Spatial Join with R-Tree on Graphics Processing Units

Tongjai Yampaka
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Tongjai.Y@Student.chula.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
prabhas@chula.ac.th

***Abstract*: Spatial operations such as spatial join combine two objects on spatial predicates. It is different from relational join because objects have multi dimensions and spatial join consumes large execution time. Recently, many researches tried to find methods to improve the execution time. Parallel spatial join is one method to improve the execution time. Comparison between objects can be done in parallel. Spatial datasets are large. R-Tree data structure can improve the performance of spatial join.**

In this paper, a parallel spatial join on Graphic processor unit (GPU) is introduced. The capacity of GPU which has many processors to accelerate the computation is exploited. The experiment is carried out to compare the spatial join between a sequential implementation with C language on CPU and a parallel implementation with CUDA C language on GPU. The result shows that the spatial join on GPU is faster than on a conventional processor.

Keyword: Spatial Join, Spatial Join with R-tree, Graphic processing unit

I. INTRODUCTION

The evolution of Graphic Processing Unit is driven by the demand for real time, high-definition and 3-D graphics. The requirement for an efficient and fast computation has been met by parallel computation [1]. In addition, GPU architecture that supports parallel computation is programmable to solve other problems. This new trend is called General Purpose computing on Graphic processors (GPGPU). Developers can use the capacity of GPU to solve other problem beside graphics and can improve the execution time by parallel computation. In a spatial database, storing and managing complex and large datasets such as Graphic Information system (GIS) and Computer-aided design (CAD) are time consuming. A spatial database characteristic is different from a relational database because of data type. Spatial data types are point, line and polygon. The type of data depends on the characteristic of objects, for example a road is represented by a line or a city is represented by a polygon. An object shape is created by x, y and z coordinates. Therefore, spatial operations in a spatial database are not the same as operations in a relational database. There are specific techniques for spatial operations.

Spatial join combines between two objects on spatial predicates, for example, find intersection between two objects. It is an expensive operation because spatial datasets can be complex and very large. Their processing cost is very high. To solve this problem R-Tree is used to improve the performance for accessing data in spatial join. Spatial objects are indexed by spatial indexing [2] [3]. The objects are represented by minimum bounding rectangles which cover them. An internal node points to children nodes that are covered by their parents. A leaf node points to real objects. The join with R-Tree begins with a minimum bounding rectangle. The test for an overlap is performed from a root node to a leaf node. It is possible that there are overlaps in sub-trees too.

The previous work [4] introduces a technique for spatial join that can be divided into two steps.

- **Filter Step**: This step computes an approximation of each spatial object, its minimum bounding rectangle. This step produces rectangles that cover all objects.
- **Refinement Step**: In this step, spatial join predicates are performed over each object.

Recently, spatial join techniques have been proposed in many works. In a survey [5], many techniques to improve spatial join are described. One technique shows a parallel spatial join that improves the execution time for this operation.

This paper presents a spatial join with R-Tree on Graphic processing units. The parallel step is executed for testing an overlap. The paper is organized as follow. Section 2 explains the background and reviews related works. Section 3 describes the spatial join with R-Tree on Graphic processing units. Section 4 explains the experiment. The results are presented in Section 5. Section 6 concludes the paper.

II. BACKGROUND AND RELATED WORK

A. *Spatial join with R-Tree*

Spatial join combines two objects with spatial predicates. Objects have multi-dimension so it is important to efficiently retrieve data. In a survey [5], techniques of spatial join are presented. Indexing data such as R-Tree is one method which improves I/O time. In [6], R-Tree is used for spatial join. Before executing a spatial join predicate in the leaf level, an overlap between two objects from parent nodes is tested. When parent nodes are overlapped the search is continue into sub-trees that are covered by its parents. The sub-trees which are not overlapped from parent nodes are ignored. The reason is that the overlapped parent nodes are probably

overlapped with leaf nodes too. The next step, the overlap function test is called with sub-trees recursively. This algorithm is shown in Figure 1

```
SpatialJoin(R,S):
For (all ptr_S ∈ S) Do
For (all ptr_R ∈ R with ptr_R.rect ∩ ptr_S.rect ≠∅) Do
        If (R is a leaf node) Then
                Output (ptr_R , ptr_S )
        Else
                Read (ptr_R.child); Read (ptr_S.child)
                SpatialJoin(ptr_R.child, ptr_S.child)
        End
End
End SpatialJoin;
```

Figure 1 Spatial join with R-Tree

The work [6] presents a spatial join with R-Tree that improves the execution time. However, this algorithm is designed for a single-core processor. The proposed algorithm is based on this work but the implementation is on Graphics Processing Units.

B. Parallel spatial join with R-Tree

To reduce the execution time of a spatial join, a parallel algorithm can be employed. The work in [7] describes a parallel algorithm for a spatial join. A spatial join has two steps: filter step and refinement step. The filter step uses an approximation of the spatial objects, e.g. the minimum bounding rectangle (MBR).

The filter admits only objects that are possible to satisfy the predicate. A spatial object is defined in the form {MBRi,IDi} where i is a key-pointer data for the object. The output of this step is the set [{MBRi,IDi},{MBRj,IDj}] if MBRi intersects with MBRj. Each pair is called a candidate pair. The next step is the refinement step. Pair of candidate objects is retrieved from the disk for performing a join predicate. To retrieve data, it reads the pointers from IDi and IDj. The algorithm creates tasks for testing an overlap in the filter step in parallel. For example in Figure 2, R and S denote spatial relations.

The set {R1,R2,R3,R4,R5,R6,…,RN} is in R root and the set {S1,S2,S3,S4,S5,S6,…,SN} is in S root. In the algorithm described here the filter step is done in parallel.

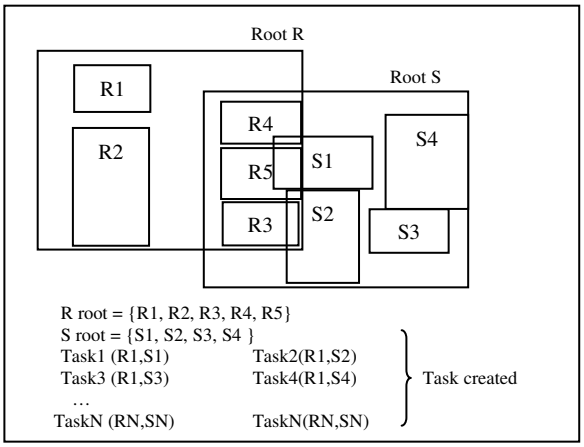

Figure 2 Filter task creation and distribution in Parallel for R-tree join

The algorithm is designed for parallel operation on a CPU. In this paper we use the same idea for the algorithm but it is implemented on a GPU.

In other research [8], R-Tree is used in parallel search. The algorithm distributes objects to separate sites and creates index data objects from leaves to parents. Every parent has entries to all sites. A search query such as windows query can perform search in parallel.

C. Spatial query on GPU

For a parallel operation in GPU, the work in [9] implements a spatial indexing algorithm to perform a parallel search. A linear-space search algorithm is presented that is suitable for the CUDA [1] programming model. Before the search query begins, a preparation of data array is required for the R-Tree. This is done on CPU. Then the data array is loaded into device memory. The search query is launched on GPU threads. The data structure has two data arrays represented in bits. The arithmetic at bit level is exploited. The first array stores MBR co-ordinate referred to the bottom-left and top-right co-ordinates of the i MBR in the index. The second array is an array of R-Tree nodes. R-Tree nodes store the set {MBRi, childNode|t|}. ChildNode|t| is an index into the array representing the children of the node i. When the search query is called, the GPU kernel creates threads to execute the tasks. Then copy two data arrays to memory on device. Finally the main function in GPU is called. The algorithm is shown in Figure 3. The result is copied back to CPU when the execution on GPU is finished.

```
Clear memory array  (in parallel).
For each thread
if Search[i] is:
  For each search[i] overlaps with the query MBR node j:
      If the child node j is a leaf, mark it as part of the
      output.
      If the child node j is not a leaf, mark it in the
      Next Search array.
Sync Threads
Copy next Search array into Search[i] (in parallel).
```

Figure 3 R-Tree Searches on GPU

III. IMPLEMENTATION

A. Overview of the algorithm

Most works have focused on the improvement of the filter step. The first filter step assumes that the computation is done with MBR of the spatial objects. In this paper, this step is performed on CPU and the data set is assumed to be in the data arrays. The algorithm begins by parallel filtering objects on GPU. The steps of the algorithm are as follows.

• **Step 1**: The data arrays required for the R-Tree are mapped to the device memory. The data arrays are prepared on CPU before sending them to device.

• **Step 2**: Filtering step, a function to find an overlap between two MBR objects is called. Threads are created on GPU for execution in parallel. The results are the set of MBRs which are overlapping.

• **Step 3**: Find leaf nodes, the results of step 2, the set of MBRs, are checked whether they are in the leaf nodes or not. If they are the leaf nodes, return the set as the result and send them to the host. If they are not the leaf nodes and then they are used as input again recursively until reaching leaf nodes.

B. Data Structure in the algorithm

Assume MBRs objects are stored in a table or a file. In the join operation, there are two relations denote as R and S. MBRs structure (shown in C language syntax) are in the form:

```
Struct MBR_object {
        int min_x,max_x,min_y,max_y;
};
/*x, y coordinate rectangle of object*/

Struct MBR_root {
        int min_x,max_x,min_y,max_y;
        child[numberOfchild];
};
/*x, y coordinate rectangle of root*/

MBR_root  rootR [numberOfrootR];
MBR_root  rootS [numberOfrootS];
/*Array of rootR and rootS relation*/

MBR_object  objectR [numberOfobjectR];
MBR_object  objectS [numberOfobjectS];
/*Array of objectR and objectS relation*/
```

C. R-Tree Indexing

An R-Tree is similar to a B-Tree which the index is recorded in a leaf node and it points to the data object [4]. All minimum bounding rectangles are created by x, y coordinates of objects. The index of data is created by packing R-Tree technique [10]. The technique is divided into three steps:

1) Find the amount of objects per pack. The number of child is between a lower bound (m) and an upper bound (M) values.

2) Sort the data on x or y coordinates of rectangle.

3) Assign rectangles from the sort list to the pack successively, until the pack is full. Then, find min x, y and max x, y for each pack to create the root node.

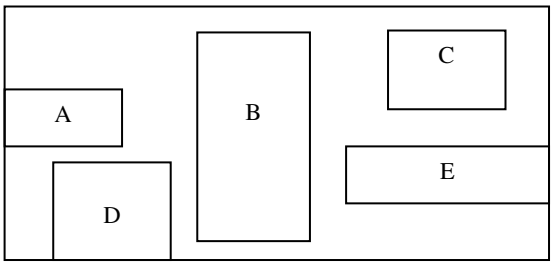

Figure 4 MBRs before split node R-Tree

An example is shown in Figure 4. It has five rectangles of objects. The objects are ordered according to x-coordinate of the rectangle. The sorted list is {A, D, B, E, C}. Define objects per pack as three. The assignments of objects into packs are:

Pack1 = {A, D, B}
Pack2 = {C, E}

In the next step, a root is created. Compute min x, min y and max x, max y.

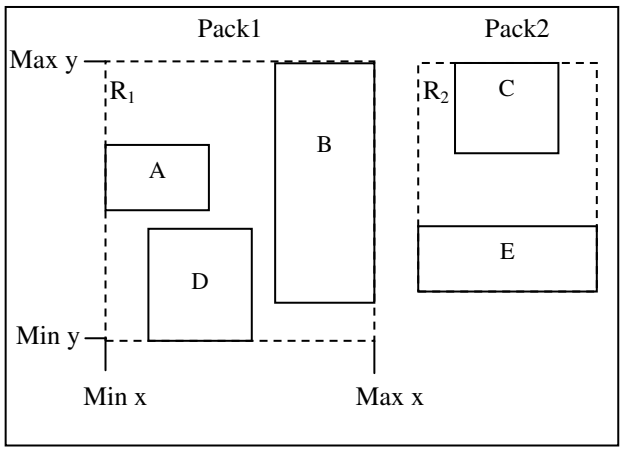

Figure 5 MBRs after split node R-Tree

The root node of pack1 is R1 and the root node of pack2 is R2. R1 points to three objects: A, D and B. R2 points to two objects: C and E. The root coordinate is computed from min x, min y max x, max y of all objects which the root covers them. In the example, only one relation is shown.

R-Tree creation is done on CPU. The difference is in the spatial join operation. The spatial join on CPU is sequential and on GPU is parallel.

D. Spatial join on GPU

To parallelize a spatial join, the data preparation is carried out on CPU, such as MBRs calculation and splitting R-Tree nodes. In GPU, the overlap function and the intersection join function are executed in parallel.

1) Algorithm

• **Overlap**: This step is the filter step for testing the overlap between root nodes R and S.

1. Load MBR data arrays (R and S) to GPU.
2. Test the overlap Ri and Sj in parallel.
3. The overlap function call is:
   Overlap ((Sj.x_min < Ri.x_max)
   and (Sj.x_max > Ri.x_min)
   and (Sj.y_min < Ri.y_max)
   and (Sj.y_max > Ri.y_min))
4. For each Ri overlap Sj
5. Find Ri and Sj children nodes.

• **Find children**: Find children nodes which are covered by the root Ri and Sj.

a) The information from MBRs indicates the children that are covered by the root.

b) Load children data and send them to the overlap function.

• **Test intersection**: This is the refinement step. Compute the join predicate on all children of Ri and Sj using the overlap function above.

2) GPU Program Structure

CUDA C language is used. The language has been designed to facilitate graphic rendering on Graphics processing units. CUDA program has two phases [11]. In the first phase, the program on CPU, called host code, performs the data initialization and transfers data from host to device or from device to host. On the second phase, the program on GPU, called the device code, makes use of the CUDA runtime system to generate threads for execution of functions. All threads execute the same code but operate on different data at the same time. A CUDA function uses the keyword "__global__" to define function that is a kernel function. When the kernel function is called from the host, CUDA generates a grid of threads on the device.

In the spatial join, the overlap function is distributed to different blocks and is executed at the same time with different data objects. To divide the task, every block has a block identity calls blockIdx.

For example:

• **Objects**

Relation R = {Robject0, Robject1, Robject2,..,RobjectN},
Relation S = {Sobject0, Sobject1, Sobject2,…,SobjectN}

• **Overlap function**: Compare all objects. Find x and y coordinates in the intersection predicate.

The sequential program on CPU executes only one pair of data at the one time.

Robject0 compare Sobject0
Robject0 compare Sobject1
Robject0 compare Sobject2
...
RobjectN compare SobjectN..timeN

On GPU, the CUDA code on device generates blocks for execution all data on different blocks.

Block0 = Robject0 compare Sobject0
Block1 = Robject0 compare Sobject1
Block2 = Robject0 compare Sobject2
...
BlockN = RobjectN compare SobjectN

The memory is allocated for execution between CPU and GPU. First, allocate memory for data structure of root R-Tree and MBRs of objects. Second, allocate memory of data arrays to store results. When the task is done copy data arrays back to host.

The nested loop is transformed to run in parallel. The rectangle of objects are mapped to 2D block on GPU. The outer loop is mapped to blockIdx.x and the inner loop is mapped to threadIdx.y.

The call to kernel function is:

kernel<<<number of outer loop,number of inner loop>>>.

CUDA kernel generates blocks and threads for execution.

IV. EXPERIMENTATION

A. Platform

The spatial join is coded in C language for sequential version. CUDA C language is used in parallel version. Both versions run on Intel Core i3 530 DDR3 2.93 GHz 2 GB memory. GPU NVIDIA GT440 1092 MHz.1024 MB and CUDA 96 Cores.

B. Dataset

In the experiment, the dataset is retrieved from R-Tree portal [12]. In the data preparation step the minimum bounding rectangles are pre-computed. The attributes in the dataset consist of Roads join River in Greece, Streets join Real roads in Germany.

TABLE I DATASET IN EXPERIMENTATION

Pair of dataset	Amount MBRs	Data size
Greece		
Rivers join Roads	47,918	0.7 MB
Germany		
Streets join Real roads	67,008	0.6 MB

Table 1 shows the number of MBRs and the size of dataset. All datasets are in text file. A C function is used to read data from a text file to data arrays.

V. RESULT

Spatial join is tested with dataset in Table 1 with two functions (Overlap function of root nodes and Intersection function of children nodes). In the experiment, the time to read data from text files and stores them to data arrays is ignored. The execution time of spatial join operation between CPU and GPU is compared. The generation of R-Tree is done on CPU in both sequential and parallel version. Only the spatial join operations are different.

A. Performance comparison between sequential and parallel

The results are divided into two functions: overlap and intersect.

TABLE II EXECUTION TIME ON GPU AND CPU

Pair of dataset	Overlap (ms)		Intersection (ms)		Total (ms)	
	CPU	***GPU***	***CPU***	***GPU***	***CPU***	***GPU***
Greece						
Rivers join Roads	18	4	72.67	22.33	90.67	26.33
Germany						
Streets join Real roads	5.33	4	74.00	39.67	79.33	43.67

The result in Table 2 shows that the execution time on GPU is faster than on CPU. For the dataset 1, the overlap function on GPU is 77.78% faster (4 ms versus 18 ms or about 4x); the intersection function is 69.27% faster (3x). The total execution time on GPU is 70.96% faster (3.4x). For the dataset 2, the overlap function on GPU is 25% faster (1.3x); the intersection function is 46.40%

faster (1.8x). The total execution time on GPU is 44.96% faster (1.8x). The speedup depends on the data type as well. If data has larger numbers, the execution time is longer too. In the experiment, the dataset 1 is floating point data. It has six digits per one element. Execution time is higher than the dataset 2 because the dataset 2 has integer data. It has four digits per one element.

The time to transfer data is significant. The data transfer time affected the execution time. The total running time in Table 2 includes the data transfer time from host to device and device to host.

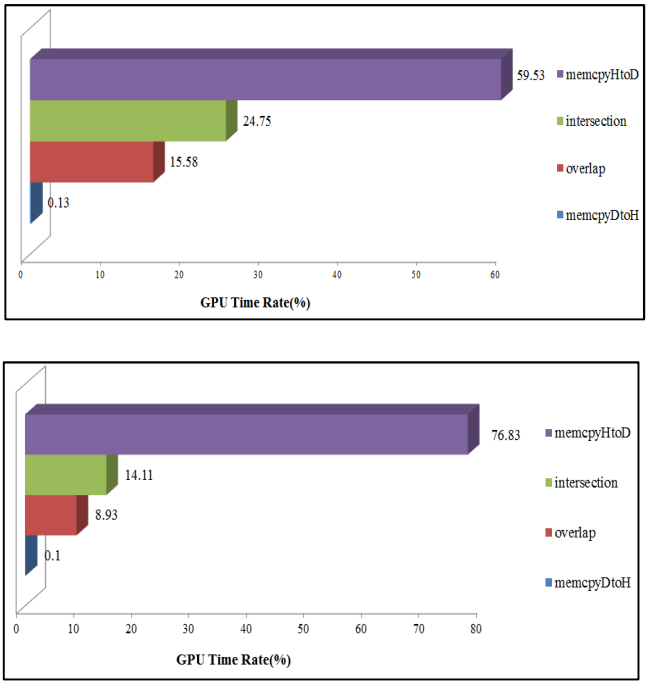

Figure 6 Transfer rate dataset 1, dataset 2

Figure 6 shows the data transfer rate on GPU. The dataset 1 has 47,918 records and its size is 0.7 MB. The data transfer time of this dataset is 59.53% of the execution time. The dataset 2 has 67,008 records and is 0.6 MB. The data transfer time of this dataset is 76.83% of the execution time.

VI. Conclusion

This paper describes how a spatial join operation with R-Tree can be implemented on GPU. It uses the multi-processing units in GPU to accelerate the computation. The process starts with splitting objects and indexing data in R-Tree on the host (CPU) and copies them to the device (GPU). The spatial join makes use of the parallel execution of functions to perform the calculation over many processing units in GPU.

However using Graphic Processor Unit to perform general purpose task has limitations. The symbiosis between CPU and GPU is complicate. There is a need to transfer data back and forth between CPU and GPU and the data transfer time is significant. Therefore, it may be the case that the data transfer time will dominate the total execution time if the task and the data are not carefully divided.

The future work will be on how to automate and coordinate the task between CPU and GPU. There are other database management functions that are suitable to be implemented in GPU too. It is worth the investigation as GPU becomes ubiquitous nowadays.

Reference

[1] NVIDIA CUDA Programming Guide, 2010. Retrieve at http://developer.download.nvidia.com

[2] A. Nanopoulos, A. N. Papadopoulos and Y. Theodoridis Y. Manolopoulos, R-trees: Theory and Applications, Springer, 2006.

[3] Xiang Xiao and Tuo Shi, "R-Tree: A Hardware Implemention," Int. Conf. on Computer Design, Las Vegas, USA, July 14-17, 2008..

[4] Gutman A., "R-tree:A Dinamic Index Structure for Spatial Searching," ACM SIGMOD Int. Conf. , 1984.

[5] E.H. Jacox and H. Samet, "Spatial Join Techniques," ACM Trans. on Database Systems, Vol.V, No.N, November 2006, Pages 1–45.

[6] Hans P. Kriegel and B. Seeger T. Brinkhoff, "Efficient Processing of Spatial Joins Using R-tree," SIGMOD Conference, 1993, pp.237-246.

[7] L. Mutenda and M. Kitsuregawa, "Parallel R-tree Spatial Join for a Shared-Nothing Architecture," Int. Sym. on Database Applications in Non-Traditional Environments, Japan, 1999, pp.423-430.

[8] H. Wei, Z. Wei, Q. Yin, "A New Parallel Spatial Query Algorithm for Distributed Spatial Database," Int. Conf. on Machine Learning and Cybernetics, 2008, Vol.3, pp.1570-1574.

[9] M. Kunjir and A. Manthramurthy, "Using Graphics Processing in Spatial Indexing Algorithm", Research report, Indian Institute of Science, 2009.

[10] K. Ibrahim and F. Cristos, "On Packing R-tree," Int. Conf. on Information and knowledge management, ACM, USA, 1993, pp.490-499.

[11] David B. Kirk and Wen-mei W. Hwu, Programming Massively Parallel Processors A Hands-on Approach, Morgan Kaufmann, 2010.

[12] R-tree Portal. [Online]. http://www.rtreeportal.org