1986 15th International Conference on Parallel Processing, August 19-22, 1986, St. Charles, Illinois, USA

A PARALLEL RANGE SEARCH ALGORITHM USING MULTIPLE ATTRIBUTE TREE

S.V. Nageswara Rao, *S.S. Iyengar*

Department of Computer Science
Louisiana State University
Baton Rouge, LA 70803, USA

R.L. Kashyap

Department of Electrical Engineering
Purdue University,
West lafayette,IN 47907, USA

ABSTRACT

The problem of range search arises in many applications in areas such as information retrieval, database, robotics and computational geometry. There are a good number of sequential algorithms for this problem based on data structures such as k-d tree, quad tree, range tree, d-fold tree, super B-tree, overlapping k-ranges, non-overlapping k-ranges, etc. In this paper we present a parallel algorithm for a Single Instruction Multiple Data (SIMD) computing system with p processing elements. We make use of the linearized multiple attribute tree as the underlying data structure. Our algorithm has the complexity of $O(kN/p)$, $p \leq N$ where k is the dimensionality and N is the number of points of the data space.

0. INTRODUCTION

The problem of range search arises in many application areas such as information retrieval, database, robotics, and computational geometry. There have been many sequential algorithms for range search problem using the data structures such as k-d tree, quad tree, range tree, overlapping k-ranges, nonoverlapping k-ranges, d-fold tree, super B-tree, k-fold tree, and multiple attribute tree [1,3,5-7].

In this paper, we present a parallel algorithm for the range search problem. We make use of the Multiple Attribute Tree (MAT) as the underlying data structure. Our model of computation is a *Single Instruction Multiple Data* (SIMD) computing system, consisting of p processing elements. The system has a single shared memory that supports simultaneous reads. The time complexity of our algorithm is given by $O(kN/p)$, $p \leq N$.

The organization of this paper is as follows: In Section 1, we discuss the MAT data structure and the corresponding directory. In Section 2 the basic range search algorithm is discussed, and its complexity is estimated. Firstly an $O(k)$ algorithm is obtained using a processor array of size N in a straight forward manner. Then, an $O(kN/p)$, $p \leq N$ algorithm using a processor array of size p and an augmented directory.

1. MULTIPLE ATTRIBUTE TREE AND LINEARIZATION

The MAT data structure was first introduced and analysed by Kashyap et al [2]. The MAT is shown to outperform the inverted file structure for partial match and complete match queries in the cases when the directory resides on the main and secondary memories in [4]. An exhaustive treatment on various structural aspects of MAT can be found in [5].

The k-dimensional MAT on k attributes $A_2, A_3, \ldots, A_k$ for a set of records is defined as a tree of depth k, with the following properties [4]:

i) it has a root at level 0,

ii) each child of the root is a (k-1)-dimensional MAT, on (k-1) attributes, $A_1, A_2, \ldots, A_k$, for the subset of the records that have the same A_1 value. This value is the value of the root of the corresponding (k-1)-dimensional MAT, and

iii) the child nodes of the root are in the ascending order of their values. This set of child nodes is called the *filial-set*. □

From the above definition we note that there is a root node at level 0, and it does not have any value. Every node of level $i, i=1,2,\ldots,k$, corresponds to a value of the attribute A_i. Fig 1(b) shows the MAT data structure for the sorted set of records of Fig.1(a). The attributes $A_1, A_2, \ldots, A_k$ form the hierarchy of of levels 1 through k. The nodes of every filial-set are ordered according to their values. Another important property is that each record or point is represented by a unique path from the root to the corresponding terminal node. The total number of nodes in the MAT is atmost kN for a given data set containing N records or points.

The MAT is *linearized* and stored as a directory. We make use of the *breadth-first* linearization in which the nodes are stored in the order they are encountered when MAT is traversed in a breadth-first manner. The directory is an array of M ($\leq kN$) directory elements and each directory element has the following fields:

A_1	A_2	A_3	A_4	Record pointer
1	1	3	3	1
1	2	1	2	2
1	1	4	1	3
1	1	2	6	4
2	3	5	7	5
1	1	3	5	6
1	2	5	6	7
2	3	5	1	8

FIG. 1(a) INPUT DATA FILE

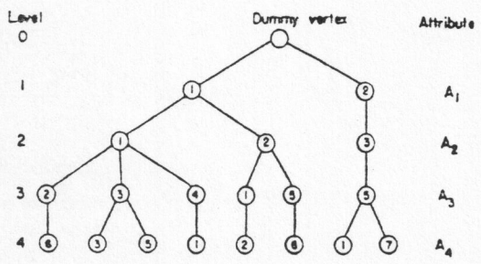

FIG. 1(b) MAT representation for data of FIG.1(a)

directory-element = **record**
value: 1..M;
first-child: 1..M;
last-child: 1..M;
end;

where the fields corresponding to a node T, at level j, and numbered n are defined as follows:

value: the value of the node T;
first-child: node number of the first child of the child set of the node T;
last-child: node number of the last child of the child set of the node T;

The idea of breadth-first linearization is illustrated in Fig.2 and the correponding directory is shown in Fig.3. In the first-child field of leaf nodes contain the pointers to the corresponding records. The time complexity of constructing this directory using a uniprocessor system is $O(N\ logN)$ (including the time required to sort the records on all the attributes) [5]. Here, we are concerned only with answering the range query, assuming that the directory is available in memory. This directory is utilised in the design of parallel range search algorithms in the next section.

2. PARALLEL RANGE SEARCH ALGORITHM

A range query is given by $Q = \bigcap_{i=1}^{k} q_i$, where q_i specifies the range $[l_i,h_i]$ at the level i. In geometric terms a range query specifies a *rectilinearly oriented hyper rectangle* in k dimensional space. Answering a range query calls for the retrieval of the points enclosed by the specified hyper rectangle, and these points form a 'sub' MAT, called the *query* MAT (QMAT), on the original MAT. These nodes of the QMAT are called the *qualified* nodes of MAT for the query Q. Answering a range query involves identifying the nodes of the QMAT on the MAT. Let R be the number of records contained in the hyper rectangle. The number of the nodes of the QMAT is no more than kR.

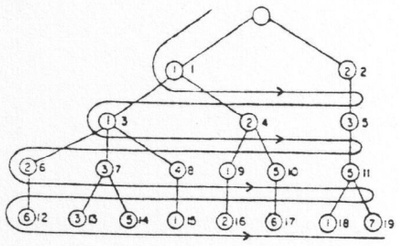

FIG. 2 Breadth-first linearization

An algorithm for range query proceeds by descending down the MAT level by level, starting from the first level. At each level i, the child sets of qualified nodes of previous level are searched for the containment in the range $[l_i,h_i]$. The algorithms is as follows:

---

algorithm RANGE-SEARCH(level,qnodes);
begin
1. tempset:= ϕ;
2. **for** each $n \in$ qnodes **do**
3. add to tempset all the child nodes of n that lie in the range $[l_{level},h_{level}]$;
4. **if** level < k
5. **then** RANGE-SEARCH(level+1,tempset)
6. **else** return the pointers given by the elements of tempset;

end

In the algorithm RANGE-SEARCH, at a level i, all the nodes that satisfy the partial query $\bigcap_{j=1}^{i} q_j$ are collected as tempset. In the next level $i+1$, the child nodes of these nodes are tested for inclusion in the range $[l_{i+1},h_{i+1}]$. At the final level, the pointers to the information about the records that satisfy the given query are retrieved.

The MAT data structure has some inbuilt parallelism with respect to answering a range query. The searching for the qualified nodes can be simultaneously carried out on the sub MATs of the same level. However, there seems to be a stringent sequentiality in the way the attributes are processed one after the other. Again, speaking in geometric terms, processing of each atttribute reduces the dimensionality of the search space by one. From the above discussion, we conclude that $O(k)$ is lower bound on the steps involved in answering a range query on the MAT-based approach.

Node-number	Value	First-child	Last-child
1	1	3	4
2	2	5	6
3	1	8	8
4	2	9	10
5	3	11	11
6	2	12	12
7	3	13	14
8	4	15	15
9	1	16	16
10	5	17	17
11	6	18	19
12	6	4	–
13	3	1	–
14	5	6	–
15	1	2	–
16	2	2	–
17	6	7	–
18	1	8	–
19	7	5	–

FIG 3 Breadth-first directory

The model of computation is in the form of an array of processor elements PE_j, j=1,2,...p, and operates in *Single Instruction and Multiple Data* (SIMD) mode. More specifically, each processor element executes the same algorithm in synchronism with all the other processor elements. We use single shared memory in which the breadth-first MAT directory and other variables are stored. As will be shown later there will be no conflicts in writing into the memory. But, simultaneous read operations are supported. The range query is represented by low-limit[i] and high-limit[i], i=1,2,...,k, which give the lower and upper limits of the range for the attribute A_i.

Firstly, consider the processor array PE_i, i=1,2,...N where N is the number of records in input set. The array base[i], i=1,2,...,(k+1) is stored in the memory, where any node of level i lies in between the entries indexed by base[i], and base[i+1]-1 in the directory. We use the array enable[i], i=1,2,...N to selectively enable and disable the processor elements of the processor array. The algorithm consists of k steps and in step i, i=1,2,...,k the level i is processed to obtain the qualified nodes of the level i. Each enabled processing element acts on a child node of a qualified node at previous level, and checks if the child node satisfies the range constraint of the current level. The processor elements corresponding to the qualified nodes of current level are enabled in the next step. For the final level, the points that satisfy the range query are obtained. The algorithm executed by the processor element PE_j, j=1,2,...N for the step i, i=1,2,...k is as follows:

```
algorithm PE_j;
begin
1.   if enable[j]
2.   then
     begin
3.      enable[j] = false;
4.      n = base[i] + j;
5.      if low-limit[i] ≤ value[k] ≤ high-limit[i]
6.      then
        begin
7.         for l = first-child[n] to last-child[n] do
8.            enable[l - base[i]] = true;
        end;
     end;
end;
```

In the above algorithm the qualified records have to be returned in the final level by suitably modifying the lines 7 and 8. This parallel algorithm implements the algorithm RANGE-QUERY and it is very easily seen that this correctly answers the range query. Since the algorithm is executed in k synchronous steps, it is evident that the time complexity of this algorithm is $O(k)$.

In applications involving large amounts of data, the value of N could be very large and the assumption of the array of N processors is not very pragmatic. We now present an algorithm that utilizes an array PE_i, i=1,2,...,p, $p \leq N$ processor elements. The basic idea of 'processing the MAT nodes level by level' is still followed; at any level the processor array is invoked required number of times along breadth of the MAT. In this case the directory is augmented with two more fields - enable and level. The former is used to selectively enable and disable the processor elements and latter field gives the level of the node. The global variable current-level gives the level-number that is currently being processed, and it is incremented after each level is processed. The array index[i], i=1,2,...,p gives the current index (into the directory) to be used by the processor element PE_i. Initially, the enable field is made true for all the nodes of level 1 and also the entries of the array index corresponding to the first level nodes are filled up. For the final level, the qualified records are retrieved. The algorithm executed by the processor element PE_j, j=1,2,...,p is as follows:

The if statement in line 2 makes sure that all processors of the SIMD array process the nodes of the current level. After the first p nodes of a level are processed, the next p nodes are processed by the use of the array index as in line 8. The lines 6 and 7 are to be modified to retrieve the qualified records in the final level. It is straight forward to see that the range query is processed correctly.

```
algorithm PE_j;
  begin
1.    l = index[j];
2.    if ((current-level = level[l]) and (enable[l]))
3.    then
      begin
4.       if (low-limit[current-level] ≤ value[l]
                        ≤ high-limit[current-level])
5.       then
         begin
6.          for m = first-child[l] to last-child[l] do
7.             enable[m] = true;
         end
8.       index[j] = l+p;
9.       enable[l] = false;
      end;
  end;
```

THEOREM:

The time complexity of the parallel range search algorithm, on an SIMD processor array if p processors is $O(kN/p)$.

PROOF: For the execution of the algorithm, at any level i, the maximum number of times the processor array is invoked is given by:

$$\lceil (\text{number of nodes of level } i)/p \rceil$$

Now, the total number of times the processor array is invoked is given by

$$\sum_{i=1}^{k} \lceil (\text{number of nodes of level } i)/p \rceil)$$

$$= (\sum_{i=1}^{k} \lceil (\text{number of nodes of level } i) \rceil)/p$$

$$= O(kN/p+k).$$

Hence, the time complexity of this algorithm is $O(kN/p)$. □

We note that for the case $p=N$ this algorithm has the same complexity as the earlier one. But, earlier is superior as it uses less memory space. We also note that the maximum number of times the processor array is invoked is given by $k(N/p+1)$. This upperbound corresponds to the wosrt-case structure of the MAT. In an average-case the number of times that the processor array is invoked will be less than this bound. The author are presently working on this aspect.

3. CONCLUSIONS

We have developed an $O(kN/p)$ algorithm for range search problem using the linearized MAT data structure and SIMD computing system. Since the attributes are processed one after the other, the MAT based parallel algorithm has a lower bound complexity of $\Omega(k)$ on the processor array containig no more than N processor elements. Here, we have presented only the order estimates for the time complexity. However, the exact upper bounds and the average-case estimtes are needed to further substantiate the importance of this method.

4. ACKNOWLEDGEMENTS

We thank the three anonymous referees for their valuable comments which have certainly improved the presentation of this paper.

5. REFERENCES

[1] BENTLEY, J.L., and FRIEDMAN, J.H., "Data structures for range searching",ACM Computing Surveys, 11, 4(Dec. 1979), 397-409.

[2] KASHYAP, R.L., SUBAS, S.K.C., and YAO, S.B., "Analysis of multiattribute tree organization", IEEE **Trans. Software Engineering**, SE-2, 6(1977), 451-467.

[3] MEHLHORN, K., *Data Structures and Algorithms 3: Multidimensional Tree Structures and Computational Geometry*, EATCS monograph on Theoretical Computer Science, Springer-verlag, 1984.

[4] NAGESWARA RAO, S.V., IYENGAR, S.S., and VENI MADHAVAN, C.E., "A comparative study of multiple attribute tree and inverted file structures for large bibliographic files", **J. Information Processing and Management**,21, 5(1985), pp. 433-442.

[5] NAGESWARA RAO, S.V., and IYENGAR, S.S., The Multiple Attribute Tree Structure, Tech. Report TR85-030,Dept. of Computer Science, Louisiana State University, Baton Rouge, USA, June 1985.

[6] OVERMARS, M.H., *The Design of Dynamic Data Structures*, Lect. Notes in Comput. Sci. 156, Springer-verlag, Berlin, 1984.

[7] WILLARD, D.E., "New data structure for orthogonal queries", **SIAM J. Computing**, 14, 1(Feb. 1985), 232-253.