Evaluating String Equi-Join on GPUs

Database Management Systems 2023

Hrushikesh Salunke 21407
Shivraj Jamgadepatil 21600

Computer Science and Automation
Indian Institute of Science, Bangalore

December 11, 2023

Agenda

Motivation

- Traditional database operations target CPUs and are very optimized for multi-threaded, multi-socket CPUs.
- GPUs provides an opportunity to further improve performance of these operation.
- String join attributes present a unique challenge cause of their variable size

Design Decisions

- Join algorithms.
  1. Naive nested loop join.
  2. Hash join.
- Calculating Join Size Upper bound before actual join computation.
- Done by using same computations as used by kernel to compute actual join.
- It avoids the need of allocating unnecessarily large GPU memory to store join output which might be the case when using some estimation models.

Design Decisions

- Use of linear probing for resolving collisions in hash table instead of chaining or quadratic probing.
- Linear probing allows very efficient lookup as shared memory utilization is higher due to sequential access pattern.
- Many of the collisions are avoided by over-allocating a little more memory to store the hash table.

Design Decisions

- In case join size does not fit in the GPU memory memory over-subscription mechanisms provided by NVIDIA are used.
  1. Pinned Memory.
  2. Unified Virtual Memory.

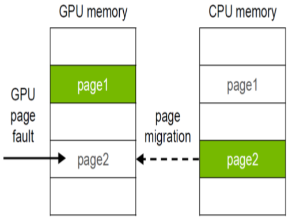

Figure: NVIDIA UVM

Methodology

- Read the tables into CPU memory.
- Compute Hash of strings using cityhash algorithm and store results in a new table.
- Generate hash table for table R on joining attribute.
- Compute upper bound for join output cardinality.
- If output size is greater than available GPU memory allocate memory using UVM else allocate memory in default way.
- Perform hash join in GPU using GPU resident hash-table.
- Materialize the results leveraging multi-threading and remove false positives.

Experimental setup

- Machine specifications.
  1. CPU: AMD Ryzen 5 3600X, 6-core 12 threads
  2. GPU: NVIDIA RTX 2060 super, 8GB
  3. DRAM: 16GB, 3200MHz
- The table sizes used for experimentation have 100K, 500K, 1 Million, 5 Million, 10 Million rows.
- To demonstrate that the approach works when output size is greater than available GPU memory, a chunk of memory is blocked.
- For the case of 10Million tuples a total of 5GB out of 8GB GPU memory was blocked memory.

Results

- We considered 4 different approaches against the postgreSQL database engine.
    - Naive GPU Join
    - Hash Join on GPU
    - Hash join using pinned memory
    - Hash Join using UVM
- Our approach provides a speedup of almost 8x over the conventional postgreSQL approach even in worst case. The graph in the next slide illustrates the same.
- Even with materialize time added to our hash computation time we achieve a speedup of more than 2X over postgreSQL for 10 Million tuples

Results

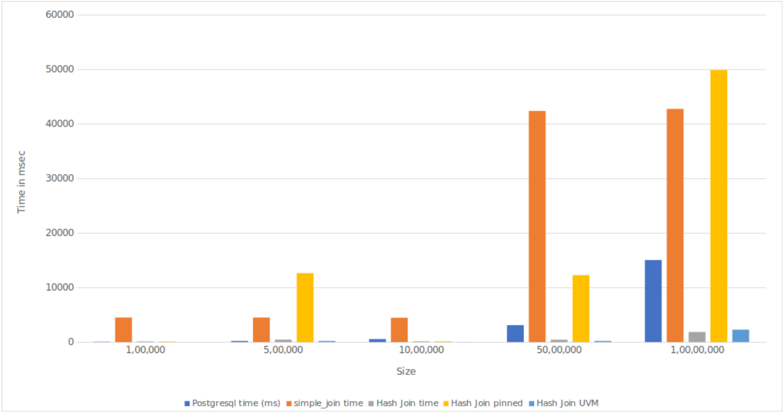

Figure: Experimental Results

Conclusions

- GPU computation presents an opportunity to speed up database operations.
- Pre-computation of join in case of non-partitioned join allows for efficient use of GPU memory without impacting the performance too much.
- Memory oversubscription techniques like UVM and pinned memory allow us to generate join outputs whose size might be bigger than available GPU memory
- Lock free hash maps allow for greater level of parallelism making join computation faster

Thank you!