounded value of G	CEIL(G)

TABLE II: EXECUTION TIME FOR STATISTICAL AGGREGATION OPERATIONS

N = 19 X = 0.0

R = 50,000 tuples of size 100 bytes

Number of Categories, C	The MPC System	Modified MPC SYSTEM	SM3 System Algorithm I	SM3 System Algorithm II
5	10.2	7.9	2.4	48.7
50	10.2	8.1	3.0	48.7
250	10.3	9.7	5.7	49.0
500	19.3	14.6	9.0	49.1
2500	89.9	76.2	35.6	51.0
5000	193.2	171.7	69.0	53.0

Note: All times are in seconds.

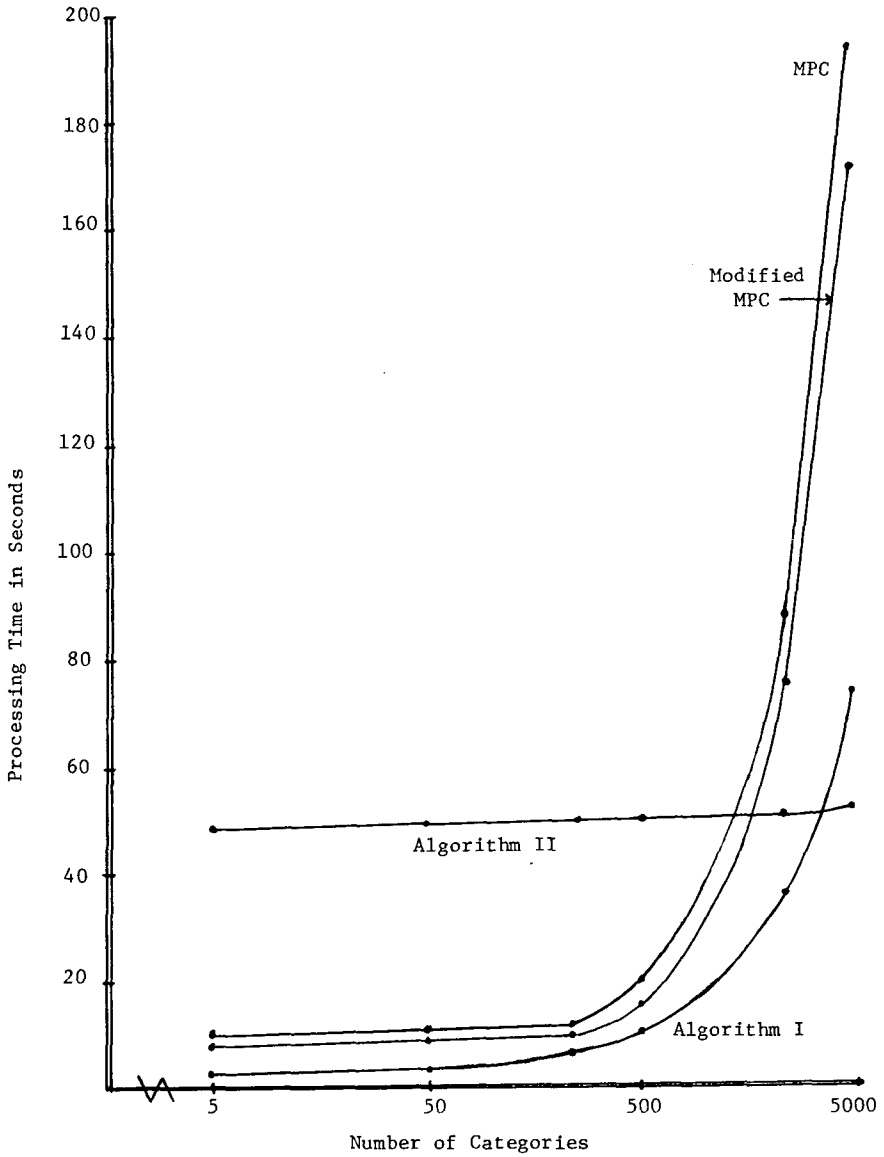

Fig. 5 Processing Time for Statistical Aggregation Operations