USF UNIVERSITY OF SOUTH FLORIDA COLLEGE OF ENGINEERING

Parallel Index-based Search Operation for Database Systems via GPUs

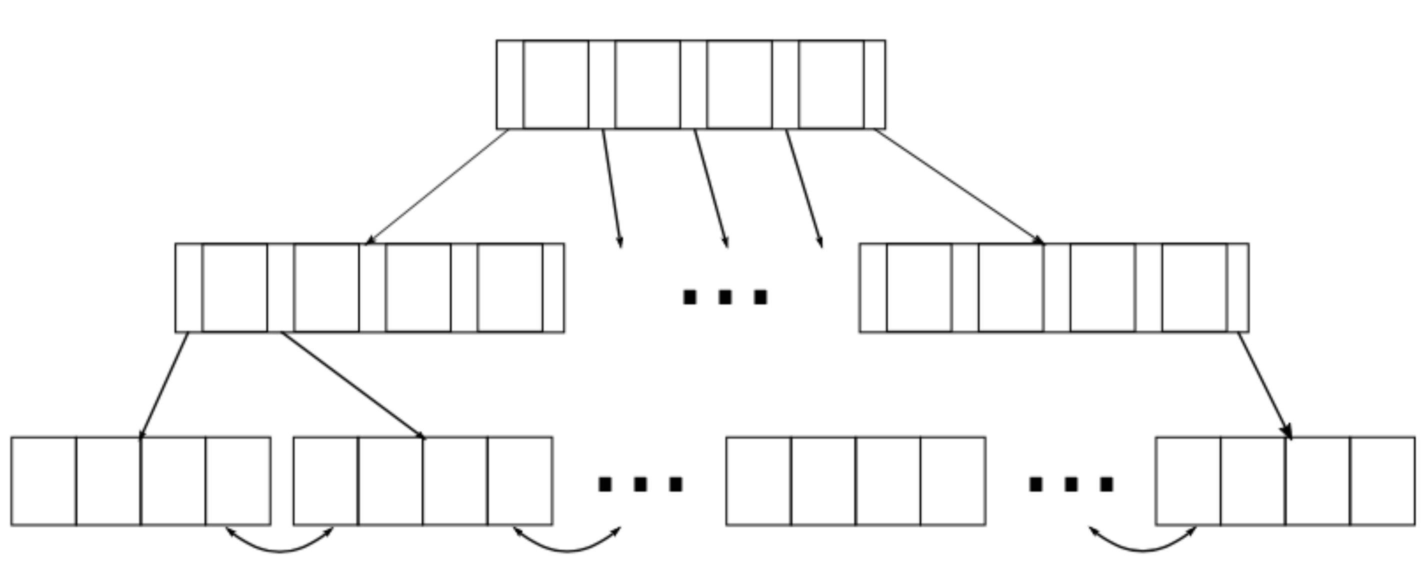
Napath Pitaksirianan, and Yi-Cheng Tu
Department of Computer Science and Engineering, University of South Florida

Abstract

Concurrently handle a large number of queries is a crucial characteristic of today's in-memory database system. By supporting such characteristics, index structure becomes a vital role in a database system. In recent years, GPUs have become the leading hardware for parallel computing. The unique architecture of high-performance computation, however, provides abundant opportunities for optimizing the algorithm towards better performance and achieving high utilization of GPU resources. This work presents our recent study in designing and optimizing parallel algorithms for index-search on Graphics Processing Units (GPUs). We present techniques to optimize the search operation on both equality and range searches by using a novel clustering technique that can maximize the utilization of an on-chip GPU cache system. To evaluate our index structure, we compare the searching time with the best CPU SIMI index-based searching.

Index Structure

- Handling a large volume of analytical workloads is a significant challenge in today's data management applications, online analytical processing (OLAP), and data warehouse.
- The state-of-art moves forward the performance of a searching operation by using an Index-based search.
- There are many types of index structure which uses in index-based searching, such as Binary tree, Red-black Tree, QuadTree, OctTree, K-D Tree, B Tree, B+ Tree.
- B+ Tree is the most common data structures for index searching in a database system, due to original designs for a system with small memory and large hard disk.
- B+ tree manages data on a page, which leaf nodes are connected as a list. This allows fast range search operation.
- In this work, we use our dynamic allocator to construct the index structure on GPUs.

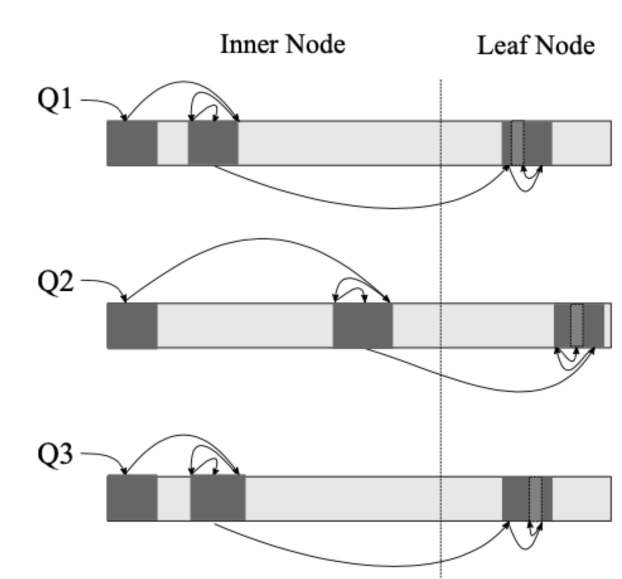


GPU Index-based Searching

- GPU index-based searching requires a large number of queries at the same time to maximize the utilization of GPUs.
- The tradition idea which loads partial top levels of the tree cannot maximize the utilization level of the cache system.
- In this work, we present new techniques to maximization of GPUs on Equality Search and Range Search.
- The queries are preprocessed by grouping similar queries and assign each group to a CUDA block can significantly improve GPUs cache utilization.
- Our study shows that the Equality Search query and the Range search query cannot handle in the same way to maximize the performance.

Equality Search Query

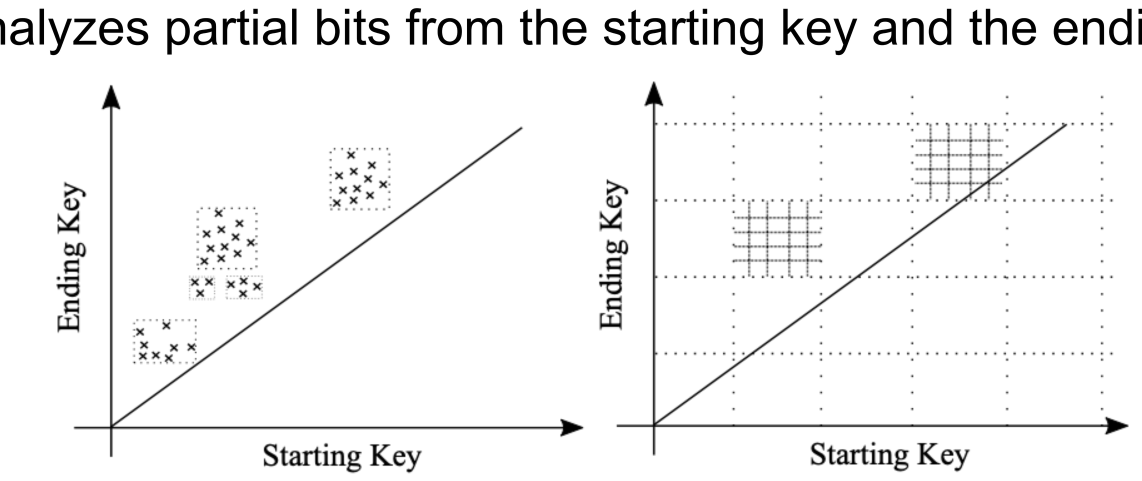
- The algorithm takes search keys and return key-pairs.
- By supporting a large number of queries, there is a high chance that multiple queries access to neighboring data.
- Grouping similar queries as one block and assign into a CUDA block can boost the performance.



- For example, Q1, Q3 access to median data, and Q2 accesses to high number datum. By grouping Q1 and Q2, the system can achieve high utilization of cache because of access to similar data from the cache system.
- We introduced "radix grouping operation" which group the queries by similarity inside bits level.

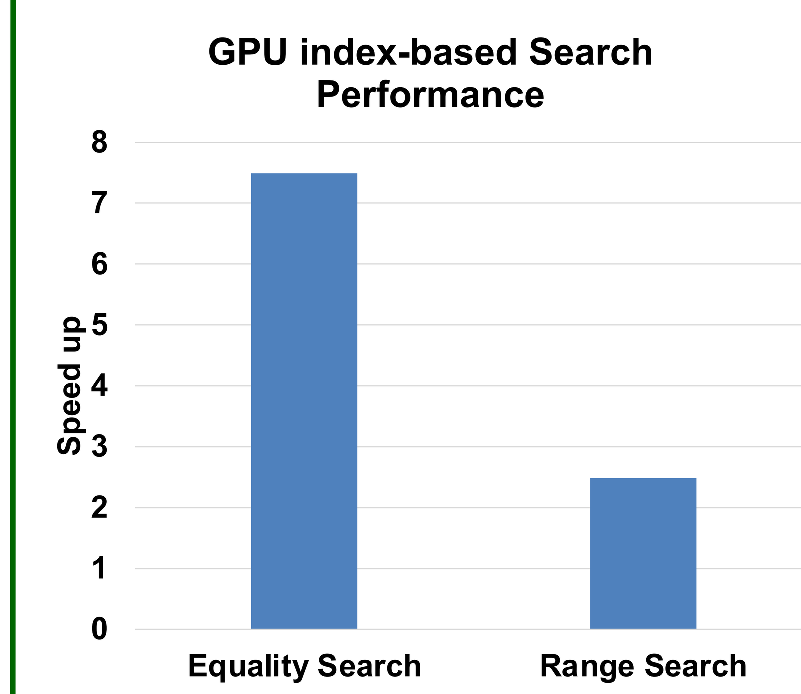
Range Search Query

- The operation takes two inputs; starting key and ending key.
- The operation returns a matching key in the range of starting key and ending key.
- Range search requires a unique method to group the queries.
- The grouping operation requires to know both the starting point and ending point.
- We present "advance radix grouping operation" which analyzes partial bits from the starting key and the ending key.

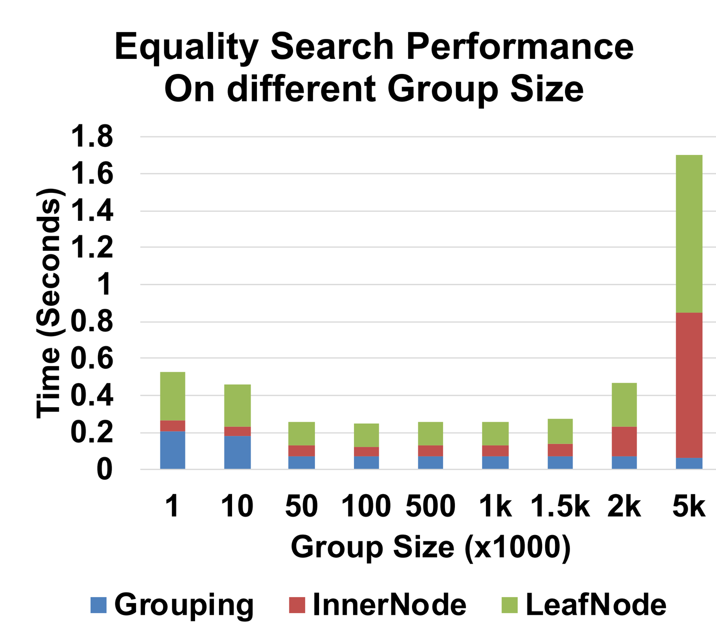


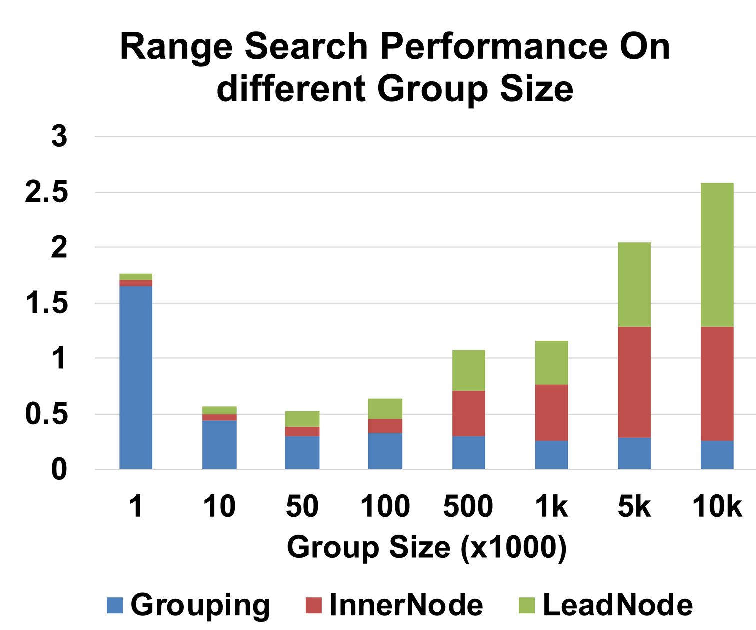
Experimental Result

- **Environment Setup**
  - CPU i9-7920X CPU @ 2.90GHz
  - 128GB of DDR4 2444-MHz memory
  - Nvidia Titan RTX GPU with 24GB of global memory.
  - End to End time comparison (I/O time included) to best SIMD CPU in [1].



- **Equality Search Experiment**
  - Experiment data are 100 to 900 millions queries.
  - The speed up is 7.4 time over SIMD CPU code.
- **Range Search Experiment**
  - Experiment data are 100 to 400 millions queries.
  - The speed up is 2.4 time over SIMD CPU code.
- **Group Size Experiment**
  - Experiment kernel running time on different group size
  - Experiment on 400 million queries.
  - 50000 Queries per group show the best performance on equality Searching and Range Searching
  - Profiler reported L1 cache hit rate is up to 91% on the best group size.





Conclusion

- GPUs required a large number of queries to achieve high performance.
- Clustering and grouping queries can significantly improve the utilization of the GPU's cache system.
- Equality Search and Range Search Query required a different technique to cluster a similar query.
- The peak L1 hit rate is up to 91% after grouping the queries.

References

[1] A. Shahvarani and H. Jacobsen. A hybrid b+-tree as solution for in-memory indexing on CPU-GPU heterogeneous computing platforms. SIGMOD Conference 2016, pp 1523–1538