A virtual memory system for a relational associative processor

by S. A. SCHUSTER, E. A. OZKARAHAN and K. C. SMITH
University of Toronto
Toronto, Ontario, Canada

ABSTRACT

The Relational Associative Processor (RAP) is an experimental "backend" cellular processor for implementing data base management systems. RAP is particularly well suited to supporting Codd's relational model of data. The capacity of a RAP device implemented with current IC and memory technology can be estimated to contain 10^8 to 10^9 bits of associatively processable data. Because many data bases are larger, a virtual memory environment for RAP has been proposed and its performance simulated. The environment incorporates conventional memories for bulk storage and a single RAP processor—both controlled by a general purpose front-end computer. The system requires that the entire relational data base be divided into pages of size equal to one RAP cell memory. A buffer memory is added to RAP to permit the overlap of paging with processing. It has been found that user environments containing small relations or queries exhibiting either long processing times relative to paging requirements or some "locality" (defined as the degree to which sequences of queries reference some relations more than others) can efficiently page data between large data bases and data base machines without significant losses in performance.

INTRODUCTION

The relational model of data is an important approach to data base management because it presents its user with a simple, consistent, and operationally complete view of data.[1] However, its high-level user-oriented view creates serious problems for efficient generalized implementation on conventional hardware. We briefly review the relational model and a special purpose peripheral processor called RAP which is designed to provide high performance relational data base operations.[2-4] A small prototype version of RAP is being implemented at the University of Toronto. This paper concentrates on the problem of using RAP to support large relational data base applications. To do this, we propose several architectural extensions and a software support system to create a virtual memory environment for a RAP processor. A model of the virtual memory environment has been simulated and the results are presented and analyzed.[5]

THE RELATIONAL ASSOCIATIVE PROCESSOR "RAP"

A relation can be viewed as a table whose rows contain data about a set of similar entities. The table's or relation's name identifies the set of entities being described and the column headings are the names of the attributes which are used to describe the characteristics of each entity. Each row contains an n-tuple of values—one for each attribute—which uniquely describe each entity. The set of values that can be taken by an attribute is called a domain. A relational data base is composed of a collection of time-varying relations that may change because of modifications, insertions, and deletions. Several relations can be interrelated through common domains to formulate complex queries.

The Relational Associative Processor (RAP) is an experimental "backend" non-numeric processor designed to efficiently implement data base management systems. Its architecture and instruction set is particularly well suited for supporting the relational model of data. In addition, recent research indicates that RAP is sufficiently generalized to support set-oriented hierarchical and network views of data. This paper concentrates on RAP's relational aspects.

RAP architecture is based on the observation that relational operations of search, retrieval, statistical computation, and update are inherently associative and set-oriented. The basic organization of RAP is shown in Figure 1. The design incorporates an array (i.e., parallel set) of circularly connected associative cellular processors which are driven by a central controller and statistical processor. A RAP instruction is a data base operation which is executed by each cell in parallel directly on the cell's memory. Each cell is composed of a microprocessor and a sequential circulating memory (e.g., a track of a drum or disk, CCD's, bubble memories, etc.). The rows of a relation are stored as blocks of data on one or more cell memories. The RAP data structure allows rows to be duplicated.

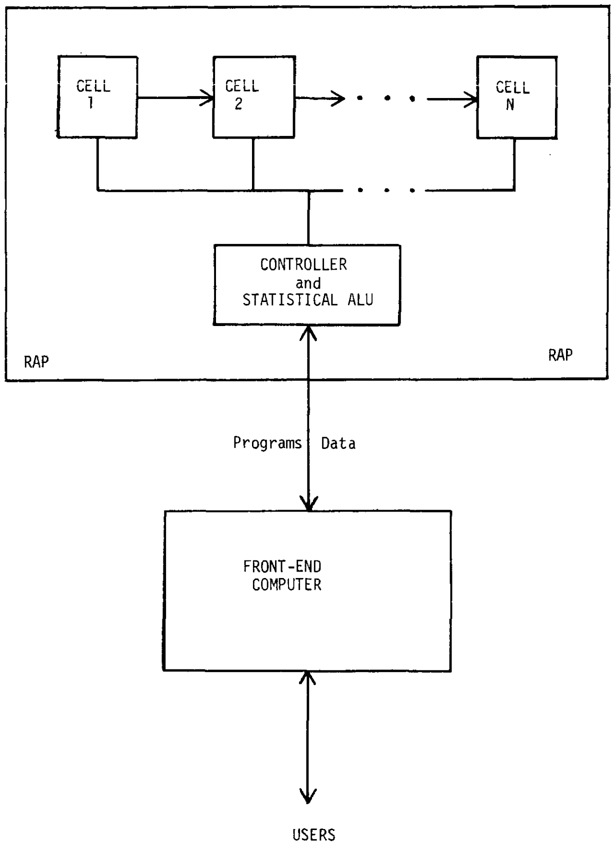

Figure 1—Basic organization of RAP architecture

Relational queries can often be constructed from just a few RAP primitives and often only a single RAP primitive is required. The RAP system is designed to execute its most important instructions within one parallel rotation of the cell memories. Because operations are accomplished in parallel on the entire storage, RAP is an effective associative processor providing significant search and manipulation efficiencies. That is, RAP accomplishes relational data base management without complex data structures and software aids such as inverted lists and hashing for multi-key searching required in conventional systems. This is especially important for applications which have extensive update activity. The extra indices and ordering requirements must be maintained whenever an update occurs in a conventional data base system. In RAP, only the relation itself has to change and this is accomplished directly on the data without having to move any portion of the data base to the front-end processor.

An analytical model was constructed which compared the performances of RAP to a conventional data base management system.[4,6] The model considered resident data bases for RAP and fast access paths in the form of inverted lists for the conventional system. The results showed that significant gains in query execution speed can be achieved by the RAP architecture over the conventional system even if it uses fixed-head disk memories. The model includes queries of the form: simple Boolean selection retrievals or updates, retrievals including statistical criteria in the query qualification, and retrievals involving implicit joins. This study indicates that, under many circumstances, on-line retrieval and updates of large data bases may only be possible with the aid of RAP-like systems.

The concept of providing large scale associative processors and memories for data base management are also currently being explored by others.[7–10] These devices are generally referred to as "data base machines". Most of the discussion to follow may also be applicable to the general theory of data base machines.

The maximum capacity of a single data base machine implemented with current IC and memory technology can be estimated to contain from 10^8 to 10^9 bits of associatively processable data.[4] Data compression, of course, should be exploited. This capacity may be sufficient for many applications but there are others which require larger storage. The costs of data base machines may not permit them to be casually duplicated when larger storage is required. This then raises the question as to what architectural extensions and software techniques can be explored to efficiently extend the address space of RAP-like data base machines. A virtual memory system for RAP is proposed which incorporates conventional disk memories for bulk storage and a single RAP processor—both controlled by a front-end general purpose computer.

RAP ARCHITECTURE FOR LARGE RELATIONAL DATA BASES

The need for the virtual memory system arises when the data base is larger than RAP memory capacity. However, it is assumed that all the pages for a query can be contained within RAP memory. This assumption is reasonable in light of the expected memory capacity for RAP (10^8 to 10^9 bits). For applications requiring very large relations, partitioning of large relations into smaller ones containing the same columns may be required. This would require a user to create a sequence of tasks to process several subqueries over the smaller relations and assemble or interrelate their results.

The proposed virtual memory system requires that the entire data base be divided into fixed size pages equal to the capacity of one cell. A page contains rows from only one relation. Each page has a unique identification. Therefore, the set of rows for a complete relation can be specified by indicating the pages that contain the rows.

The extensions to RAP architecture for implementing a virtual memory system are displayed in Figure 2. In this configuration, the front-end general purpose computer contains a virtual memory monitor and acts as an I/O interface for paging data between conventional bulk memories (e.g., direct access secondary memory devices) and RAP. A large portion of the data base resides on the bulk memories and only the "working set," that is the collection of pages containing the relations required by the currently processing queries, resides in RAP memory.

In order to reduce paging overhead, overlapping of paging I/O with query processing is essential. To achieve this, each cell has been extended to contain two memory components. At a given time, one serves as the active or processor memory while the other acts as the buffer or alternate memory. The memory serving as the active memory is used in query processing while the other acts as the I/O buffer to permit paging to take place concurrently with query processing. The roles of the memories can be reversed for each cell independently as required during the operation of the system. Thus, RAP logically contains two cyclic memories—one serving as the active associative memory while the other is a buffer. The hardware logic for switching between cell memory pairs has been designed in detail for RAP.[4] The extra logic required for doing this adds little to the complexity of the overall system.

The RAP controller is still connected to the front-end computer via a dedicated channel to receive programs and transfer results of users' requests. However, the current buffer is driven by a separate I/O controller which is connected to the front-end computer by a separate channel. The paging to and from the buffer is accomplished by standard channel programs before a new query is processed.

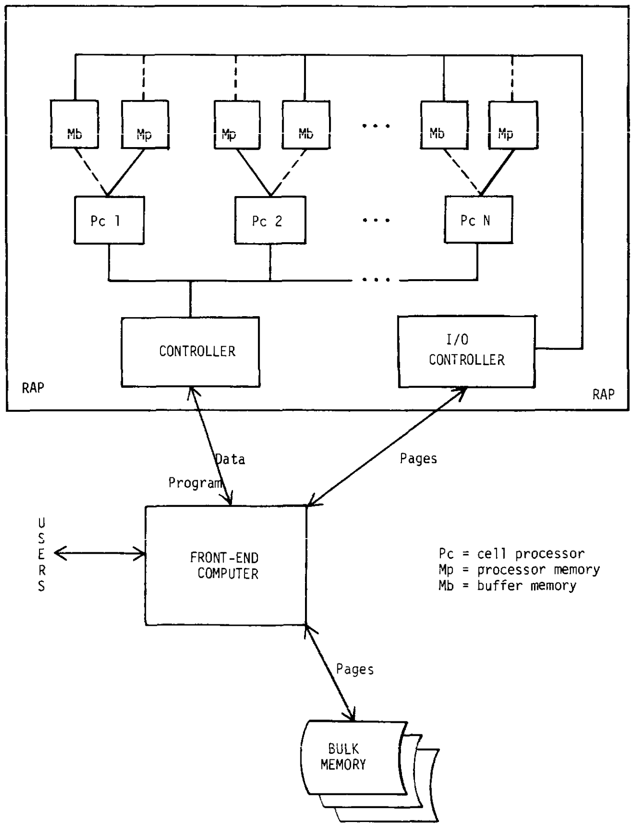

Figure 2—Hardware architecture of the virtual memory system

The paging for a waiting query is overlapped with the processing of a currently executing query. Look-ahead paging is possible because RAP associatively processes all the pages of the relations referenced in a query. Since each query must specify which relations are to be processed, it specifies which pages are to be processed at the same time. Once the required pages are transferred and execution of the current query is finished, the roles of the memory pairs can be switched.

There are two other features of the RAP design which are not shown in Figure 2 that help to increase the efficiency of the virtual memory system simulated in this study. They are the fast read-out scheme and the multiprogramming facility.[4] The fact read-out scheme for retrieval queries interleaves selected tuples from different cells during a memory revolution to maximize the channel utilization. The multiprogramming facility allows queries to be divided into two priority classes and executes them with respect to a preemptive scheduling discipline to provide a foreground-background query execution facility. This allows the RAP processor to control query execution by not allowing a long query to tie-up the device. The logic for implementing hard-wired two-task multiprogramming requires the duplication of less than 10 percent of the cell hardware.[4]

A VIRTUAL MEMORY SYSTEM

Locality

The virtual memory system to be discussed is designed to exploit user characteristics that affect total system performance with respect to paging. In conventional virtual memory systems, the important property of locality of address references in a user program is used to reduce paging. In an analogous way, we define "locality of query references" or simply "locality" as a phenomenon which is manifested by a nonrandom distribution of references to the relations in a data base during the execution of a sequence of queries. High locality is reflected when a small subset of the relations in a data base are referenced more often than the other relations, i.e., the frequency of relation reference is described by a distribution with low variance. The effect of varying degrees of locality is studied in the simulation.

Unfortunately we lack experimental data and are not aware of any studies performed in connection with locality in relational data bases. However, we will present some ad hoc arguments for its existence. Locality will be loosely examined with respect to four classes or environments of processing.

There are many data processing tasks that occur in a reservation system environment such as for hotels, airlines, shipping, etc. As a deadline approaches for an entity, e.g., the scheduled time for a plane flight, the activity in the data base is expected to become concentrated within the area representing this entity. If there exists a relation for each flight which records the passenger reservations for a particular flight, then the access to those relations scheduled to depart shortly will have greater frequency of reference resulting in high locality.

A second argument for high locality is that the use of high level interactive query languages encourages "browsing". In such an environment, a user can interactively search for relevant data by step-wise refinement of queries on the same relations. The iterative refinement of queries can cause the same relations to be accessed repeatedly during a short period of time.

In a data processing environment using batch multiprogramming, we expect that queries are generated within the constructs of host language programs. These programs are usually involved in the generation of complex reports or batch updating. The logic of such programs is often repetitive which cause queries to be embedded within program loops. This could cause sequences of queries to concentrate references over a subset of the relations in the data base.

A fourth argument for high locality is that certain relations may play an important role by interrelating other relations. These relations have a high percentage of attributes whose domains are common to other relations. These relations must be referenced extensively to link data in different relations in the formulation of many queries. Because these "linking relations" may be referenced often, queries over such a data base could exhibit high locality.

The system presented was designed to exploit high locality characteristics when they exist. The results will show that a modest amount of locality can greatly extend the processing efficiency of a data base machine for processing data bases which are large relative to the capacity of the machine's memory.

The virtual memory system structure

The principal modules and data flow of the virtual memory system are shown in Figure 3. The overall control of the system is concentrated in a central MONITOR. The MONITOR is also responsible for communicating the block of individual operations of a query to drive the RAP processor.

As jobs enter the system they are classified by the CLASSIFIER module as either class-1 or class-2. Class-1 jobs are queries that have high priority and are to be processed uninterrupted such as on-line retrievals or any update. Class-2 jobs are retrieval queries with long processing times which can be run in the background, interrupted by class-1 jobs, and

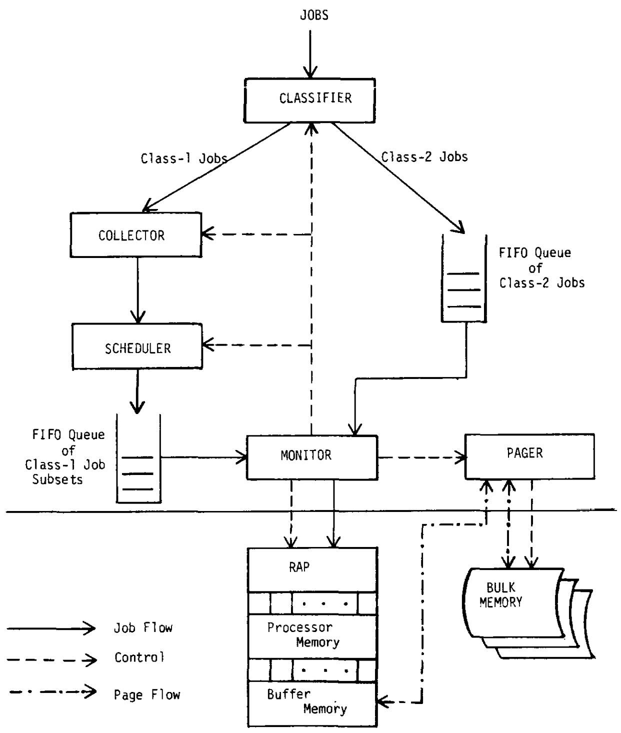

Figure 3—Structure of the virtual memory system

then resumed when no class-1 jobs are waiting to be processed. We will assume that all class-2 jobs can be resumed from their point of interruption regardless of possible effects of an intervening class-1 update job. If the semantics of the query is such that it must be completed in its entirety before any of its relations can be updated, it would be classified originally as a class-1 job. The data for each job that determines its class can be either found by examining the query's instructions or by user specified parameters. We further assume that all jobs classified as class-1 can be reordered during a small segment of time. (This policy has been made over-simplistic for purposes of the simulation. It is acknowledged that updates must often execute in a specific order.)

After classification, class-2 jobs simply enter a class-2 FIFO queue. Class-1 jobs, however, are kept in a job pool by the COLLECTOR module. Under the direction of the MONITOR, all class-1 jobs in the pool will be released to a module called the SCHEDULER. Two policies have been suggested for controlling the COLLECTOR. Only the first has been simulated. In this case, class-1 jobs are collected as long as there exist previously scheduled class-1 jobs that are waiting and/or processing. When such jobs no longer exist the COLLECTOR releases its jobs to be scheduled. If no class-1 jobs have been collected, then the current class-2 job is started or resumed if a preempted one exists. The class-2 job is allowed to process until the next class-1 job is received and preprocessed. The single class-1 job is immediately scheduled and, when its paging requirements have been satisfied, it then preempts the class-2 job and acquires the processor.

The second policy is to let the COLLECTOR collect jobs for a fixed amount of time which would be determined by the MONITOR. In retrospect, it is felt that this would have given better results because larger pools will be presented to the SCHEDULER which is designed to exploit high locality by reducing page faults through job reordering.

The SCHEDULER receives the pool of class-1 jobs and orders them by first finding the job which causes the least amount of page faults with respect to the current contents of RAP memory. This can be done because the relations referenced in the queries are known by the MONITOR. It continues to order the rest of the jobs by finding the next job whose pages most overlap RAP and so on. The ordered pool of jobs are then partitioned into an ordered set of job subsets such that all the relations for each subset can be contained within the data base machine memory. The ordered subsets are then placed in a FIFO queue to wait for the processor.

The final module to be discussed for the virtual memory system is the PAGER. It is responsible for the paging or buffering strategy and the page replacement policy. Paging is performed on a non-demand basis. This is possible because the pages required by a subset are known before processing starts. The demand paging philosophy encountered in programming systems does not apply here, because, having job subsets sitting in the queue, one can start the processing of a current subset and at the same time start paging data into the buffer for the following subset. This scheme is aimed at reducing the average amount of idle processor time that a subset causes due to paging.

The replacement policy for pages in the data base machine memory is to first replace any pages that have not been updated and are not referenced by jobs in the ordered subsets waiting in the queue. If more space is required, then those referenced farthest in the job subset are replaced. An actual page-out occurs only if the replaced page was updated, otherwise it is simply overwritten in the buffer. Page-outs occur after the role of the cells' memories, which contain the pages to be replaced or paged-out, are switched from active memory to buffer memory.

The effectiveness of the SCHEDULER and PAGER in exploiting locality is measured by an index called "system locality (SLOC)". SLOC is defined as follows. Let pg[i] be the number of unique pages that are required by subset i and pgcom[i] be the number of pages in common between subset i and the contents of RAP memory at the time when paging for subset i is to take place. Thus,

$$\mathrm{SLOC} = \sum_{i=1}^{n} \mathrm{pgcom}[i] / \sum_{i=1}^{n} \mathrm{pg}[i]$$

for i=1,2,. . ., n consecutive class-1 job subsets. The system locality will be near one for environments requiring little paging between subsets and near zero for environments requiring excessive paging. System locality depends on the degree of locality, the number of pages in the entire data base relative to the number of pages that can be contained in RAP memory, and the size of the relations in the data base.

SIMULATION OF AN ON-LINE ENVIRONMENT

Lacking experimental user data, an on-line environment has been hypothesized and simulated for the described virtual memory system.[5] The data base machine is modelled with the expected parameters for a moderately large size RAP processor. The simulation was coded in GPSS on an IBM 370/165.

The on-line environment

The application environment studied could be conceptualized either as an on-line reservation system for hotels or airlines or a parts distribution and inventory system. Most of the jobs fit the class-1 description with a few class-2 background batch jobs representing management reporting, billing, financial accounting, payroll, etc. The arrival rate of queries is fast and most require short processing times. Most queries involve only a few relations. The following is a list of parameters used in the experiments. The times are given in terms of the number of revolutions or circulations of the RAP memory.

RAP:

(a) 1 revolution (rev) = 50 milliseconds
(b) RAP processor = 200 cells
(c) RAP memory capacity = 400 pages
(d) active memory = buffer memory = 200 pages
(e) page size = $0.5*10^6$ bits
(f) paging rate = 1 page/rev

Data base:

(a) total data base size = 2000 or 5000 pages
(b) number of relations for short jobs = exponential (mean = 1.5 relations)
(c) number of relations for long jobs = exponential (mean = 3 relations)
(d) number of pages per relation = 5, 10, 20, or 25 pages

Locality:

(a) low—modelled by a uniform distribution
(b) medium—modelled by an exponential distribution
(c) high—modelled by a hyperexponential distribution

As each job (query) is generated in the simulation, one distribution is referenced, depending on the locality being simulated, to determine the relations needed by the job.

Arrival and processing time distributions:

(a) job interarrival time = Poisson (mean = 40 rev/job)
(b) processing time for short jobs = exponential (mean = 4, 8, 16, or 32 rev)
(c) processing time for long jobs = Erlang-2 for class-2 and exponential for class-1 (both with mean = 104 rev)
(d) processing time for class-1 jobs = hyperexponential (due to (b) and (c))
(e) processing time for class-2 jobs = processing time for long jobs (due to (c))

Job proportions:

(a) proportion of class-1 vs. all jobs = 0.90
(b) proportion of updates vs. all jobs = 0.40
(c) proportion of long updates vs. all jobs = 0.03

The above parameters were chosen in such a way as to create a system load which guarantees that a class-1 and/or class-2 job would almost always be present in the job mix. Thus, the RAP processor in this experiment would rarely be idle due to empty job queues. However, idle processor time would be caused by extensive paging requirements.

Simulation results

The goal of this study was to determine the relative effects on the average time-in-system, referred to as the response time, for a class-1 job when the following were varied:

(a) short job mean processing time
(b) number of pages per relation
(c) locality
(d) data base size

Also studied was the effect of these variables on system locality which reflects the paging performance from the system's point of view.

Figure 4 shows the effects of processing time for short jobs (proctime), locality, and data base size (dbsize) on RAP system response for class-1 jobs. For this experiment the number of pages for a relation was fixed at 10. The response time, RTIME, indicates the average time a class-1 job spent in the system. The results can be approximately summarized as a family of straight lines of the form "y=a*x+b". For the experiment we get:

RTIME := proctime + ((log dbsize)/locality)

where ":=" means "proportional to." That is to say, response time is directly proportional to processing time, as would be expected, with an upward shift depending on locality and data base size. The line "y=x" is shown because it represents the theoretical optimum for the environment simulated. The average response time can never be less than the average time it takes to process a job.

The somewhat linear upward shift in the curves away from the optimum is due to idle processor time because of paging. The paging requirements are reduced by high locality, but increased in a diminishing way (this will be explained later) when the total data base size grows larger relative to RAP memory capacity.

It should be noted that the curves slowly converge toward the optimum for longer processing times. This is to be expected since the number of pages for a relation are fixed in this experiment. Longer processing times allow more paging to overlap with processing and, when the processing times are long enough, there will be time to do all paging before the next job subset is to be processed.

Figure 5 shows the effects of relation size in terms of the number of pages (npgrel), locality, and data base size (dbsize) on RAP system response time for class-1 jobs. For this experiment the average processing time for a short job was set at 8 revolutions. The results can be summarized as follows:

RTIME := ((log dbsize)/locality) * npgrel + proctime

That is to say, response time is directly proportional to the size of relations and the slope of the relationship is affected by locality and data base size. The line "y=proctime" is shown since it represents the average experimental optimum as discussed earlier.

The fact that response time is directly related to the

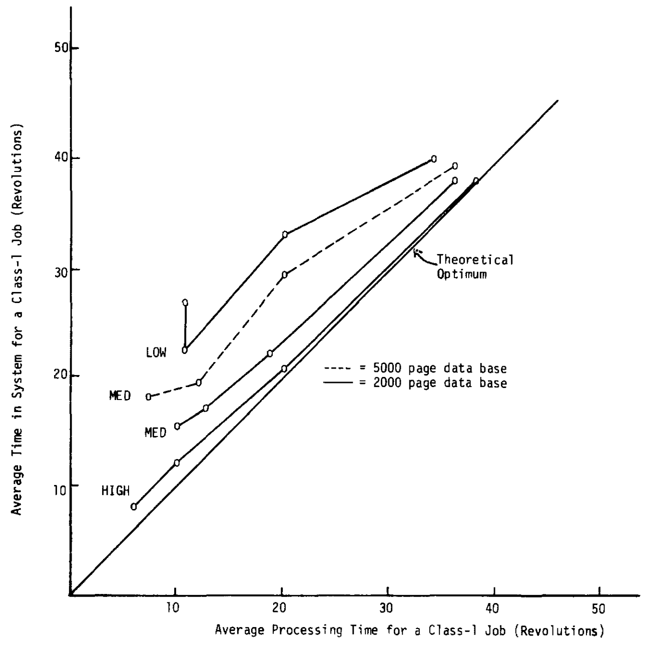

Figure 4—The effect of locality, processing time, and data base size on system response

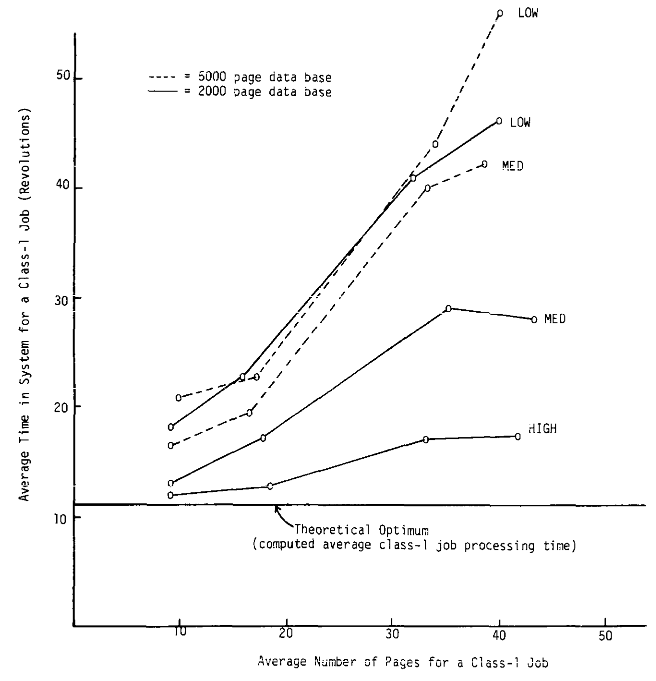

Figure 5—The effect of relation size, locality, and data base size on system response

average number of pages occurring in relations of the data base is not surprising. Since all pages of a relation are required to be in RAP memory to be processed, the more pages required, the more paging is to be expected. Given a fixed average processing time, we expected and also observed that response is degraded when large relations must be paged.

However, the effect of locality can be profound because it affects the slopes of the response time curves. When the locality is higher, there is a good chance that some of the relations currently in RAP memory will be needed by the following job subset. Therefore, the effect of paging larger relations will not be so dramatic.

Data base size also affects the slope of the response time curves. As the data base gets larger relative to RAP memory, the effect of locality is diminished. That is, the same pattern of locality will be spread over more relations causing the locality to appear more random. It should be noticed that for the low (random) locality curve, the effect of increasing the size of the data base was very small. It is observed that paging due to random locality will be independent of data base size. When extensive paging is required because of random reference, it does not matter what size of pool contains the pages.

Figure 6 shows the effects of these experiments on the average system locality (SLOC). The data from which these curves are derived also showed that system locality is independent of a job's processing time or relation size for a fixed locality distribution. The results can be summarized as follows.

$$\text{SLOC} := (1/(\log \text{dbsize}))*\text{locality}$$

That is, the higher the locality the better the system will perform since less paging will be expected. However, as the data base size grows, the effect of locality will diminish and performance will approach that of a random locality environment.

SUMMARY AND CONCLUSIONS

The Relational Associative Processor (RAP) is an experimental "backend" non-numeric processor for implementing a data base management system which supports Codd's relational model of data. RAP architecture is based on the observation that relational operations of search, retrieval, statistical computation, and update are inherently associative and set-oriented. The design incorporates an array (i.e., parallel set) of interconnected associative cellular processors which are driven by a central controller and statistical processor. Each cell is composed of a microprocessor and a sequential circulating memory (e.g., a track of a drum or disk, CCD's, bubble memories, etc.). Each data base operation is executed within the cells which operate in parallel directly on their memories. RAP accomplishes a complete set of data base operations without the need for indexing and its associated software and maintenance.

The capacity of a RAP device implemented with

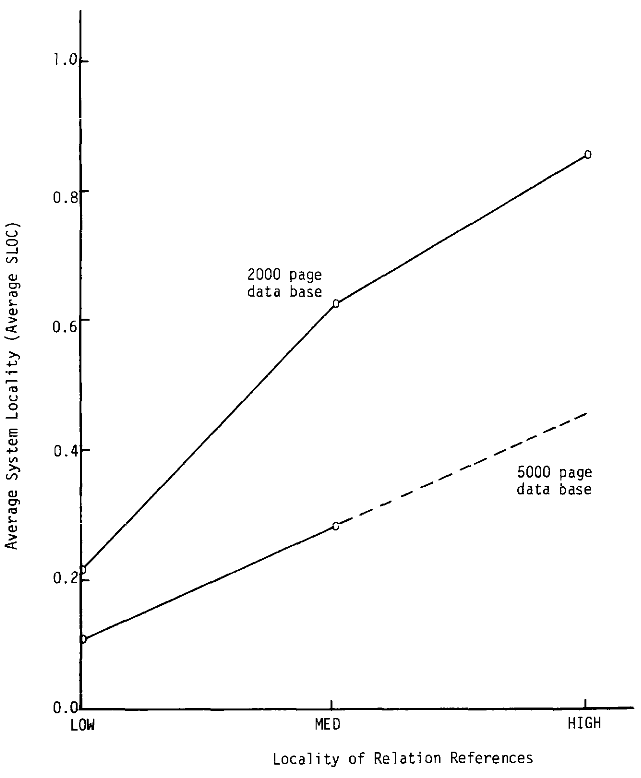

Figure 6—The effect of locality and data base size on system locality

current IC and memory technology can be estimated to contain from 10^8 to 10^9 bits of associatively processable data. Because of this limitation, a virtual memory environment for RAP has been proposed and its performance simulated. The environment incorporates conventional memories for bulk storage and a single RAP system—both controlled by a front-end computer. An entire relational data base is divided into pages of size equal to one RAP cell memory. A buffer memory is added to RAP to permit the overlap of paging with processing.

Each RAP program represents one relational query (job) on one or more relations. The system maintains an ordered queue of waiting queries. Before the next selected query is processed on RAP, a table look up is made to determine if all the pages for the query are present in RAP memory. If any pages are not currently residing in RAP, they are paged in replacing the pages for the relations referenced farthest in the job queue and/or those not needed at all such that pages that have not been modified are replaced first. The jobs waiting to be processed are scheduled by ordering them into a queue in such a way that paging is reduced. It has been found that user environments which contain small relations or process queries exhibiting either long processing times relative to paging requirements or some "locality of relation reference," can efficiently page data between large data bases and data base machines without significant losses in performance.

ACKNOWLEDGMENT

The authors are happy to acknowledge the contribution of Raymond Nakano who developed the simulation program and made many valuable comments throughout the preparation of this paper.

This work has been supported by the Federal Department of Communications, the Department of Supplies and Services, and the National Research Council of Canada.

REFERENCES

1. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *CACM 13*, 6, 1970, pp. 377-87.
2. Ozkarahan, E. A., S. A. Schuster and K. C. Smith, *A Data Base Processor,* CSRG-43, Computer Systems Research Group, University of Toronto, September 1974.
3. Ozkarahan, E. A., S. A. Schuster and K. C. Smith, "RAP—An Associative Processor For Data Base Management," *AFIPS Conf. Proc.*, Vol. 44, 1975, pp. 379-87.
4. Ozkarahan, E. A., *An Associative Processor For Relational Data Bases—RAP,* Ph.D. Thesis, Dept. of Computer Science, University of Toronto, January, 1976.
5. Nakano, R., *A Simulator for a RAP Data Management System,* M.Sc. Thesis, Department of Computer Science, University of Toronto, 1976.
6. Ozkarahan, E. A., S. A. Schuster and K. C. Sevcik, *Performance Evaluation of a Relational Associative Processor,* CSRG-65, Computer Systems Research Group, University of Toronto, January 1976.
7. DeFiore, C. F. and P. B. Berra, "A Data Management System Utilizing an Associative Memory," *AFIPS Conf. Proc.*, Vol. 42, 1973.
8. Moulder, R., "An Implementation of a Data Management System on an Associative Processor," *AFIPS Conf. Proc.*, Vol. 42, 1973.
9. Su, S. Y. W. and G. J. Lipovski, "CASSM: A Cellular System for Very Large Data Bases," *Very Large Data Bases Conf. Proc.*, Vol. 1, No. 1, September, 1975.
10. Lin, C. S., D. C. P. Smith and J. M. Smith, "The Design of a Rotating Associative Memory for Relational Data Base Applications," to appear in *TODS*, Vol. 1, No. 1, 1976.