A Parallel Algorithm for Generating Maximal Interval Groups in Interval Databases Based on Schedule of Event Points

Nabil Arman
Associate Professor of Computer Science
Palestine Polytechnic University, Palestine

ABSTRACT

Interval databases queries are computationally intensive and lend themselves naturally to parallelization to speed up the solution of such queries. In this paper, a parallel algorithm to generate all maximal interval groups form a given interval set is presented. The algorithm makes use of intraoperation parallelism to speed up the generation of the maximal groups. The development of efficient algorithms to enumerate all intervals that have certain properties has attracted a large amount of research efforts due to the important role of interval-based reasoning in different areas like rule-based systems, including Expert Systems (ESs), Information Distribution Systems (IDSs), and database systems to name a few. These algorithms are very crucial to answer certain queries about these intervals.

Key Words: Maximal interval groups, interval database, parallel databases.

1. Introduction

Interval-based reasoning has an important role in many areas like rule-based systems, including Expert Systems (ESs), Information Distribution Systems (IDSs), and database systems. Intervals are appropriate and convenient for representing events that span continuous period of time. One may query an interval database to determine what events occur during a given interval. Algorithms to enumerate all intervals that have certain properties have attracted a large amount of research efforts due to the important role of interval-based reasoning in different areas, including rule-based systems and database systems [1,2,3,4,5]. These algorithms have an important role in all these systems. An algorithm that finds an interval in an interval tree, represented as a red-black tree, which overlaps a given interval is presented in [4]. However, the algorithm has the overhead of building and maintaining the interval tree and it can only determine pairs of intervals that overlap. Our algorithm, on the other hand, determines all interval groups that overlap and makes use of intraoperation parallelism to speed up processing.

Many queries in interval databases, including the generation of maximal interval groups, have data requirements that may run into terabytes. Handling such large volumes of data at an acceptable rate is difficult, if not impossible, using single-processor systems. In fact, a set of commercial parallel database systems, such as Teradata DBC series of computers have demonstrated the feasibility of parallel database queries. As a matter of fact, the set-oriented nature of database queries naturally lends itself to parallelization [6]. In a database of *n* intervals, there is a need to find all maximal groups, where each group has the intervals that overlap. In a temporal database that stores all courses classes and their times, a query may be asked to generate all groups of classes that meet at a certain time point. In an IDS, it is always needed to check the time validity of rules to determine if they can be chained. This has an important role in controlling the operation of an IDS which is a corner stone of Command, Control, Communication, Computer, and Intelligence (C4I) systems. This paper presents a parallel algorithm to generate all maximal interval groups form a given interval set.

2. Interval Grouping Parallel Algorithm

The generation of the maximal interval groups in interval databases can be parallelized using intraoperation parallelism. The processing of this query can be speeded up by parallelizing the execution of many individual operations involved in the generation of the maximal interval groups. To simplify the explanation and presentation of the algorithm, it is assumed that there are *n* processors, $P_1, \ldots, P_n$, and *n* disks $D_1, \ldots, D_n$, where disk D_i is associated with processor P_i.

A serial algorithm that generates the maximal interval groups was presented in [7]. The algorithm doesn't make use of the benefits of parallel architectures which are becoming more popular for query processing in large databases.

Before explaining the grouping algorithm, some concepts that will be used in the algorithm are explained. The algorithm uses the concept of event points and event point schedule [1]. An *event point* is a point on the spatial dimension, where some intervals are leaving a certain interval group and other intervals are entering another interval group. The set of these event points constitutes a *schedule of event points*. In our algorithm, the real schedule is determined dynamically as the algorithm progresses. The algorithm uses intervals where an interval $I = [t_1, t_2]$ is represented as an object with fields $low[I] = t_1$ (the low endpoint) and $high[I] = t_2$ (the high endpoint). Two intervals overlap if their intersection is not empty. The algorithm also sorts the intervals in *Lexicographic Ordering*. This can be performed using a parallel sort algorithm like range-partitioning sort or parallel external sort-merge[6]. An interval set is sorted in lexicographic ordering if whenever interval $[i,j] < [h,k]$ then either $i < h$ or $i=h$ and $j < k$. Let *IS* denote an interval set and let $t_1, t_2, \ldots, t_m$ denote all potential event points. Let t_{m+1} be *high[last_interval]*, which is an event point representing a guard condition for the algorithm. Let $LIG(t_i)$ denote the Low Interval Group of t_i, which is the set of intervals *I* whose $high[I] >= t_i$ and $low[I] < t_i$. Let $UIG(t_i)$ denote the Upper Interval Group of t_i, which is the set of intervals *I* whose $low[I] < t_{i+1}$ and $low[I] >= t_i$, where t_{i+1} is the next event point. Then for every event point t_i and its next event point t_{i+1},

$$IG(t_i) = LIG(t_i) \cup UIG(t_i)$$

Thus, $IG(t_i)$ for event point t_i consists of the set of intervals whose $high[I] >= t_i$ and $low[I] < t_i$, and the set of intervals whose $low[I] < t_{i+1}$ and $low[I] >= t_i$, where t_{i+1} is the next event point of t_i.

The grouping algorithm is implemented by the procedure Parallel_Determine_Interval_Groups as given in Figure 1, which can be invoked with any interval set *IS* to be grouped into maximal groups, such that each interval group *IG* has the maximum number of intervals such that for any interval I_1, and I_2 in *IG*, $I_1 \cap I_2 \neq \phi$. Parallel_Determine_Interval_Groups Algorithm determines all potential event points that represent the set of all distinct low endpoints in the interval set. In doing that, the interval set *IS* is partitioned and allocated to the *n* processors using range partitioning on the intervals' low end points. Each processor P_i determines all potential event points in its partition locally. The results form processors $P_1, \ldots, P_n$ are merged together to form *event_points.*

The interval set *IS* and *event_points* are replicated on all *n* processors to be used in computing the maximal interval groups. The event points are distributed on the *n* processors in a round-robin scheme, where each processor P_i determines maximal interval group G_i based on its assigned event point locally. If the number of event

```
Procedure Parallel_Determine_Interval_Groups(Interval Set: IS)
{
        Partition IS using range partitioning on interval low end points
        Sort IS in lexicographic ordering using parallel external sort-merge
        $P_i$ determines all potential event points in its partition locally
        Merge results from $P_1, \ldots, P_n$ to form event_points
        Replicate IS and event_points on all n processors
        Distribute event_points on the processors in a round-robin scheme
        $P_i$ determines maximal interval group $G_i$ using its assigned event point locally
        Merge $IG_i$s from $P_1, \ldots, P_n$ to produce the final result
}
```

Figure 1. Parallel_Determine_Interval_Groups Algorithm

points m is less than the number of processors n, then m processors are used. The maximal interval groups from processors $P_1, \ldots, P_n$ are merged to produce the final result

Example: Consider the interval set: {[0,1],[0,3],[0,5],[0,7],[0,9],[0,11],[2,13],[4,13]} and assume there are 4 processors P_1,P_2, P_3, and P_4.

The algorithm sorts the interval set if it is not sorted using a parallel sort algorithm. The algorithm then partitions the interval set using range partitioning. Assume the partition vector is <1,2,3>. Based on this vector, intervals whose low end point is less than 1 are placed on D_1. Intervals whose low end points are greater than or equal to 1 and less than 2 are placed on D_2. Intervals whose low end points are greater than or equal to 2 and less than 3 are placed on D_3. Finally, intervals whose low end points are greater than 3 are placed on D_4. Thus, *IS* is distributed as follows:

D_1 contains [0,1], [0,3], [0,5], [0,7], [0,9], and [0,11]
D_2 contains no intervals
D_3 contains [2,13]
D_4 contains [4,13]

Therefore, the processors determine the event points in parallel as follows:

P_1 determines event point: 0
P_2 determines no event point
P_3 determines event point: 2
P_4 determines event point: 4

The event pointes from $P_1, \ldots, P_n$ are merged to produce the event points 0,2, and 4.

After replicating *IS* and *event_points* on all *n* processors, the algorithm distributes the event points on the *n* processors in a round robin scheme. Therefore,

P_1 is assigned event point 0
P_2 is assigned event point 2
P_3 is assigned event point 4
P_4 is not assigned any event point and is free to be used if needed

The processors determine maximal interval groups based on the event points as follows:

(1) Event point t_j =0.
    Next event point t_i =2 determined from the event points
    P_1 determines the maximal group $IG(t_j=0)$ = {[0,1],[0,3],[0,5],[0,7],[0,9],[0,11]}
(2) Event point t_j =2.
    Next event point t_i =4 determined from the event points

P_2 determines the maximal group $IG(t_j=2)$= {[0,5],[0,7],[0,9],[0,11],[2,13]}

(3) Event point t_j =4.

Next event point t_i =Null since 4 is the last event point

P_3 determines the maximal group $IG(t_j=4)$= {[0,5],[0,7],[0,9],[0,11],[2,13],[4,13]}

The maximal groups from the processors are merged to produce the final result.

3. Conclusion

A parallel algorithm for generating the maximal interval groups has been presented. The algorithm is very crucial to answer certain queries about the intervals in an interval set. The algorithm can be used to generate the maximal interval groups needed in many systems, including IDS, expert systems, and temporal database systems. The algorithm makes us of intraoperation parallelism to speed up the generation of the maximal interval groups in an interval database.

References:

[1] A. Aiken, J. Hellerstein, and J. Widom, Static Analysis Techniques for Predicting the Behavior of Active Database Rules, ACM Transactions on Database Systems, Vol. 20, No. 1, 1995, pp. 3-41.

[2] S. Chamberlain, Automated Information Distribution in Bandwidth-Constrained Environments, 1994 IEEE MILCOM Conference Record, Vol. 2, October, 1994.

[3] S. Chinn and G. Madey, A Framework for Developing and Evaluating Expert Systems for Temporal Business Applications, Expert Systems With Applications. 1997.

[4] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms, McGraw-Hill Book Company, 2001.

[5] J. Harrison, Active Rules in Deductive Databases, ACM CIKM, Washington D.C., USA, November, 1993.

[6] A. Silberschatz, H. Korth, and S. Sudarshan, Database System Concepts, McGraw Hill, 2005

[7] N. Arman, "An Efficient Algorithm for Generating Maximal Interval Groups in Interval Databases," J. J. Appl. Sci., Vol. 6, No.1, 2004, pp. 19-27.