A Survey on Massively Parallelism for Indexing Multidimensional Datasets on the GPU

Pramod B. Deshmukh
Asst. professor
DYPSOEA, Ambi, Pune
SPPU, Pune

Yogesh B. Lokare
BE scholar
DYPSOEA, Ambi, Pune
SPPU, Pune

Ajay V. Katware
BE scholar
DYPSOEA, Ambi, Pune
SPPU, Pune

Pankaj A. Patil
BE scholar
DYPSOEA, Ambi, Pune
SPPU, Pune

ABSTRACT
CUDA is a parallel programming environment that facilitates significant performance improvement by leveraging the massively parallel processing capability of the GPU. The general purpose computing, on graphics processing unit (GP-GPU) has turn up as a new cost effective parallel computing framework, in high performance computing research that enables large amounts of datasets to be processed in parallel. Large scale scientific data intensive applications have been playing a major role in modern high performance computing research. This large amount of data can be accessed by scientific data analysis applications such as multi-dimensional range query, but not much research has been conducted on multidimensional range query on the GP-GPU. Inherently multi-dimensional indexing trees such as R-Trees are not well suited for GPU environment because of its irregular tree traversal. It has been known that traversing hierarchical tree structures in an irregular manner make it difficult to exploit parallelism and to maximize the utilization of GPU processing units. Then to avoid the drawbacks of R-Tree the novel MPTS (Massively Parallel Three-phase Scanning) R-tree traversal algorithm for multi-dimensional range query was proposed, that Recursive access to tree nodes into sequential access. Furthermore, the recursive tree search algorithms often fail because of the GPU's tiny runtime stack size. Then the proposed work of a novel parallel tree traversal algorithm—massively parallel restart scanning (MPRS) for multi-dimensional range queries avoids recursion and irregular memory access. Then the proposed MPRS algorithm traverses hierarchical tree structures with mostly contiguous memory access patterns without recursion, which offers more chances to optimize the parallel SIMD algorithm.

Keywords
CUDA, GPGPU,Parallel multi-dimensional indexing, multi-dimensional range query,Parallel R-tree

1. INTRODUCTION
Modern GPUs that have many processing units are being with success and widely used as high performance accelerators for several general computations in variedfields.The general purpose computing on graphics processing unit (GP-GPU) has turn up as a new cost effective parallel computing paradigm in high performance computing research that allows large amount of data to be processed in parallel. GPUs enable large independent datasets to be processed in a single instruction multiple data (SIMD) fashion, so a broad range of computationally expensive but inherently parallel computing problems, such as medical image processing , scientific computing , and computational chemistry , have been successfully accelerated by GPUs.

In many scientific disciplines, sensor devices and simulators generate truly large amounts of multi-dimensional datasets, and the datasets are expanding in size every day. Multidimensional range query is one of the most common access patterns into such datasets, and is an important class of problems in data-intensive scientific computing and computer graphics as well. In order to handle multidimensional range queries efficiently, a large number of efficient and scalable indexing structures such as R-trees [6], R*-trees [5], and Hybrid-trees [7] have been proposed and improved. The R-tree is a data structure for organizing and querying multidimensional non-uniform, overlapping data. But because of irregular tree traversal R-Trees are not well suited for GPU environment. Regardless of their popularity, hierarchical multi-dimensional indexing trees are known to be inherently not well- suited for parallel processing due to their irregular and recursive back-tracking tree traversal patterns. In computer graphics, task parallelism is exploited to utilize a large numberof GPU cores, i.e., each processing unit in GPU traverses a binary BVH for different rays or objects. With such task parallelism approach, a very large number of queries can be processed concurrently, but it does not improve the response time of each individual query. As another form of parallelism, data parallelism can be exploited to make each individual query run faster.

CUDA (Compute Unified Device Architecture) programming model allows programmers to run parallel algorithms on GPU that can take the advantage of data-parallelism or task-parallelism while running serial portion of the algorithms simultaneously on CPU. Although NVIDIA keeps improving GPU architecture and CUDA programming model so that application programmers write general-purpose parallel programs on GPU, Still GPU has many restrictions because of the runtime stack size. As the stack size of the GPU is very small it will make difficulties to convert various sequential algorithms into parallel algorithms with parallel random access memory. In order to resolve the tiny runtime stack problem, several treetraversal methods have been proposed in the computer graphics community, such as kd-restart [8],

fixed short stack [9], rope tree[10], and parent link [11] search algorithms.

It also discuss a Massively Parallel 3 -phase-search algorithm for a spatial indexing structure - KDB-tree [4], which enables us to avoid irregular search path. Massively Parallel Three-phase Scanning algorithm performs pruning of irrelevant tree nodes and efficiently serves multi-dimensional range queries on the GPU.

Multi-dimensional range query may overlap multiple bounding boxes of a single tree node. Hence, legacy multi-dimensional range query algorithms use recursion or stack, and visit the overlapping child nodes in depth-first order. Since the runtime stack on the GPU is tiny, we develop a variant of multi-dimensional indexing trees-MPHR-tree, where each tree node embeds the largest leaf index (monotonically increasing sequence number of a leaf node, or a Hilbert value) of its sub-tree, which helps avoid the recursion and irregular memory access. The embedded leaf index is necessary for a novel multi-pass range query algorithm—massively parallel restart scanning that traverses the MPHR-tree structures in a mostly sequential fashion. MPRS avoids visiting the already visited nodes by keeping track of the largest index of visited leaf nodes.

2. RELATED WORK

In spatial-temporal database community, there has been extensive research on multidimensional indexing tree structures, starting with the seminal work on R-trees [6]. R-tree is a balanced tree structure whose tree node consists of an array of minimum bounding boxes (MBBs). The MBB of a tree node is the smallest multidimensional box that encompasses all the data in the sub-tree, i.e., the MBB in R-tree leaf node encloses nearby spatial objects and the MBB of internal tree nodes encloses all the underlying MBBs of lower level sub-trees in a hierarchical way.

CUDA threads use the fast shared memory as their runtime stack, but the latest Tesla GPUs have only 48 bytes of shared memory. Due to their tiny stack sizes, the recursive search algorithms of multidimensional indexing structures often fail. Although a large number of multi-dimensional indexing structures have been proposed, the search algorithms of those methods are similar in a sensethat they recursively prune out the sub trees depending on whether a given query range overlaps the bounding boxes of sub-trees.In order to resolve the tiny runtime stack problem, several tree traversal methods have been proposed in the computer graphics community, such as Kd-restart [8], fixed short stack [9], rope tree [10], and parent link [11] search algorithms.

As multi-core architectures have evolved, a couple of recent efforts were made to exploit SIMD execution of GPU to improve database query performance. Zhou et al. Proposed to compare multiple keys of B+-trees at the same time using SIMD instruction [12]. Kaldewey et al. [13] also proposed a parallel search algorithm, called *P-ary search*, for one dimensional sorted lists and showed that it outperforms binary search algorithm on the GPU. Kim et al. Presented *FAST (Fast ArchitectureSensitive Tree)*, which rearranges a binary search tree into tree-structured blocks to maximize data-level and thread-level parallelism on GPU architecture [14]. Each block of FAST is the unit of parallel processing in a single streaming multi- processor (SMP) of the GPU. For multi-dimensional range queries, there can be several child nodes to visit. After one path is taken and the path search is completed, it is necessary to backtrack to the last place where there were multiple choices in paths so that another path can be taken.Although GPUs have a large number of processing units, each processing unit of high-performance GPUs is known to run a lot slower than a CPU core. Hence, some efforts have been made to improve the query processing throughput instead of reducing response time of each query. Fix et al. proposed a braided parallel one dimensional query processing method for B+-trees, wherein a single B+-tree exists in global memory of GPU and multiple independent queries are concurrently processed across SMs, and a block of threads in each SM process individual query in parallel [15]. Luo et al. [16] proposed a parallel R-tree traversal algorithm on GPU. Their work is similar to ours in a sense that they also tried to avoid irregular memory access and recursion by employing a queue in the shared memory of SMP. Their algorithm transforms the R-tree search into a breadth-first search (BFS). But storing tree nodes to visit in the shared memory is not very scalable since GPUs provide very small shared memory space.

2.1 Parallel Multidimensional Indexing on GPU

2.1.1 Background of CUDA

In order to process a large amount of data in parallel, a CUDA performs parallel to access small portions of the large input dataset in parallel. Each CUDA thread block consists of a set of CUDA threads that need to share intermediate data results and cooperate on memory access with each other through thread synchronization mechanisms that CUDA provides. In addition to the little small shared memory, CUDA enabled GPU cards to access global memory which can be shared by all CUDA threads.

A warp is the minimum thread scheduling unit in CUDA architecture, but the warp is uncontrollable by programmers. Instead, programmers will specify the number of blocks and the number of threads per block when accessing a CUDA kernel function. The blocks are distributed across the multiple SMPs, and multiple threads in a single block are executed by a set of CUDA process units during a single SMP at the same time.

2.2 Stackless Multidimensional Range Query Processing

In computer graphics, Stackless ray traversal algorithms have been proposed mainly because a large number of rays are traced in parallel and the overhead of using runtime stack can be very high, i.e., the size of memory space for runtime stack can be as large as the maximum stack depth times the number of rays. Hence, several Stackless traversal algorithms have been proposed for efficient ray tracing on the GPU.

2.2.1 Kd-Restart for Range Query Processing

The Kd-restart traversal algorithm [8], modifies the standard kd-tree traversal to eliminate all stack operations by restarting the search at the root of the tree. While the removal of the stack was motivated by the limitations of current GPUs, it also reduces the working set size needed to trace rays through a kd-tree Instead of backtracking, the kd-restart algorithm traverses a tree structure, multiple times from root node to a leaf node. In each leaf node, it visits, the algorithm computes a crossing point of the ray with hyper-plane boundaries of the leaf node. With the piercing point along the ray, kd-restart algorithm truncates the ray and searches the kd-tree with the updated ray from root node. Since the ray is truncated per each restarted traversal, it avoids visiting already visited leaf nodes.

However, if a query is not a line segment, but a multi-dimensional region, pruning out visited regions will not create a simple rectangular region, which will complicate the next restart. Hence, kd-restart algorithm cannot be directly applied in multi-dimensional range query processing. Another problem with kd-restart algorithm is that its memory access pattern is very irregular, and the number of tree node accesses for each query is very diverse, which significantly hurts SIMD efficiency.

2.2.2 Rope Trees

In order to avoid backtracking to previously visited tree nodes, auxiliary links—ropes between neighboring tree nodes can be added to kd-trees [17], [18]. Havran et al. [17] proposed rope tree where each node has pointers called ropes which store the neighboring nodes in each dimension, i.e., a three-dimensional kd-tree node has six ropes.

2.2.3 Parent Link Algorithm

Hapala et al. [11] proposed a parent link search algorithm for bounding volume hierarchies. In their proposed bounding volume hierarchies, each tree node stores a pointer to its parent node. When a tree traversal needs to backtrack to its parent node, the parent node can be fetched from global memory using the parent pointer. Although parent link algorithm eliminates the stack operations, the backtracking using parent pointer requires additional global memory accesses. A parent link algorithm can be used not only for ray tracing, but also for n-ary data parallel range query processing, thus we develop parent link algorithm for n-ary R-tree and multi-way BVH to compare against our MPRS algorithm.

2.2.4 Skip Pointer

Skip pointer [10] is similar to rope tree in a sense that each tree node has an auxiliary link to its right sibling node or a right sibling of its parent node. If the current tree node is not hit by a ray, skip pointer is followed instead of backtracking to its previously visited parent node. Unlike rope tree, skip pointer does not take into account the ray direction, which is known to incur performance penalty.

The Skip pointer algorithm does not consider any direction preference, we can adopt it for multi-dimensional range query in order to avoid stack operations and make the search path always visit non-visited node. If a tree node has no overlapping child node, skip pointer algorithm follows the skip pointer to visit a right sibling node or a sibling of its parent node.

2.2.5 Short Stack for R-tree

Horn et al. [9] extended Foley's kd-restart algorithm by employing a stack of bounded size. Pushing a new node onto stack will delete the node at the bottom of stack. When a tree traversal backtracks, it first searches the short stack. If the short stack is not empty, the parent node can be visited by accessing the topmost node on the stack. If the stack is empty, it restarts the search operation at the root of the tree again as in kd-restart. Since the short stack algorithm can be used for n-ary tree structures and range query processing, we implemented the short stack algorithm for parallel R-trees and multi-way bounding volume hierarchies. For multi-dimensional range queries, the short stack helps reduce the number of global memory accesses compared to a parent link and skip pointer.

2.3 Braided Parallel Indexing vs Data Parallel Partitioned Indexing

In GPU computing, *braided parallelism* implies that multiple independent jobs run in parallel on different SMPs, and each independent job is processed in a data parallel fashion across multiple processing units in a single SMP. Braided parallelism is commonly used in GPU applications since it fits nicely with multi-SIMD architecture of GPU. However braided parallelism does not scale when the number of submitted tasks is small.

Since maximizing the utilization of GPU processing units plays key role in improving the performance of CUDA applications, we compared two approaches that parallelize index search operations. One method is braided parallelism that assigns a different query to each SMP, i.e. a GPU that has 16 SMPs can execute 16 queries concurrently. Since there's only a single index in GPU memory, the index will be shared by all the SMPs, but different parts of the index will be accessed to serve different queries.

As task parallelism scales with a large number of concurrent jobs, this braided parallel query processing improves query processing throughput when a large number of queries are continuously submitted. However it would not help reduce the execution time of running each query.

In order to improve the response time of individual query, we devised another method that makes maximum use of *data parallelism*, where we partition the index into sub-indexes and distribute them to each SMP. Partitioning spreads and decreases the amount of work to be done for a single query across multiple SMPs because each SMP has a smaller partitioned index to work on.

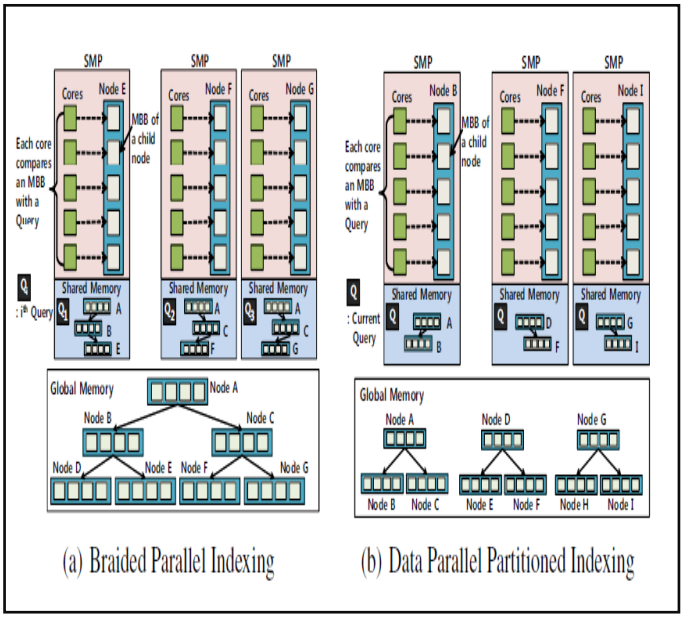

Fig 1: Braided Parallel Indexing vs Data Parallel Partitioned Indexing

Figure 1 illustrates the differences between braided parallel indexing scheme and data parallel partitioned indexing scheme. In braided parallel indexing shown in Figure 1(a), each SMP processes different user query, hence if fewer number of queries are submitted than the number of available SMPs, the utilization of processing units would be poor. However in data parallel partitioned indexing shown in Figure 1(b), the same single query is processed by all SMPs

concurrently with different partitioned indexes, thus utilization would be higher than that of braided parallel indexing even when the number of submitted queries is small.

2.4 Massively Parallel Three-Phase Scanning (MPTS) R-Trees on GPU

In MPTS search algorithm[3], a set of threads in a CUDA block cooperate to check in parallel if the minimum bounding boxes (MBBs) of child nodes overlap a given query, i.e., the number of threads in each block is set equal to the number of node fan-outs (the maximum number of child nodes). This parallel search scheme is desirable for SIMD architecture since all the threads in a single block read the same tree node and each thread independently determines whether a child node overlaps a given range query. After all the threads are done by comparing a query with MBBs of child nodes, they should agree with, which child node to visit next if there are more than one overlapping child node. The recursive search algorithm navigates down one of the overlapping nodes, and it backtracks to the current node so that it visits another overlapping node. This recursion needs a large run-time stack space, especially when the size of tree structure is large. Current run-time stack frame stores child nodes of the current node overlap so that when it backtracks to the current stack frame it restores the overlap information without comparing the MBBs and chooses the next child node to visit. However the recursive range query function is not scalable since it often fails when the size of the index is large and query range is also large.

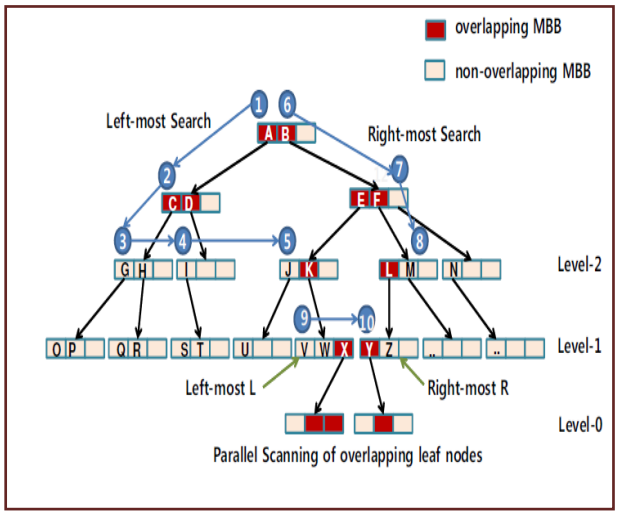

Fig 2: MPTS R-Tree Search with Sibling Check

In order to avoid backtracking, MPTS search algorithm selects at most one child Node to visit no matter how many child nodes overlap a query. As shown in Figure 2, MPTS search algorithm keeps choosing the leftmost node in each level in the first phase, (from step 1 to step 3), and in the second phase, the rightmost node in each level is visited (from step 4 to 6). Any node that is not in between the leftmost and rightmost nodes has no chance of overlapping the query. If there's an overlapping node outside of the leftmost and rightmost nodes, it contradicts that they are the leftmost or rightmost nodes. This pruning process determines which nodes are irrelevant and reduces the number of tree nodes to visit. If a level-1 node has an MBB of child leaf nodes that overlaps the query, the child leaf node is fetched and the data stored in the leaf node are compared against the query.

In the example shown in Figure 2, let's assume a single warp consists of three threads and the maximum child nodes of each tree node are also three. In step one (circled one in the figure), two threads will find the red-colored left and middle MBBs (A and B) of the root node overlap a given query range. The third thread will find out that the root node doesn't have a third child node and wait for the other two threads to finish. In the leftmost search phase, the middle overlapping MBB B will be ignored, but the left child node A will be chosen and visited. In step 2, again the first and second threads find out the left and middle MBBs (C and D) overlap, and C will be chosen just because it is located in the leftmost position among them. In step 3, threads will find out none of the MBBs (G and H) overlap. In traditional recursive tree traversal algorithms, we should go back to the parent Node, but backtracking should be avoided in GPU environment. instead, we can blindly navigate down further following the rightmost child node (i.e. H → Q and R) although we know they do not overlap. This approach will increase the distance between leftmost and rightmost nodes and the probability of false hits. A better way of avoiding false hits and reducing the distance between leftmost and rightmost nodes is the *sibling jump* shown in steps 3, 4, and 5 in Figure 2, which fetches its right sibling node when there's no overlapping MBB in current node.

As itnavigates down the trees we access the one and only child node in each level. Hence MPTS search algorithm does not require backtracking or global memory access. The penalty of eliminating the backtracking is that we may have to visit more number of leaf nodes. The leftmost leaf and the rightmost leaf node can be located very far from each other in the tree structure. In traditional R-trees, the MBBs of a tree node are stored in random order. Thus, MPTS search algorithm might have to scan all the leaf nodes even for a very small query range.

Scientific datasets are usually static, i.e., they do not change once they are acquired from sensor devices. Taking advantage of this, we sort the multi-dimensional data objects using a space filling curve—Hilbert curve that preserves good spatial locality [19]. As Hilbert curve clusters spatially nearby objects, we can create tight bounding boxes for the sorted datasets. With the bounding boxes, R-trees can be constructed in a bottom-up fashion as in Packed R Trees [20]. The bottom-up construction makes the node utilization of low level tree nodes almost 100 percent, however it may result in large overlapping regions for the bounding boxes in the root node.

2.5 Massively Parallel Hilbert R-Tree

MPTS reduces the number of leaf nodes to be accessed, but still it accesses a large number of leaf nodes that do not have requested data. Hence we designed a variant of R-trees that work on the GPU without stack problem and does not access leaf nodes that have not requested data called MPHR-Trees (Massively Parallel Hilbert R Trees.

MPHR-tree tags each leaf node with a sequential number—leaf index from left to right, and internal tree nodes of MPHR-tree store the maximum leaf index of its sub-trees as shown in Figs. 3.While traversing the tree structure, the query processing threads keeps track of the largest leaf index that they have visited. The maximum leaf index stored in each tree node is used to avoid recursive backtracking and re-visiting previously visited nodes. Instead of the sequential leaf index, the Hilbert value of data objects can be used to allow dynamic insertion of data object into previously constructed

MPHRtree, but the Hilbert value, usually requires a larger amount of storage, and multiple data objects can be mapped to the same Hilbert value if the level of Hilbert curve is not fine grained enough to distinguish all the data objects.

In order to sort the Hilbert values of the multi-dimensional data, we employed Thrust [21], which is an open source C++ STL-like GPU library that implements many core parallel algorithms including radix sort. After sorting the entire data objects using the radix sort on the GPU, B number of consecutive data objects are stored in the same leaf node where B is the maximum number of data that the leaf node can hold. After assigning all the data objects to leaf nodes, the bottom-up constructed MPHR-tree builds MBRs of leaf nodes via parallel reduction and stores the MBRs in their parent nodes. After creating parent nodes, the bottom-up construction goes up one level, and repeats until only one root node is left. This construction can be easily parallelized on the GPU.

2.6 MPRS: Massively Parallel Restart Scanning

Massively parallel restart scanning algorithm[2] is a multi-dimensional range query processing algorithm we propose which traverses the hierarchical tree structures from root node to leaf nodes multiple times as in kd-restart algorithm [8]. For a given range query, no matter how many MBRs of child nodes overlap a given query range, our MPRS algorithm always selects the leftmost overlapping child node unless all its leaf nodes have been already visited. Once it determines which child node to access in the next level, it does not store the overlapping child node information in the current tree node as an activation record but immediately discards it because MPRS algorithm does not backtrack to already visit tree nodes.

The MPRS range query algorithm resembles the B+-tree search algorithm in that both search algorithms scan leaf nodes. In one-dimensional B+-tree, range query traverses hierarchical trees to find out the leftmost (smallest) data value within the query range, and performs leaf level scanning until it finds out a data value greater than query range and terminates the search. However, in multi-dimensional space, data objects that overlap a given query range may not be located in a single span of leaf nodes. Even after sorting the multi-dimensional data objects using the Hilbert space filling curve, a query range may overlap leaf nodes in multiple segments on the Hilbert curve. The MPRS search algorithm finds the smallest (in terms of the Hilbert curve index) multi-dimensional data object that overlaps a query range (D3 in Fig. 3). After it finds out an overlapping data object that has the smallest Hilbert value, it starts scanning its next sibling data objects to find out if they also overlap.

Our MPRS search algorithm scans and compares the overlap of a given query with data objects on the continuous Hilbert segment in a massively parallel way using a large number of threads on the GPU. If any single thread finds an overlapping data object, the scanning keeps fetching the next group of data objects and compares the overlap. However while scanning data objects on the Hilbert curve, all threads may find out none of the data objects are in the query range. If so, it stops scanning leaf nodes and restarts traversing MPHR-tree to find out the starting point of the next Hilbert curve segment that overlaps the query range. When restarting the tree traversal, MPRS search algorithm uses the leaf index stored in tree nodes in order to avoid visiting already visited leaf nodes. In the restarted tree traversal, any tree node whose maximum leaf index is smaller than the maximum leaf index of previously visited leaf nodes is ignored. This is simply because if we have visited a leaf node v there's no reason to visit internal tree nodes which are parent nodes of v's left siblings. In each restart traversal, only if a tree node overlaps a query and it is the leftmost child node that has at least one unvisited leaf node, the tree node is accessed in the next level. If all the overlapping data objects are stored in a single span of consecutive leaf nodes, the root node is accessed only once, which is the best case. Due to the clustering property of Hilbert curve, it is unlikely that the overlapping leaf nodes are widely spread throughout a large number of leaf nodes interleaved by non-overlapping leaf nodes. However, there might still be a chance that some nearby data objects can be spread across many non-contiguous sections of a Hilbert curve. In such a case, multiple restart tree traversal is necessary to skip a large number of non-overlapping sections of the Hilbert curve.

In order to reduce the number of restart tree traversal, we employ minimal backtracking, i.e., instead of starting a new tree traversal from root node immediately after visiting a non-overlapping leaf node, our MPRS algorithm fetches a parent of the last visited leaf node from global memory. In the parent node, it checks if it has any other leaf node that overlaps the query range and has a leaf index higher than the maximum leaf index of previously visited leaf nodes. If the parent node does not have such an overlapping leaf node, MPRS algorithm starts another tree traversal from root node. If the parent node has an overlapping but unvisited leaf node, leaf node scanning continues from the leaf node.

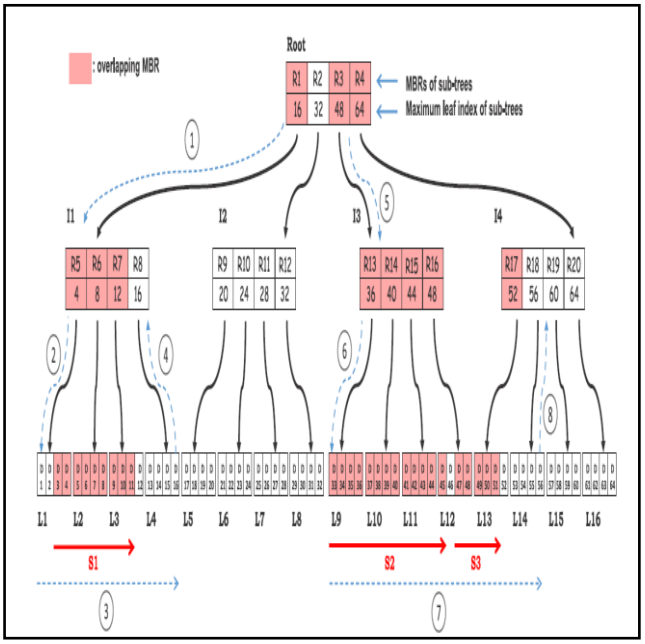

Fig 3:Massively parallel restart scanning with MPHR-tree structure

Fig. 3 shows an example of MPHR-tree and the MPRS search path for the data objects and the query ill. In the beginning, visited LeafIdx is set to 0 and each thread compares the MBR of each child node with a query range. Suppose the query range overlaps MBRs—R1, R3, and R4 in the root node. The MPRS algorithm ignores R3 and R4, and visits the leftmost child node L1 (1). Again, the query is compared with the MBRs in the node I1, and the leftmost overlapping leaf node L1 (2) is selected. Once we reach a leaf node, the multi-

dimensional coordinates of the data objects in the leaf node L1 are compared with the given query range. If some of the data objects overlap in a leaf node, its right sibling leaf node L2 will be visited and checked for overlapping data(3). In the example, MPRS algorithm keeps visiting right sibling leaf nodes L3 and L4. However because L4 has no overlapping data objects, its right sibling leaf node L5 is not accessed but we check its parent node I1 (4). Note that I1 was a previously visited node in this example, but note that parent check may visit an unvisited internal tree node if an overlapping section of Hilbert curve is long. Since I1 does not have any other overlapping child node which was never visited, MPRS algorithm restarts the search from root node.

When leaf node scanning stops, the visitedLeafIdx is set to 16 that is the largest leaf index stored in node I1. In the next restart traversal, although R1 overlaps the query range, R1's leaf index 16 is not greater than the current visitedLeafIdx, thus thread 1 ignores R1. Thread 2 also ignores R2 since its MBR does not overlap the query range. Thread 3 detects the overlap between R3 and the query, and since its leaf index 48 is greater than the current visitedLeafIdx, I3 will be selected as the next child node to visit(5). Thread 4 will also find its MBR R4 overlaps, but I4 will not be accessed since it is not the leftmost overlapping child node in current tree traversal. In I3, L9 will be selected as the child node to visit (6). In L9, data objects that overlap the query—D33, D34, D35, and D36 are found. Thus, its right sibling nodes L10, L11, L12, L13, and L14 are scanned and the visitedLeafIdx will be updated to 56 (7). Since L14 does not have any overlapping data object, parent check optimization fetches its parent node I4 from global memory. In node I4, R17 is the only MBR that overlaps but its leaf index 52 is smaller than the current visitedLeafIdx 56, hence it returns after setting visitedLeafIdx to 64 (8). In the next round of restart, a block of threads do not find any overlapping child node in the root node that has a leaf index value greater than 64. Finally, the search kernel function returns and the search finishes.

3. CONCLUSION AND DISCUSSION

In this paper, we discuss parallel multi-dimensional range query algorithm for GPU. The MPHR and MPRS algorithms improve the utilization of GPU architecture for range query processing, and avoids the irregular search path of transforming the tree traversal problem into a sequence data processing problem.

We have also compared the performance of braided parallel indexing against the data-parallel partitioned indexing and that shows braided parallel indexing improves system throughput when a large number of concurrent queries are submitted and data parallel partitioned indexing helps improve individual query response time. We also extended several Stackless rays tracing algorithms—short stack, parent link, and skip pointer for multidimensional range query with n-ary indexing trees, and conducted comparative performance study and showed our MPRS range query processing algorithm outperforms the other Stackless tree traversal algorithms mainly because our MPRS algorithm accesses mostly sequential memory blocks and does not backtrack to previously visited tree nodes.

In this work, it will discuss a novel parallel multi-dimensional indexing structure, MPHR-trees and MPRS tree traversal algorithm for multi-dimensional range query processing on the GPU. It has been known that multidimensional indexing structures are not well suited to parallel systems due to recursion and irregular tree access patterns. MPRS tree traversal algorithm (i) uses a large number of GPU threads to process a single query in a SIMD fashion in order to improve the query execution time, (ii) avoids warp-divergence by fetching only a single tree node in each step for a block of threads in a streaming multiprocessor, (iii) avoids recursion or stack operations by restarting tree traversal and avoiding visiting previously visited tree nodes by tracking the largest leaf index of visited tree nodes, (iv) and accesses mostly contiguous memory block by leaf node scanning.

4. REFERENCES

[1] NVIDIA CUDA Compute Unified Device Architecture. (2014). [Online]. Available: http://www.nvidia.com/

[2] Jinwoong Kim,Won-Ki Jeong and Beomseok Nam, "Exploiting Massive Parallelism for Indexing Multi-Dimensional Datasets on the GPU," *IEEE Trans. Parallel Distrib. Sys.*,VOL. 26, NO. 8, AUGUST 2015.

[3] Jinwoong Kim, Beomseok Nam, "Parallel Multi-dimensional Range Query Processing with R-Trees on GPU," *School of Electrical and Computer Engineering ,UNISTUlsan, 689-798, Korea.*

[4] R. E. Bellman, Adaptive Control Processes: A GuidedTour. Princeton, NJ, USA: Princeton Univ. Press, 1961.

[5] Norbery Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R∗-tree**:** An efficient and robust access method for points and Rectangles. In *Proceedings of 1990 ACM SIGMOD International Conferenceon Management of Data (SIGMOD)*, pages 322–331, May 1990.

[6] Antonin Guttman. R-trees**:** A dynamic index structure for spatial searching. In *Proceedings of 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 1984.

[7] Kaushik Chakrabarti and Sharad Mehrotra. The Hybrid tree: An index structure for high dimensional feature spaces. In *Proceedings of the 15th International Conference on Data Engineering (ICDE)*, pages 440–447, 1999.

[8] T. Foley and J. Sugerman, "KD-tree acceleration structures for a GPU raytracer," in Proc. ACM SIGGRAPH/EUROGRAPHICS Conf. Graph. Hardware, 2005, pp. 15–22.

[9] D. Horn, J. Sugerman, M. Houston, and P. Hanrahan, "Interactive k-d tree GPU raytracing," in Proc. Symp. Interact. 3D Graph. Games, 2007, pp. 167–174.

[10] B. Smits, "Efficiency issues for ray tracing," J. Graph. Tools, vol. 3,no. 2, pp. 1–14, 1998

[11] Hapala, T. Davidoiv, I. Wald, V. Havran, and P. Slusallek, "Efficient stack-less BVH traversal for ray tracing," in Proc. 27th Spring Conf. Comput. Graph., 2011, pp. 7–12.

[12] Jingren Zhou and Kenneth A. Ross. Implementing database operations using simd instructions. In *Proceedings of 2002 ACM SIGMOD InternationalConference on Management of Data (SIGMOD)*, 2002.

[13] Tim Kaldewey, Jeff Hagen, Andrea Di Blas, and Eric Sedlar. Parallel search on video cards. In *Proceedings of the First USENIX*

conference on Hot topics in parallelism (HotPar 09), 2009.

[14] Changkyu Kim, Jatin Chhugani, Nadathur Satish, Eric Sedlar, Anthony D Nguyen, Tim Kaldewey, Victor W. Lee, Scott A. Brandt, and Pradeep Dubey. Fast: Fast architecture sensitive tree search on modern cpus and gpus. In *Proceedings of 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2010.

[15] Jordan Fix, Andrew Wilkes, and Kevin Skadron. Accelerating braided b+ tree searches on a gpu with cuda. In *2nd Workshop on Applications forMulti and Many Core Processors: Analysis, Implementation, and Performance (A4MMC), in conjunction with ISCA*, 2011

[16] Lijuan Luo, Martin D.F. Wong, and Lance Leong. Parallel implementation of R-trees on the GPU. In *Proceedings of the 17th Asia and South Pacific DesignAutomation Conference (ASP-DAC)*, 2012.

[17] V. Havran, J. Bittner, and J. Zara, "Ray tracing with rope trees," in Proc. 14th Spring Conf. Comput. Graph., 1998, pp. 130–139.

[18] S. Popov, J. Gunther, H.-P. Seidel, and P. Slusallek, "Stackless KDtree traversal for high performance GPU ray tracing," Comput. Graph. Forum (Proc. Eurograph.), vol. 26, pp. 415–424, 2007.

[19] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, "Analysis of the clustering properties of the hilbert space-filling curve," IEEE Trans. Knowl. Data Eng., vol. 13, no. 1, pp. 124–141, Jan. 2001.

[20] R. Rew, G. Davis, and S. Emmerson. (1997). NetCDF User's Guide for C [Online]. Available: http://www.unidata.ucar.edu/packages /netcdf/cguide.pdf

[21] NVIDIA Thrust. (2014). [Online]. Available: https://developer.nvidia.com/thrust