PARALLEL PROCESSING OF NESTED RELATIONAL DATABASES

Saleh El-Shehaby* Mohamed S. Abougabal, Alaa Eldin Hafez, and Alaa S. Youssef
Dept. of Computer Science and Automatic Control
Faculty of Engineering, Alexandria University, EGYPT

ABSTRACT

This paper addresses the problem of parallel processing of nested relational algebraic operators. A modified scalable vertical partitioning technique, as well as a novel hybrid approach, that yields a high degree of parallelism, are proposed. Also, a heuristic approach, for devising parallel execution schemes that guarantee near optimum speedup, is introduced. It is validated versus the optimum speedup obtained using exhaustive search.

Keywords: *NINF relations, Nested relational algebra, Parallel processing.*

1. INTRODUCTION

Current trends in software, led to the appearance of the nested relational model in database systems [1,2,3,4,5], in order to support the new fields of non-traditional database applications. Also, the trend nowadays, in hardware, is towards parallel computer architectures. Recently, some work was done in the field of parallel processing of nested relational algebraic operators. A vertical partitioning technique for processing union and difference operations was introduced, as well as a horizontal technique for processing the remaining five primitive operators [6,7]. Moreover, a horizontal technique for processing union and difference operations was proposed in [8].

Although, only the union and difference operators were studied in [6,7], using the vertical partitioning technique, yet close study of the technique indicates that it is applicable to all nested relational algebraic operators. However, it is only useful if the number of available processors, p, is equal to the number of leaf-nodes, L, of the scheme tree representing the nested relations being processed. The technique does not apply if $p<L$, and the extra available processors are not utilized if $p>L$.

In the horizontal partitioning techniques proposed in [6,7,8], the extra available processors are wasted if $p > n$, where n is the number of tuples being processed. In case of binary operations, n is the maximum of the number of tuples of the two nested relations.

In this paper, the vertical partitioning technique of [6,7] is modified and presented in section 2. The modified technique is still applicable to all nested relational algebraic operators, and overcomes the scalability problem, mentioned above, when $p<L$.

In section 3, a novel hybrid partitioning approach is proposed. The approach employs the horizontal partitioning technique of [8], which is applicable to all binary operators, and the modified vertical partitioning technique of section 2. It is shown to yield higher degrees of parallelism, compared to those offered by existing partitioning methods.

The recursive partitioning of internal relations, using a heuristic approach, which employs the developed techniques of sections 2 and 3, as well as the horizontal technique [8], is proposed in section 4. This heuristic approach is validated, experimentally, and shown to be efficient in most practical situations. Finally, the paper is concluded in section 5.

---

*Dr. El-Shehaby is with the Medical Research Institute, University of Alexandria, EGYPT.

2. THE MODIFIED VERTICAL PARTITIONING TECHNIQUE

The existing vertical technique [6,7] is not applicable if the number of processors is less than the number of leaf-nodes, of the scheme tree, representing the nested relations under cosideration. In that technique, a manipulated tuple, is divided into a number of partitions. Each partition is a path from the root node to one leaf-node of the scheme tree, and one processor acts on each partition. After all processors complete execution, the resulting tuple is constructed by gathering together the partial outputs of the processors.

In this section, a new modified vertical technique is proposed. It is based on partitioning the external relation. The nested relation is partitioned into a group of nested relations, each having a scheme tree, that is a sub-tree of the original relation scheme tree. Each partition may have more than one leaf-node in this case. The natural join of this group of nested relations yields the original nested relation. The problem of scalability of the technique prevails when the number of processors, p, is less than the number of edges, b, emerging from the root node of the tree to be partitioned. b is normally less than the number of leaf-nodes, L. We propose to solve the problem in two steps.

Step-1: Each sub-tree, corresponding to an internal nested relation, that is an immediate son of the root node, is assigned a weight, W_i, proportional to the total sizes of nodes of the sub-tree, counted in disk blocks, and is given by: $W_i = \Sigma N_j \ \forall j$ such that node j is in subtree i.

Step-2: The heuristic algorithm, which is presented in Figure (1) after [9,Pp.107-110], is applied.

The correctness of the technique is established in [9,10]. It may be shown that the time complexity of the algorithm is in the order of $O(b \log_2 b + b \log_2 p)$ comparisons. The algorithm does not require accessing the tuples of the nested relation, and hence does not require any disk access. Its memory requirements are limited to the sizes of the input and output lists, declared in Figure (1), and are proportional to b and p, respectively.

This proposed vertical partitioning technique has the following advantages over the existing vertical technique proposed in [6,7].

1- The technique applies parallelism on the relation-level, and not on the tuple-level. This reduces the amount of synchronization overhead required. The reduction in synchronization overhead results in considerable gain of speedup, especially when the algorithm is implemented on shared nothing architectures.
2- The algorithm is scalable, and can be applied to any number of available processors, less than or equal to the number of leaf-nodes. Thus, eliminating the constraint imposed by the existing technique [6,7]. This allows for gradual increase in speedup with the increase in number of available processors. The importance of this feature is evident in the vertical partitioning phase of the hybrid approach presented in the next section.

This approach is useful for $p \leq b$. If $p > b$, then the approach of the next section is applied to increase the degree of parallelism.

3. THE NOVEL HYBRID PARTITIONING APPROACH

It is pointed in the above sections that the maximum number of utilized processors is b in the case of vertical partitioning and n in the case of horizontal partitioning. Therefore, if a hybrid approach is devised, which either applies horizontal partitioning, proposed in [8], followed by verticaal partitioning, of section 2 (this is referred to as *HV* partitioning), or vice versa (VH partitioning), then the maximum number of usable processors could be nb. The advantages of each of these two partitioning methods will be demonstrated.

3.1. HV Partitioning

In this situation, the nested relation is first partitioned to n horizontal partitions. The remaining available processors are equally allocated for vertical partitioning of each horizontal partition (nested tuple in this case). The scalability feature of the proposed technique of section 2 is useful in cases where $n < p < nb$.

Algorithm Pack (IL, p, OL)
Input: 1- An Input List, IL, of tasks. Each entry corresponds to one internal relation, that is an immediate son of the external relation. Associated with each entry is its weight W_i.
2- The number of available processors p.
Output: An Output list, OL. Each entry corresponds to one processor and is composed of a list of tasks (internal relations) assigned to that processor, as well as the total weight assigned to the processor.
Begin
Sort IL in descending order of weights;
Initialize the total weight assigned to each processor to Zero;
While (task list IL is not empty) do
begin
Get the first task from the task list IL;
Search the processor list OL for the first processor with minimum total weight;
Add this task to the chosen processor list and increment that processor's total weight by this task weight;
Remove this task from the task list IL;
end;
End.

Figure 1. Algorithm for scalable vertical partitioning

Algorithm Distribute (IL, P, OL)
Input: 1- An Input List, IL, of b tasks. Each entry corresponds to one vertical partition of the tree. Associated with each entry is its weight W_i.
2- The number of available processors P such that
P > b.
Output: An Output list of integer values, OL. Each entry corresponds to one vertical partition, and is equal to the number of processors assigned to that partition.
Begin
Assign one processor to each vertical partition;
New_P := P - b;
Get the total sum of weights T;
For each vertical partition i do
begin
Calculate the percent weight of i as
$percent\_weight_i := W_i / T$;
multiply $percent\_weight_i$ by New_P to get the real number of processors for that partition $Real\_P_i$;
Increment the number of processors assigned to partition i by the floor of $Real\_P_i$;
end;
Get the remaining number of processors R;
Sort IL in descending order of fraction part of percent weights;
Assign the R remaining processors to the first R vertical partitions in the sorted list IL;
End.

Figure 2. Algorithm to distribute available processors over vertical partitions

3.2. *VH Partitioning*

In this situation, the nested relation is first partitioned vertically, to b partitions. If p=nb, then each vertical partition is partitioned to n horizontal partitions. Otherwise, when $b < p < nb$, we propose the following algorithm which determines the number of processors to be assigned to each vertical partition of the scheme tree.

Step-1: Each vertical partition is assigned a weight that is proportional to the total sizes of nodes of the sub-tree corresponding to the partition, counted in disk blocks. $W_i = \Sigma N_j \quad \forall j$ such that node j is in subtree i.

Step-2: The algorithm, which is presented in Figure (2), is applied.

The correctness of the technique is established in [10]. It may be shown that the time complexity of the algorithm is in the order of $O(b \log_2 b)$ comparisons. The algorithm does not require accessing the tuples of the nested relation, and hence does not require any disk access. Its memory requirements are limited to the sizes of the input and output lists, declared in Figure (2), and are proportional to b and p, respectively.
This hybrid approach as explained in sections 3.1 and 3.2 utilizes up to nb processors. However, if p>nb then the extra processors may be utilized by partitioning of internal relations, as proposed in section 4 below.

4.PARTITIONING OF INTERNAL RELATIONS

This section investigates the extension of the proposed partitioning techniques, in the availability of more than nb processors. The main idea is to make use of the recursive nature of nested relations and nested relational algebraic operators.
The definition of a nested relation is recursive. The external relation R is defined as a set of attributes, of which some are non-atomic. These non-atomic attributes are nested relations themselves. A corresponding property, in the definitions of nested relational algebraic operators, is that most of the operators are of recursive nature. For example, the application of the extended union operator on R involves the recursive union of all internal relations of R.
An algorithm, employing recursive partitioning of internal relations, for parallel processing of a nested relational algebraic operator, <op>, is depicted in Figure (3). An important remark should be noted about this general algorithm. The decision that is marked by an * , which is choosing the appropriate partitioning technique to be applied to a certain external/internal relation, is a difficult and sensitive problem by itself. This problem was thoroughly investigated analytically for a two level scheme tree, and heuristically for any general scheme tree.

Algorithm <op> (R_1, R_2, R_3)
Input: Two nested relations R_1 and R_2.
Output: The resulting nested relation R_3.
Begin
  Partition R_1 and R_2 using a suitable method* ;
  Cobegin
  for each 2 corresponding partitions $PR1_i$ and $PR2_i$
    do begin
    operate on root ;
    for each 2 corresponding sub-relations $SR1_i$ & $SR2_i$
      do <op> ($SR1_i$, $SR2_i$, $SR3_i$) ;
    end;
  Coend;
  Merge resulting partitions $PR3_i$ if necessary;
End.

Figure 3. Skeleton of parallel algorithm for processing <op>.

4.1. *Analytical Analysis For Two-Level Trees*

Nested relations with one level of nesting were considered, in order to find analytically the partitioning methods, that yield minimum time complexity, for the parallel processing of an operator, with a certain recursive cost function. The obtained results are useful in situations where the nested relational model consists of one level of nesting.

This appears in many practical situations. The approximated results are summarized in Table (1), and are used as heuristics in sections 4.2 and 4.3. Detailed proofs are provided in [10, Pp.85-120]. The used symbols: H, V, HV, and VH, denote horizontal, vertical, horizontal followed by vertical, and vertical followed by horizontal, partitioning, respectively.

4.2. *A Heuristic Approach For Any General Tree*

The cost of sequential processing of any nested relational algebraic operator, was shown to be in the order of one of the five generic cost functions depicted in Table (1) [3]. These cost functions represent the number of disk block accesses. N is the number of disk blocks containing the atomic attributes of an external/internal relation. Three cases were studied for each of the five generic cost functions: no split cost, sorted tuples, and unsorted tuples.

Table 1. Heuristics (approximated analytical results).

Cost Function	No. of Processors	No Split Cost	Sorted Tuples	Unsorted Tuples
N log N	$p \le b$	H	H	V
	$b < p \le n_1$	H	H	V
	$n_1 < p \le n_1 b$	H	HV	VH
Log N	$p \le b$	H	V	V
	$b < p \le n_1$	H	H	V
	$n_1 < p \le n_1 b$	HV	HV	V
N	$p \le b$	H	V	V
	$b < p \le n_1$	H	H	V
	$n_1 < p \le n_1 b$	HV	HV	V
N^2	$p \le b$	H	H	H
	$b < p \le n_1$	H	H	H
	$n_1 < p \le n_1 b$	H	HV	HV
Constant	$p \le b$	V or H	V	V
	$b < p \le n_1$	H	H	V
	$n_1 < p \le n_1 b$	HV	HV	V

Case 1: No Split Cost

No partitioning overhead is considered. This occurs when the nested relation is in the required partitioned form, or when the operator is applied on a relation that is internal in the query tree (i.e. not a leaf relation) so it is resulting from a previous operation and is in the required partitioned form. Also, for some operators, the partitioning process is trivial. For example, horizontal partitioning in the case of extended projection, is just a matter of assigning equal numbers of tuples to each processor. Thus, no partitioning overhead is accounted for.

Case 2: Sorted Tuples

Split cost (partitioning overhead) is accounted for, and the tuples of the nested relation are sorted.

Case 3: Unsorted Tuples

Split cost is accounted for, and the tuples of the nested relation are not sorted. A distinction between this case and case 2 is made, because the horizontal partitioning cost is sensitive to whether the tuples are sorted or not. Consequently, the hybrid partitioning cost is affected by the state of tuples, if sorted or not, too.

Cases 2 and 3 arise when the relation is a leaf relation in the query tree, or when it is partitioned in a form, that is not the optimal for the given cost function, i.e. re-partitioning is required.

The results of Table (1) were used as heuristics, and were validated for different classes of scheme trees, which represent shallow, balanced, and deep trees. This is detailed in the following section.

4.3. *Experimental Validation Of The Heuristic Approach*

The heuristic approach, summarized in Table (1), was validated as follows. Experiments using nested relations with different scheme trees, that depict a wide nature of scheme trees encountered in real situations, were conducted. These schemes are depicted in Figure (5). The input to an experiment is a scheme tree, with all related information of sizes and numbers of tuples in each node, and the recursive cost function of the operator. The

experiment was repeated for each of the three cases which affect the partitioning overhead, and were mentioned in the previous section. During an experiment, the number of available processors was made to vary, in steps, from 1 to 1000. For each step, the costs of two parallel execution schemes were computed. The first is the cost of the heuristic scheme, and the second is that of the optimum. The optimum scheme is obtained by exhaustively examining all possible partitioning methods at each node, and selecting the combination that yields the minimum cost. The cost measure was the number of disk blocks accessed by the longest task. We call this a comparison point. About 18,000 comparison points were examined for each of the scheme trees of Figure (4). The speedup achieved by each of the two execution schemes is computed by the following equation [11,12].

$$\text{Speedup} = \frac{\text{Sequential processing cost}}{\text{Parallel processing cost}} . \quad (1)$$

The absolute error in speedup, E_a, is obtained by subtracting the speedup obtained by the heuristic approach, S_H, from the speedup obtained by the optimum approach, S_O. The relative error in speedup, E_r, is computed by dividing the absolute error by the heuristic speedup.

$$E_a = S_O - S_H; \quad (2)$$

and,

$$E_r = \frac{E_a}{S_H} = \frac{T_H - T_O}{T_O}, \quad (3)$$

where

T_O is the time complexity of execution of the optimum scheme;

and,

T_H is the time complexity of execution of the heuristic scheme.

For each set of nested relations with the same scheme tree, average speedup factors obtained by the optimum and heuristic methods, were calculated, for different numbers of available processors. The standard deviation, corresponding to each average value, was calculated and found to be low. Average relative errors, in speedup, for different cases were computed. In addition, corresponding to each average relative error figure, a figure named **0.1-P** was computed. It indicates the percentage of cases that yielded a relative error, in speedup, less than or equal to 0.1 .

Table 2. Performance Results Summary.

Class of nested relations (fig.4)	Overall average E_r (relative error)	% of cases with $E_r \leq 0.1$ (0.1-P)
Class a	0.04	90.4%
Class b	0.14	79.9%
Class c	0.82	66.9%
Class d	0.14	80.0%
Class e	0.50	75.0%
Class f	7.00	63.0%

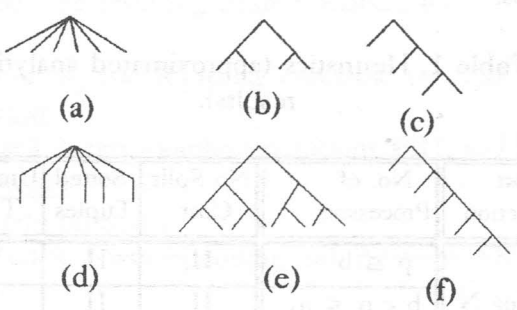

Figure 4. Classes of scheme trees of examined nested relations.

$$0.1\text{-}P = 100 * = \frac{\text{number of cases} \in \text{which } E_r \leq 0.1}{\text{total number ofcases}}$$

or, $0.1\text{-}P = 100 * \text{Prob}(E_r \leq 0.1)$. (4)

The results are summarized in Table (2) (refer to [10] for detailed results). Shallow trees yielded best performance. Deep trees showed the worst performance, as their nature favors horizontal partitioning. In cases when the heuristics imply other partitioning techniques, their performance degenerates. The results showed, also, that increasing the number of nested levels, implies a moderate degradation in the performance of the heuristic approach. This is expected and was found

to be tolerable.

Close study of the results reveals the fact that the heuristics of table 1 are valid in most practical situations.

5. CONCLUSION

In this paper, a new modified technique for vertical partitioning of nested relations, for parallel processing of algebraic operators, was introduced. In addition, a novel hybrid approach, combining the introduced vertical and the existing horizontal partitioning techniques, was suggested. The approach that was taken is independent of the underlying processors interconnection topology. Advantages of the novel techniques over the existing ones, were demonstrated. The novel vertical technique is scalable, and allows for gradual increase in speedup with the increase in number of available processors, in contrast to the existing vertical technique that works on a fixed number of processors. The hybrid approach yields higher degrees of parallelism, not achievable by the vertical nor the horizontal approaches, by allowing for more data partitioning, in the availability of more processors. This, in turn, yields higher speedup factors in execution times.

The problem of partitioning of internal relations, in addition to the external relation, was also addressed. Deciding which partitioning methods to be applied to the external/internal relations, in order to maximize the speedup gained, using the available number of processors, appeared to be a complex problem. This issue was studied thoroughly, and a heuristic approach was devised and validated. It was shown that these heuristics could be applied effectively, to partition any nested relation, using any of the existing horizontal or proposed vertical or hybrid techniques, in order to process any nested relational algebraic operator.

REFERENCES

[1] Chen, Q., and Kambayashi, Y., "Nested relation based database knowledge representation", ACM SIGMOD conference, Denver, Colorado, June 1991.

[2] Tansel, A.U., and Garnett, L., "Nested historical relations", ACM SIGMOD conference, Portland, Oregon, June 1989.

[3] Hafez, A.M., "Storage Model for nested relations", PH.D. Case western reserve university, August 1989.

[4] Colby, L.S., "A recursive algebra and query optimization for nested relations", Technical report, 1989, Indiana university.

[5] Roth, M.A., Korth, H.F., and Silberschatz, A., "Extended algebra and calculus for nested relational databases", ACM transactions on Database Systems, Vol.13, No.4, December 1988.

[6] Mahmoud, H.A., " Towards parallel nested relational algebra", M.Sc. thesis, Comp. Sc. & Auto. Control Dept., Fac. of Engineering, Alex. university, Egypt, 1991.

[7] Abougabal, M.S., and Mahmoud, H.A. " Towards parallel nested relational algebra", Second IASTED International Conference, Alexandria, Egypt, May 1992.

[8] Hafez, A.M., "Parallel algorithms for union and difference for non first normal form relations", Second IASTED International Conference, Alexandria, Egypt, May 1992.

[9] Goodman, H., Introduction to the design and analysis of algorithms, McGraw Hill Inc., 1977.

[10] Youssef, A.S., " Parallel processing of nested relational databases", M.Sc. thesis, Comp. Sc. & Auto. Control Dept., Fac. of Engineering, Alex. university, Egypt, 1994.

[11] Polychronopoulus, C.D., and Banerjee, U., "Processor allocation for horizontal and vertical parallelism and related speedup bounds", IEEE Transactions on computers, Vol.C-36, No.4, April 1987.

[12] Madala, S., and Sinclair, J.B., "Performance of synchronous parallel algorithms with regular structures", IEEE Trans. on parallel and distributed systems, Vol.2, No.1, January 1991.