Exploiting Parallelism in a Relational Associative Processor

Paul J. Sadowski† and S. A. Schuster‡

Computer Systems Research Group
University of Toronto

Abstract

The Relational Associative Processor (RAP) is a special purpose non-numeric back-end processor used in supporting general Data Base Management Systems. In particular, it is ideally suited for supporting a relational data base. The architecture and instruction set of RAP are discussed in this context. It is the purpose of this paper to show that RAP performance can be enhanced considerably by more fully exploiting its parallel nature. It is shown that a greater degree of concurrent activity will result in better overall performance. An operating system executive to support this high level of concurrency is proposed and modelled using simulation techniques. Both the analytic and simulation results support the value of the proposals.

1. Introduction

Data Base Management Systems (DBMS) are generalized tools for maintaining and manipulating data. Their functions include the responsibility for data integrity, providing security mechanisms so that different users (possibly sharing the same data) do not adversely affect each other, providing centralized control of the data base, etc. [SIBL 76]. The goals of a DBMS are geared toward the end-user. These include the support of high level logical data views so that users are not encumbered with implementation details, and high level interfaces so that users can query a data base in a language which reflects the user's needs and is not too far removed from English [AC 75].

Unfortunately, the higher the level of abstraction presented to the user, the greater the disparity becomes between a user view of the data base and its physical realization. This occurs as a result of the inability of current hardware to provide physical data structures that match logical ones. In order to overcome this disparity, huge efforts must be made in software support. The added weight of this software, however, tends to degrade performance considerably in terms of response time to queries, storage requirements, reliability, etc. [SNOS 77]. In order to alleviate this situation, special hardware devices which support modern data base management systems have been designed. The Relational Associative Processor (RAP) is one such device. It eliminates the need for expensive mapping mechanisms between user logical data structures and hardware storage structures. The associative search capability of RAP also eliminates the need for access paths (software mechanisms providing efficient search on selected portions of a data base) found in conventional systems. RAP is particularly well suited for supporting a relational data base and its architecture and instruction set are discussed in this context. A performance evaluation of RAP [OSSE 77] has already shown it to be substantially superior to conventional systems. However, in the latter part of section 2, we describe a memory organization which can considerably improve RAP performance. It is this performance enhancement under a new memory configuration that is the main theme of this paper.

Finally, a simulation study is undertaken (section 5). It models the new RAP organization under a variety of conditions. The goals of the study include showing the feasibility of the proposals along with a performance evaluation of RAP in a new context.

2. RAP Environment

2.1 RAP Architecture

Figure 1 illustrates the overall RAP organization. It can be seen that RAP itself is composed of three major components: a controller, a statistical arithmetic unit and a chain of "cells". The controller is responsible for coordinating activities on cells. It receives RAP instructions from a front-end or host machine and initiates their execution on appropriate cells. Any data items passed between RAP and the front-end (in particular, data insertion and retrieval) are also the responsibility of the controller. The purpose of the statistical arithmetic unit is, as its name suggests, the gathering of statistics such as averages, minimum/maximum etc., over sets of qualified records.

The main feature of RAP is the parallel arrangement of cells. Each cell, according to the original design, uses a circulating serial memory with a specially designed processor to support the operations of a DBMS. For example, comparator elements facilitate the testing of data base contents (specifically, data items) against literals, in an associative manner. This means that record or tuple qualification is directly performed by the hardware. Also, because the cells operate in parallel, directly on the data, a data base can be so distributed as to make operations on it independent of its size.

† Present affiliation: Ontario Hydro, Data Management Department, Toronto, Ontario.

‡ On leave at Intel Corporation, Memory Systems Division, Sunnyvale, California.

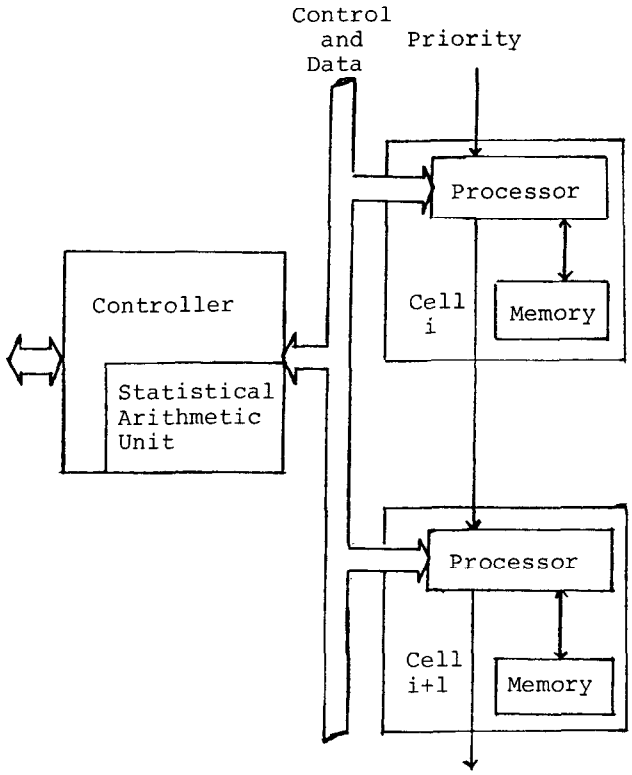


Figure 1. RAP System Architecture

The data structure chosen for RAP cells is referred to as a "RAP relation". Essentially this is a relation as defined by Codd [CODD 70] in normalized form, except that duplicate tuples are allowed and several hardware-controlled, one bit fields known as *mark bits* are added. The *mark bits* associated with each tuple, usually transparent to the end-user, provide support for high-level assembly language control of selection and other operations.
The current hardware design requires that at any given time there must be at most one relation residing on any given cell. If a relation has more tuples than can be handled on one cell, then tuples of that relation can be stored on several cells.

As far as the user is concerned, much of the RAP operation is transparent; only the general purpose front-end computer is seen. The *front-end* (mainframe) supports the user community with the necessary I/O and data communication facilities, compiles queries from a high level query language such as SEQUEL [AC 74], and sends RAP primitives to the controller. It is also responsible for the maintenance of internal tables pertaining to the data base, and issues such as security, integrity and prevention of deadlock.

2.2 The RAP Instruction Set

The set of instructions allowed to operate on RAP relations include the following:

(a) selection
(b) retrieval
(c) update
(d) statistical calculation
(e) insertion/deletion
(f) relation creation/destruction
(g) decision and transfer (i.e. flow control)

The majority of RAP instructions are based on the following format:

{<label>} <opcode> {<mark option>}
{<object>: <qualification>]} {[<parameter>]}

The optional *label* is a symbolic instruction address, while the *opcode* specifies the operation. As an example we describe one RAP instruction. The **Select** instruction picks qualified tuples from a given relation and sets or resets the mark bits of those tuples according to a marking specification. The format is as follows:

{<label>} **Select** {<mark option>} [Rn: <qualification>]

For example, the instruction:

Select mark(M1) [R1: D1 = 'a' & D2 = 'b']

will set (to 1) the mark bit M1 of all tuples in R1 whose data items satisfy the boolean condition D1 = 'a' and D2 = 'b'. While other cells may be busy executing different instructions, the selection instruction will be executed only by those cells containing R1. It will be executed within one pass (scan) over the cell memory(s) in a time that is dependent on the speed of the cell processor, memory access architecture, and the capacity of cell memory. Those same tuple mark bits could be reset with the same instruction by simply changing the *mark option* to **reset**(M1). If the mark option is omitted, then the instruction is equivalent to a null operation. A detailed description of the RAP instruction set can be found in Schuster et al.[SNOS 77].

2.3 Performance Enhancement

It can be noted from the previous sections that both the architecture and instruction set of RAP are well suited for supporting the relational view of data (although it is also capable of sustaining both network and hierarchical models). A recent study [OSSE 77] shows some of the performance characteristics of the original RAP system modelled under varying query/user environments. The results indicate a marked superiority over a conventional system (in particular an inverted file system), even when favourable assumptions for the conventional system were made, such as its use of head-per-track disks and that no CPU time, only I/O time, was modelled. Although the models used were only first order accurate, RAP showed an *across the board* superiority in that the results were good in both retrieval and update situations. A conventional system would normally be tuned to one or the other.

However, recent developmental changes have inspired extensions to the original hardware design and overall performance considerations giving rise to the RAP.2 processor [SNOS 77]. These are illustrated by the following recent advances:

(1) Hardware experience has led RAP primitives to evolve into a more consistent and complete instruction set, including features for controller register manipulation.

(2) Developments in semiconductor technology have initiated the use of block addressable memory (CCD, RAM, magnetic bubble device etc.) in the RAP cells.

(3) By using state variables which determine cell status [SNOS 77], instructions can now be selectively broadcast to one or more cells. Each cell can thus be thought of as an independently processable unit.

The second of these is clearly the most crucial. The cell organization becomes substantially different. In fact, in the current implementation, as opposed to the original serial memory configuration, a cell consists of 60 x 18 256-bit words. This leads to a pseudo-random addressing capability which has considerably improved RAP performance.

Since the memory is configured as separately addressable blocks, one possible extension to RAP is to eliminate the restriction of only allowing one relation per cell. The idea here is to maximize the concurrency in query execution on top of the already parallel organization of cells. Consider, for example, a data base of only one relation consisting of 100 tuples. Clearly if all tuples reside on one cell then execution time would be 100 units (assuming 1 unit time per tuple). If, on the other hand, the tuples were split over two cells (50 tuples on each cell), then execution time is reduced by 50% to 50 units, disregarding any system overhead. This can be extended to include a distribution over 3, 4, ..., N cells.

With this horizontal distribution of relations *across* cells, as opposed to the original vertical layout or one relation per cell, a little bit of analysis indicates a substantial improvement in RAP performance.

It is this possibility of performance enhancement under a new memory configuration that is the main theme of this paper. The following section examines various organizations and situations geared toward performance considerations. A simulation study is then undertaken to illustrate the feasibility of some of the proposals, and provide analytic validation and some performance prediction under varying conditions.

3. Concurrency in RAP

The ensuing discussions embrace the philosophy of maximizing the concurrency in RAP with the intent to improve performance. Clearly, the more that is done in parallel, the better will be the overall performance (until, of course, the management of parallelism starts to degrade its benefits). It is in this context that RAP performance issues will be described.

3.1 Single Query Processing

(A) Accessing one relation:

The greatest degree of parallelism, on an individual relation basis, can be achieved by splitting the relation over as many cells as possible. The ultimate situation is that of having N tuples of a relation spread over N cells. However, under these circumstances, system overhead begins to play a significant role. The system overhead described here refers to the time required by the controller to issue instructions to the appropriate cells. Clearly, if a relation resides on only one cell, then each instruction is issued once. However, if that same relation is spread over N cells, then each instruction must be issued N times. In fact, on the current implementation, each instruction is loaded only once, but each cell on which it will operate must have its status set to reflect its readiness to accept that instruction. This results in a substantial increase in the amount of time spent in the controller. The following analysis will illustrate the situation.

Let
NUM_TUPLES=number of tuples
C = instruction complexity (time required to operate on one tuple)
I = controller interaction (time required to prepare a cell to receive an instruction)
L = time required to load an instruction into the appropriate cells
NUM_CELLS = number of cells
TIMEH = time required to execute one instruction on a relation spread across NUM_CELLS cells (where there is an equal number of tuples on each of the cells).
TIMEV = time required to execute one instruction on a relation on one cell (special case of TIMEH).

Therefore:

$$TIMEV = NUM\_TUPLES \times C + I + L$$

$$TIMEH = \frac{NUM\_TUPLES}{NUM\_CELLS} \times C + NUM\_CELLS \times I + L$$

In order that **TIMEH < TIMEV** holds, then:

$$NUM\_CELLS < \frac{NUM\_TUPLES \times C}{I}$$

The above inequality thus provides a bound on the superiority of a horizontal distribution over a vertical one. An optimal solution can in fact be found by differentiating TIMEH with respect to NUM_CELLS and setting the result to 0. We get:

$$optimal\ NUM\_CELLS = \sqrt{\frac{NUM\_TUPLES \times C}{I}}$$

These values, however, are derived on a single instruction basis. They do not take into account the arrival rate of requests, the number of responders (i.e. those tuples which satisfy the request), varying instruction complexity, etc. All these factors will result in queueing at the controller, which will add to the overhead and consequently degrade response time. The results quoted above can thus be noted as being optimistic. It should also be pointed out that if setting cell status is bussed, then the optimal number of cells in a horizontal organization is exactly the number of tuples (i.e. one tuple per cell).

(B) Accessing more than one relation:

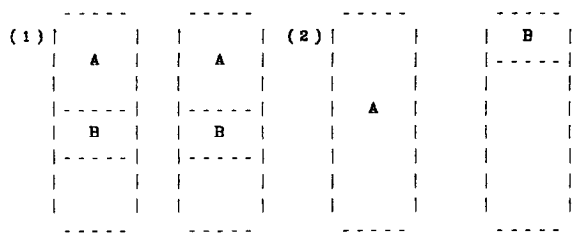
(1) Selection:

Optimal concurrency, for each relation, is achieved as described in (A) above. Associated with this are the following two points:

(a) *time in cells* is the same regardless of whether x relations are each evenly distributed over N cells (with each cell handling portions of all x relations) or each of x relations reside on N/x cells (with each cell handling only one relation).

(b) system overhead, as defined in (A), will be greater when relations are spread over as many cells as possible.

The statement made in (a) above, however, does not hold if one relation contains many more tuples than another. For example, assume relation A has 100 tuples and relation B has 2 tuples. The following diagram illustrates two possible cell configurations.



In the first configuration the time taken for a selection over both relations is 1 + 50 = 51 unit (1 unit per tuple), whereas in arrangement (2) the time required is 100 units of time, because of concurrency.

The previous calculation does not include system overhead associated with the appropriate tasks. This, however, should be negligible since RAP access time is on the order of milliseconds, while the controller cycle time would be measurable in microseconds.

The aforementioned analysis and results seem to indicate that, in general, there should be a trend toward equalizing the number of tuples on cells. Configurations of this sort would yield better performance by reducing disparities between cell execution times.

3.2 Multi-Query Processing

(1) Single relation access

In the original RAP design, where only one relation resided on a given cell, access by multiple queries was easily resolved since execution had to proceed serially. With the proposal that relations be spread across cells, the following two possibilities present themselves (assume N cells and 2 queries):

(a) - first query access
    - second query access
(b) - have one query access the first n/2 cells while the other query accesses the latter n/2 cells
    - after the selections have taken place, the queries switch the cells on which they were working and proceed.

In either of the above situations, the "time in cells" is the same except that there will be a higher system overhead associated with (b). The overhead here refers to the necessity of maintaining separate instruction queues for each cell so that the above preemptive mechanism could take place. Also, the instructions will have to be bussed twice. An important point to be noted here, though, is that if the predominant performance measure is user response time, then approach (b) will provide faster initial response (i.e. passing data to the user from qualified records as they are found, rather than performing a complete selection before responding to user requests). The preceding is an indication of scheduling issues which will be dealt with in the following section.

4. Scheduling

4.1 A Scheduling Model

The following points, adapted from [SU 75], describe a scheduling model for RAP.

(1) N jobs to be scheduled. Jobs are *not* independent (since queries access shared data and updates may impose precedence constraints). Exact attributes of a job are given upon its arrival, and there is no advance knowledge of when the next job will arrive.

(2) There is a fixed number of cells. They are *not* identical (since they can each handle distinct relations) and are available continuously from time t=0.

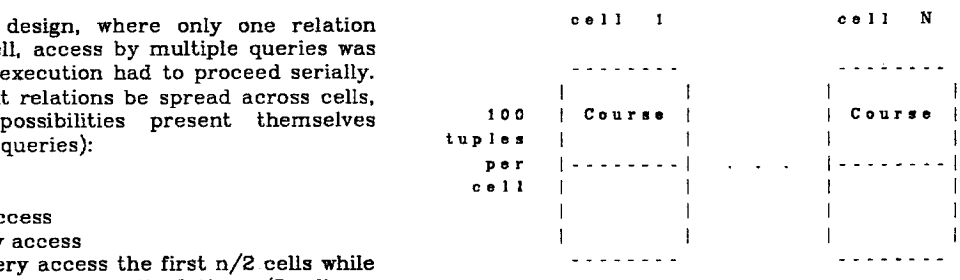
(3) Each processor can be assigned to at most one job at a time, but each job can be scheduled on *more* than one processor (since a relation may be spread over more than one cell). A processor here refers to a RAP cell.

(4) Processing of jobs cannot be suspended at *any arbitrary* time, but only at specified intervals (at the end of a revolution).

(5) Depending on the task, it can be assigned only to specific processors and not arbitrary ones. In particular, a query cannot run on any arbitrary cell which happens to be available. It must be scheduled on specifically those cells which contain the relation the query will access.

4.2 Low Level Scheduling

The following discussion embraces analysis of three completely different but acceptable dispatching policies that can be used at the RAP cell level. Consider the following data base with corresponding queries.



Query (1)
1. **Select mark**(M1)[course:course.meeting_time='mwf10']

2. **Select mark**(M2)

   [course:course.**mked**(M1) & course.year_of_study='2']

3. **Read_All** [course(name,time,place):**mked**(M2)]
4. **Select reset**(M1M2)[course]

Query (2)

1. **Select mark**(M3)[course:course.meeting_time='mwf10']

2. **Select mark**(M4)

   [course:course.**mked**(M3) & course.year_of_study='3']

3. **Read_All** [course(name,time,place):**mked(M4)**]

4. **Select reset**(M3M4)[course]

Based on the assumptions that there are N cells, each containing 100 tuples, and that system overhead is negligible (due to order of magnitude speed differences between RAP cells and the controller), the following policy calculations are made.

[A] First execute query (1); upon completion, execute query (2). In this scheme the total time required to complete both queries is 400 + 400 = 800 + transfer time (assuming 1 unit per tuple). However, because of the implied precedence, the disparity between mean finishing times is very high (exactly 400). This type of situation is definitely undesirable due to its unsolicited (and possibly time dependent) priority scheme.

[B] Intermix the instructions (i.e. on any given cell first execute an instruction from query (1) and then execute an instruction from query (2)). Under this arrangement the total completion time is 8 × 100 = 800 + transfer time. As can be seen, the total time is the same but the mean times are converging to a more consistent level (i.e. the disparity between mean finishing times has now been reduced from 400 to 100 time units).

[C] Have the first stage consist of query (1) executing on the first N/2 cells (assume N is even for convenience) while query (2) operates on the latter N/2 cells. After this stage completes, have query (2) run on the first N/2 cells and query (1) on the latter N/2. What this policy accomplishes is that again the total time required for completing both queries is 800 + transfer time, but the mean finishing times are now exactly the same and the initial response time (time between user hitting return on his terminal and receiving the first character of the query response) is faster.

From the above computations it can be seen that the latter two policies are preferable, with [C] being the most desirable. To facilitate either one of these disciplines it becomes necessary to have separate job queues for each cell. This will certainly increase overhead in the RAP environment, but the critical issue that stands out more so is that RAP now evolves into something entirely different. No longer does one have to look at *relation at a time* access, but rather the view is shifted to *cell at a time* capabilities. The reader should be aware of the impact of this development. In the original design (and on the current prototype) instructions are issued to all cells containing the relation being accessed. With *cell at a time* capabilities, instructions can be broadcast to individual cells at different times. This would allow a preemptive dispatching policy, such as that described above, to be employed.

With this as a background, we present a simulation study which looks at some of the issues and performance concerns of RAP and sheds some light on them, while also showing the reality and practicality of the proposals. In particular, the proposed distribution of relations *across* RAP cells and the proposed *cell at a time* access are investigated.

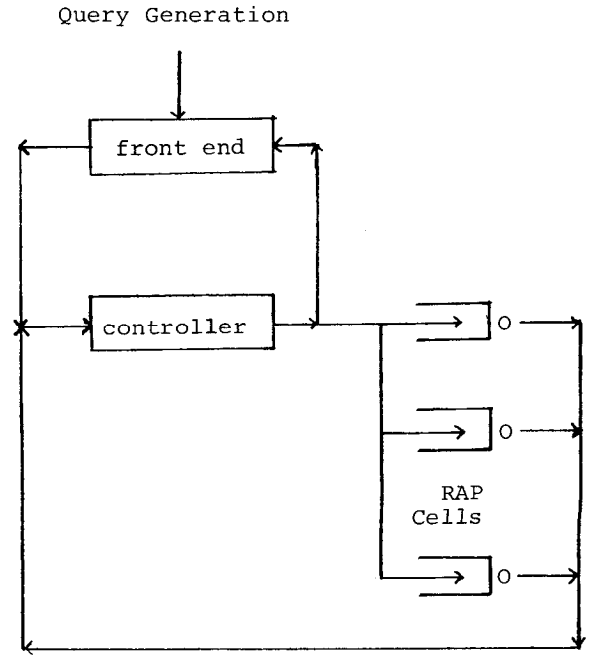
5. A Simulation Study

5.1 The Model

This section presents a refined version of the ideas discussed earlier. It in fact describes the proposed RAP executive in finer detail.

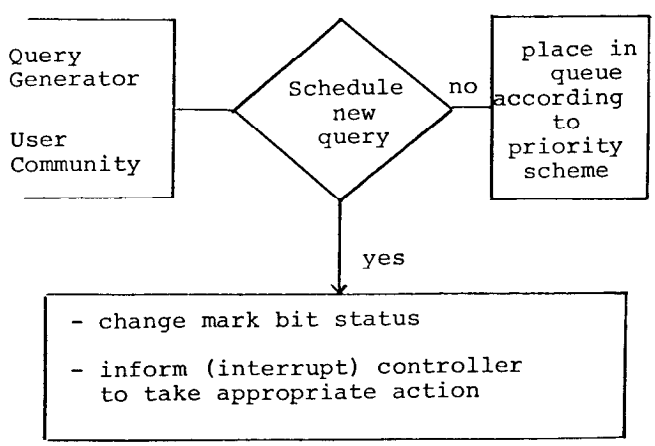
5.1.1 Overall View

The following diagram illustrates a RAP-based system from a high level viewpoint (see also Figure 1). The directed lines represent communication paths between the various devices. The representation of RAP cells is adapted from queueing theory models to show clearly that there are individual queues for each cell. These queues, however, are maintained by the controller.



5.1.2 High Level Scheduling

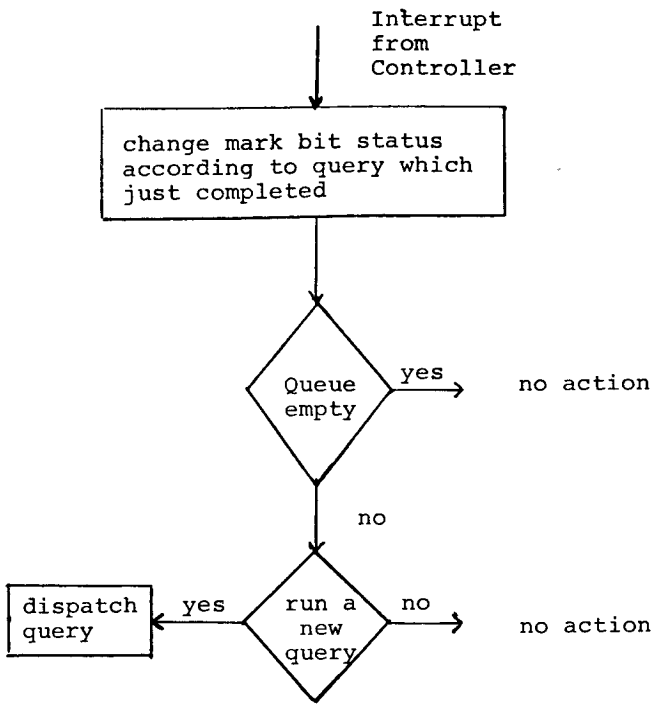
There are two distinct scheduling algorithms which must be performed by the front-end processor. One of these occurs at each query arrival while the other takes place at each query departure or completion. The following illustration points out the decision process necessary at each query arrival.



Essentially, only two choices are possible. Either the query is allowed to enter the current mix, or it is placed on a queue according to some priority discipline. The decision is based on mark bit availability, the number of queries in the current mix, and the scheduling policy in force. If the new arrival is permitted to proceed, then the controller is notified (via the interrupt mechanism) to take the appropriate actions. If, on the other hand, the query is blocked, then it must be placed in a queue according to a predefined priority scheme. This scheme could be FCFS (First Come First Served), SJF (Shortest Job First), or possibly an ordering based on mark bit usage.

Note that a new arrival can be admitted to the mix even if the waiting queue is not empty. The scheduling policy chosen will have to decide whether or not this would be allowed.

The second type of scheduling that takes place in the front-end processor occurs at each query completion.



Every time a query terminates execution, the waiting queue (assuming it is not empty) must be scanned to determine if a waiting query can be started. There are three possible disciplines which can be used in order to arrive at a decision.

(1) If there are sufficient mark bits available, then initiate the first job on the queue (strictly FCFS).

(2) Scan the queue and if there exists a query which can run (i.e. sufficient mark bits are available), then dispatch it.

(3) Scan the queue and initiate one or more jobs which can run.

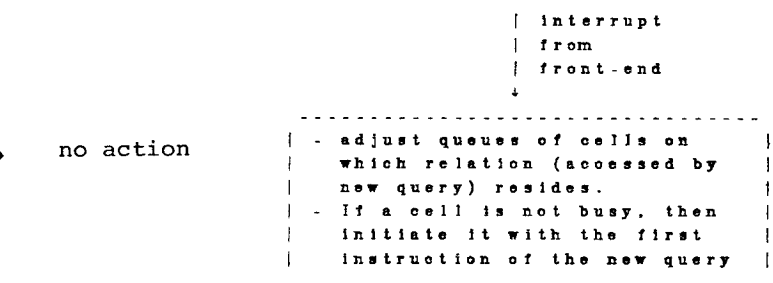
It should be noted that the only dispatching qualification for a given query is mark bit availability.

The reader should be satisfied that the scheduling decisions made at the front-end processor can substantially affect performance characteristics such as average user response time. The scheduling discipline chosen, then, should be geared toward the particular application environment in which it will operate. For example, an SJF priority ordering would be applicable if there exist large disparities in query execution times and the shorter requests represent a greater percentage of the mix.

As a final note, all high level scheduling must also take into account and enforce any precedence constraints imposed by updates.

5.1.3 Dispatching

Whenever the front-end releases a query to the controller (via a query arrival or completion), the controller must then take actions associated with dispatching that query to the appropriate RAP cells. The following diagram illustrates the situation.

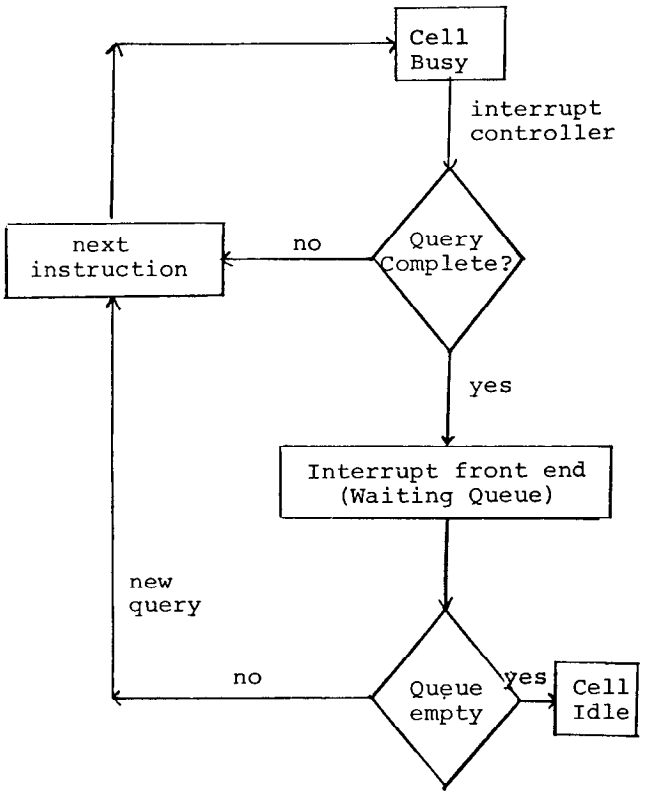


Depending on the relation being accessed, the appropriate cell queues are adjusted according to a predefined dispatching policy. For example, a First Come First Served scheme would be the most basic and easiest to implement. A preemptive policy, however, may yield better results (see section 4.3).

5.1.4 RAP Cell/Controller Cycle

The ensuing diagram is presented in order to illustrate the interaction environment of a RAP cell and the RAP controller. The controller simply issues instructions to an individual cell in a FCFS fashion from the cell queue established by the dispatcher.

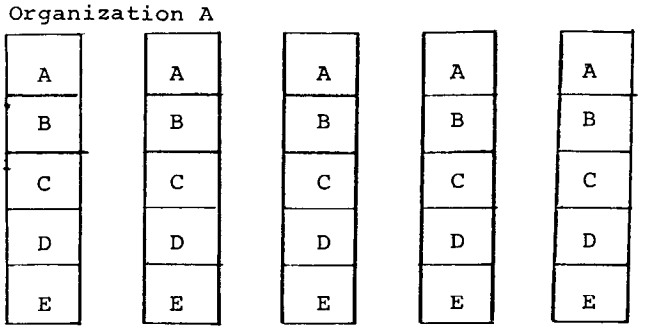
Whenever a query finshes executing, the front-end is notified via an interrupt. At this point, the high level scheduler will check the waiting queue (assuming it is not empty) to see if a blocked query can be initiated. In the meantime, the controller continues issuing instructions to the cell which just interrupted it from that cell queue, assuming it is not empty. If the cell queue is empty, then the cell simply enters an idle state.



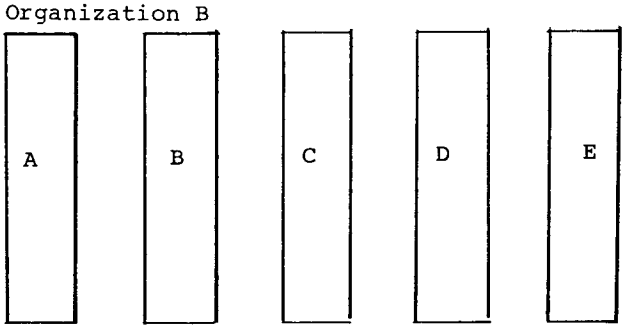
5.2 Model Validation

Essentially, the validation of a simulation model consists of running the simulation using an environment for which the results are known directly or can be derived, exactly or approximately by analytic methods. In this specific evaluation, the latter route was undertaken. The particular case used involved two distinct memory organizations which reflect the proposed RAP architecture against the current implementation. The following diagrams illustrate the point:

Organization A



Organization B



The above diagrams clearly illustrate two diametrically opposed views. Organization A shows how relations could be split horizontally over cells to achieve a greater degree of concurrent execution. Organization B, on the other hand, reflects the current status of the RAP machine, where only tuples of one relation may reside on any given cell.

The query stream used in the validation process and subsequent tests remains fixed with the following characteristics:

1) all queries are composed of N instructions, where N is uniformly distributed on the closed integer interval [4,8].
2) each instruction has a complexity of 1.
3) mark bit usage is based on the following distribution
   Probability(choose 1 mark bit) = 0.60
   Probability(choose ≤ 2 mark bits) = 0.90
   Probability(choose ≤ 3 mark bits) = 0.97
   Probability(choose ≤ 4 mark bits) = 1.00
4) the arrival stream is a Poisson process
5) all relations have an equal probability of being accessed

The following system parameters also remain fixed throughout the course of the evaluation:

a) both high level schedulers and the dispatcher operate on a strictly first come first served basis.
b) execution time of the controller is given on an interaction basis, where an interaction involves several assembler or machine instructions.
c) a maximum multiprogramming mix is strictly enforced in the controller. In the majority of runs this is set to 4. Therefore, if 4 jobs are executing in the current mix, then no more queries are allowed to enter, regardless of mark bit availability.

In the model validation, the arrival rate of queries is set to .01 and the controller speed (length of an interaction) is set to 1. The results are given in the following table:

Organization	Controller Interactions	Wait for Controller	RAP Time	Front-End Time
A	666/30	0.0	666	673
B	167/30	0.4	661	6666

There are three major points to be noted from the above figures. First of all, the time spent in RAP is considerably better when organization A is used as opposed to B. Also, the time spent waiting for the controller can be disregarded in both configurations. Finally, there is a greater number of controller interactions in environment A (exactly 5 times as many).

The results just mentioned coincide directly with the analytic predictions made earlier. In fact, the number of controller interactions matches exactly (as it should), while the waiting time for the controller shows it to be negligible compared with RAP time (which includes queueing at the controller and queueing at the cells). It should be pointed out that the results quoted above are taken from one particular query stream. Other tests were performed using different query streams that exhibit the same characteristics. Unfortunately, the number of tests required to yield a meaningful statistical evaluation, even to achieve reasonable confidence intervals for first order accuracy, are beyond the resource and time constraints of this paper. The tests performed, however, are all in agreement with the trends they illustrate. For example, a similar query stream yielded the following results:

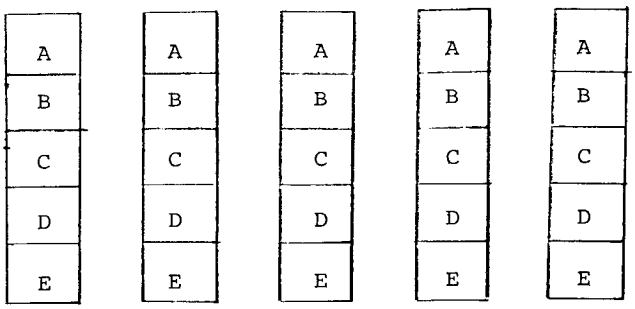
Organization	Controller Interactions	Wait for Controller	RAP Time	Front-End Time
A	930/30	1.7	478	886
B	186/30	0.1	857	5068

Again, the superiority of a horizontal layout of relations as opposed to vertical one is noted.

The confirmation of previous analysis leads us now to use the simulation model in determining RAP performance characteristics under varying environments.

5.3 Tests/Results

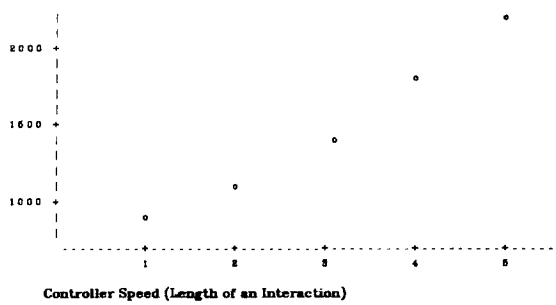
The majority of the tests were run using the following data base:



Each of the five relations is composed of exactly 100 tuples and the tuples are distributed evenly so that individual cells have 20 of each. The parameters defining the query stream and certain aspects of the RAP system remain fixed under the conditions presented in the previous section.

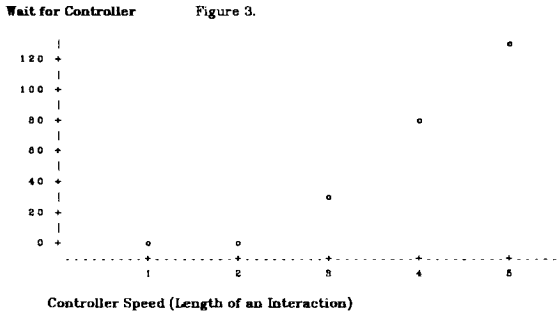
One of the first tests of the new RAP organization was determining the effect of variations in the speed of the controller with respect to a RAP cell. On the current implementation, a PDP-11/10 CPU is used as the controller. The instruction times for this machine are very well defined. Unfortunately, RAP instruction times are dependent on the technology used. In the simulation study, the cell speed is thus set to a constant 1 unit time per tuple. Controller and front-end cycle times are set on a ratio or percentage basis against this constant. For example, the following test varies the ratio of controller to cell speed from 1 to 5 (i.e. the controller is slowed down to being 20% the speed of a cell). The accompanying graph (Figure 2) yields the results of the test (the ordinate represents overall response time).

Overall Response Time Figure 2.



Coupled with the above data, we present a graph (Figure 3) depicting average waiting time at the controller as a function of controller speed.

Wait for Controller Figure 3.



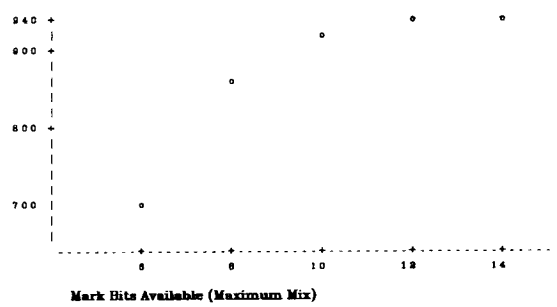
It should be pointed out again that the results just quoted apply to a specific query stream. Similar query streams, however, exhibit the same behavioural trends.

The major point to be noted from the previous two graphs is that, though waiting time at the controller increases dramatically with slower controller interactions, the net result is not so substantial when considering overall time in system. Essentially, this occurs because the waiting time at the controller is a small percentage of the overall time. In fact if a more realistic data base was used (i.e. much more than 20 tuples per cell), then this percentage would be even smaller and thus even less noticeable. Another point to be noted is that these numbers are very dependent on the relationship between controller speed and RAP cell speed. Performance measurements of the current prototype have shown that cell speed is approximately 300 microseconds per tuple qualification. The numbers generated above then, give a pessimistic outlook on performance since PDP-11/10 instruction times are in many cases 2 or 3 microseconds (times are dependent on memory cycles used, addressing mode, etc.) [DEC 75]. Therefore, even if a controller interaction consists of 40 or 50 instructions, the controller interaction to cell speed (tuple qualification) ratio will be less than 1.

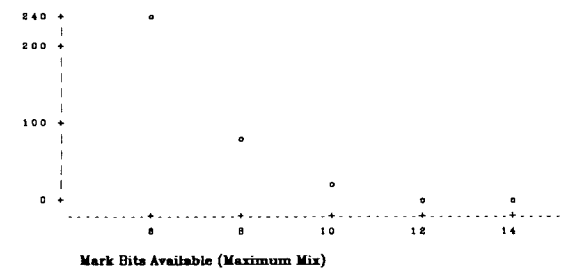
What all this leads to is that the proposed RAP organization and supporting operating system generate favourable performance characteristics without a noticeable amount of overhead. The reason for this, of course, is that the majority of the activity of the controller (dispatching, cell queue manipulation, etc.) is performed in parallel with cell operations.

Another test performed on the model involved variations in mark bit availability. The intent was to measure response time to queries as a function of multiprogramming level. An increase in the allowable maximum query mix was facilitated by a corresponding increase in mark bit availability. The following graphs illustrate the results.

Time in RAP Figure 4.



Time in Front-End Figure 5.

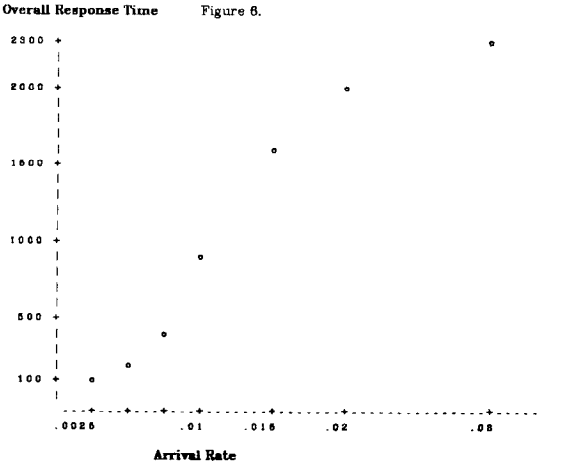


As expected, there is an increase in RAP time (Figure 4) and a corresponding decrease in the amount of time spent in the front-end computer (Figure 5). Both graphs approach an asymptote due to the fixed query stream (in particular, the arrival rate of that stream). The two curves balance each other out to yield a negligible difference in overall time. There is an observed increase in waiting time at the controller, but it is so small (much less than 1% of the overall time) as to make it insignificant.

The results then seem to indicate that an increase in the hardware availabilty of mark bits has no significant effect on performance. What should be noted, though, is that performance is *not degraded.* It is clearly advantageous to have a greater number of mark bits available so that a preemptive dispatching policy, such as that described in section 4.2, could be operative.

The ensuing test can really be thought of as part of the model validation. The purpose is to show performance fluctuations as a function of arrival rate. It should be clarified that since the characteristics of a real query stream are unknown, the arrival sequence is chosen to be a Poisson process. In the previous tests, the arrival rate was set to .01 time units (compared with 1 unit fixed cell speed). The mean interarrival time was thus 100 units. The accompanying graph (Figure 6) illustrates how overall response time changes when the arrival rate is altered.

Overall Response Time Figure 6.



As expected, the time in system (response time) increases dramatically with a corresponding increase in arrival rate. The response time approaches a limit, since we are looking at a fixed query stream. The worst possible situation would occur if all queries arrived at the same time. It should be made clear that the time spent in the RAP part of the system approaches a limit also, but much more quickly due to the imposed limit on the maximum multiprogramming level. An additional point to be made here is that although the overall time is increasing, the contention at the controller remains negligible.

All of the preceding tests support the value of the proposals. Although the partial preemptive policy discussed in section 4.2 was not modelled, RAP *was* modelled in terms of *cell at a time* access. Using the results in the model validation phase, we find that the time thus spent in the controller is less than 3% of the overall response time (28 units of time spent in the controller compared with an overall time of 912). *Relation at a time* access would of course yield slightly better results, but the 3% figure is certainly acceptable. This percentage is, of course, dependent on the particular query stream and data base employed.

There are some drawbacks and limitations to the simulation. First of all, the data bus between the RAP controller and the cells was not modelled. There were, however, two considerations made in this regard. First, the data bus can be considered to be an order of magnitude faster than the controller, and secondly, it can also be considered to be operative in full duplex mode. Both these factors suggest that it need not be part of the model.

Another feature which was not incorporated in the simulation model was mark bit binding. This is a task which must be performed by the front-end before allowing a query to proceed into the controller. Coupled with this is the fact that all front-end interactions were assigned one unit of time. Both these factors, however, only affect front-end time.

Finally, with regard to the query stream, no differentiation was made between individual RAP instructions. The justification for such a modelling decision is based on the consideration that a real query stream is unknown and specific knowledge about instructions of a given stream would not affect the predicted trends.

A multitude of additional tests could be undertaken in order to extend the performance evaluation accomplished to date. For example, tests could be run to measure performance fluctuations when different dispatching and/or high level scheduling policies are employed (one such possibility would be the preemptive dispatching discipline described in section 4.2).

6. Summary and Conclusions

RAP is a special purpose back-end processor which eliminates much of the computational load found in general data base management systems running on conventional architectures (in particular, the overhead associated with expensive mapping mechanisms and access paths). The RAP organization provides an associative search capability which is effectively independent of data base size, and an instruction set which is tailored for supporting operations on a relational data base.

It was shown that by spreading relations *across* RAP cells, as opposed to the current implementation where only one relation per cell is allowed, better overall performance can be achieved. A variety of situations were discussed in this context.

It was also shown that shifting RAP operations from *relation at a time* to *cell at a time* access would result in more consistent initial and mean response times to queries. There would, however, be a corresponding increase in overhead due to the necessity of maintaining separate instruction queues for each cell.

In order to support the high degree of concurrency, an executive for RAP was proposed. High level schedulers would reside in the front-end processor, while the dispatching of individual instructions and queue maintenance for each cell would be the responsibility of the controller. The proposed environment was modelled using simulation techniques. Although in many respects the simulation used rough approximations, the results indicate that the proposed configuration yields favourable results with little overhead and negligible contention at the controller.

Both the analytic and simulation results thus support the value of the proposals. There are a number of possibilites for future work based on the research described in this paper. One such possibility would be the use of deterministic scheduling techniques (in particular, precedence graphs) to determine optimal instruction dispatching at the RAP controller. Another area for further research involves investigating the possibility of a self-adjusting system. In such a scheme, relations would be redistributed on the RAP cells according to changing requirements of the query stream. This may provide more consistent response times in a dynamically changing environment.

7. References

[AC 75]
Astrahan, M. M., Chamberlin, D. D., *Implementation of a Structured English Query Language*, Communications of the ACM 18, 10, October 1975, pp. 580 - 587.

[AK 76]
Anderson, G. A., Kain, R. Y., *A Content-Addressed Memory Designed for Data Base Applications*, Proceedings of the 1976 International Conference on Parallel Processing, 1976, pp. 191 - 195.

[CD 73]
Coffman, E. G., Denning, P. J., *Operating Systems Theory*, Prentice-Hall, 1973.

[CHAN 76]
Chang, P. Y., *Parallel Processing and Data Driven Implementation of a Relational Data Base*, Proceedings of the 1976 Conference of the ACM, 1976, pp. 314 - 318.

[CLS 73]
Copeland, G. P., Lipovski, G. J., Su, S. Y. W., *The Architecture of CASSM: A Cellular System for Non-numeric Processing*, Proceedings of the First Annual Symposium on Computer Architecture, 1973, pp. 121 - 128.

[CMM 67]
Conway, R. W., Maxwell, W. L., Miller, L. W., *Theory of Scheduling*, Addison-Wesley, 1967.

[CODD 70]
Codd, E. F., *A Relational Model of Data for Large Shared Data Banks*, CACM 13, 6, 1970, pp. 377 - 387.

[COFF 76]
Coffman, E. G. (editor), *Computer and Job Shop Scheduling Theory*, John Wiley and Sons, Inc., 1976.

[DEC 75]
Digital Equipment Corporation, *PDP11 04/05/10/35/40/45 Processor Handbook*, 1975.

[DUFF 77]
Duff, T. S., *The Slogo Reference Manual Third Edition*, Dynamic Graphics Project Technical Memo DGP 3, Computer Systems Research Group, University of Toronto, 1977.

[FREE 77]
Freen, R., *A Partitioned Data Base for use with a Relational Associative Processor*, M.S. Thesis, University of Toronto, 1977.

[LSS 76]
Lin, C. S., Smith, D. C. P., Smith, J. M., *The Design of a Rotating Associative Memory for Relational Database Applications*, ACM Transactions on Database Systems, Vol. 1, 1, March 1976, pp. 53 - 65.

[MCGI 77]
McGill, M. J. (editor), *SIGIR-SIGARCH-SIGMOD Third Workshop on Computer Architecture for Non-Numeric Processing*, May, 1977.

[NAKA 76]
Nakano, R., *A Simulator for a RAP Virtual Memory System*, M.S. Thesis, University of Toronto, 1976.

[OS 77]
Ozkarahan, E. A., Sevcik, K. C., *Analysis of Archtectural Features for Enhancing the Performance of a Data Base Machine*, ACM Transactions on Database Systems, December 1977, pp. 297 - 316.

[OSSE 77]
Ozkarahan, E. A., Schuster, S. A., Sevcik, K. C., *Performance Evaluation of a Relational Associative Processor*, ACM Transactions on Database Systems, June 1977.

[OSSM 75]
Ozkarahan, E. A., Schuster, S. A., Smith, K. C., *RAP - An Associative Processor for Data Base Management*, AFIPS Conference Proceedings, Vol. 44, May 1975, pp. 379 - 387.

[SC 75]
Smith, J. M., Chang, P. Y., *Optimizing the Performance of a Relational Algebra Database Interface*, Communications of the ACM 18, 10, October 1975, pp. 568 - 579.

[SIBL 76]
Sibley, E. H. (editor), *Special Issue: Data-Base Management Systems*, ACM Computing Surveys, Vol. 8, 1, June, 1976.

[SNOS 77]
Schuster, S. A., Nguyen, H. B., Ozkarahan, E. A., Smith, K. C., *RAP.2 - An Associative Processor for Data Bases*, Proceedings of the 5th Annual Symposium on Computer Architecture, April, 1978.

[SOS 76]
Schuster, S. A., Ozkarahan, E. A., Smith, K. C., *A Virtual Memory System for a Relational Associative Processor*, AFIPS Coference Proceedings, Vol. 45, June 1976, pp. 855 - 862.

[STON 77]
Stone, H. S., *Multiprocessor Scheduling with the Aid of Network Flow Algorithms*, IEEE Transactions on Software Engineering, Vol. SE-3, 1, 1977, pp. 85 - 93.

[SU 75]
Su, Z., *Dynamic Scheduling with Preemption: A Deterministic Approach*, Ph.D. Thesis, University of Toronto, 1975.

[TJAD 76]
Tjaden, G. S., *Hierarchical Properties of Concurrency*, Proceedings of the 1976 International Conference on Parallel Processing, 1976, pp. 55 - 64.