Parallel Hash Join
CS 631-Project

Name: Neeraj Kerkar (18305R002), Shah rukh Hussian (18305R009), Abhishek Singh(18305R010)

System Model:

Relations are read from csv files and stored in memory area. Memory area is logically divided into morsels . Each thread executes on different morsel area. Final write to output hash table is written using compare and swap to avoid inconsistency.

Our Implementation:

dispatch queue: This is a linked list of pipeline jobs such as build/probe.

pipeline job struct: This stores a function pointer to the corresponding task. It also stores pointers memory areas (say 4 areas) where the input records for the job are located. Each area can be sliced into morsels. It also keeps track of the number of records processed in each memory area. A pointer to the corresponding query execution plan (qep) node is stored.

Worker threads: These threads continuously call the dispatch function in a loop.

Dispatch function: Looks at the job at the head of the dispatch queue. If there are unprocessed tuples in a memory area, then a slice of say 10000 tuples from the memory area (called a morsel) is processed. The task specified in the job struct is called and a pointer to the morsel is passed to it as an argument. Each thread processes morsels from a particular

memory area first. If that memory area is exhausted then the thread processes morsels from other areas (work stealing). Once a morsel is processed, info about the number of records processed for that memory area is updated in the pipeline job struct. If all memory areas for the job are processed, the job is removed from the queue and a new job is added (if dependencies are completed).

Dispatch queue synchronisation: Access to this queue is protected by a lock.

Query execution plan node : Used to track dependencies between tasks. If a node stores a pointer to a parent, it means that the parent is dependent on the child finishing execution first. A node stores the number of dependencies it has, as well as the number of dependencies completed so far. When a job is finished and removed from the dispatch queue, its parents dependency info is updated. If all its dependencies are done, the parent task is added to the dispatch queue.

Shared hash tables : This is mmaped by the main thread. One hash table for every build relation.

Build task: For every record in the morsel, a pointer to the record is inserted into a shared hash table for the corresponding relation. Insertion is made lock free using CAS (which is built-in in gcc).

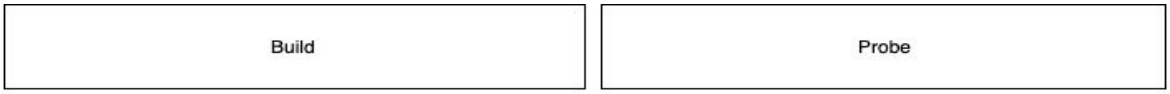
Probe task : For every record in the morsel, probes the hash tables of all the build relations. The pointers of all the matching records from all the hash tables are collected. For the given morsel record, all possible combinations with the collected records are generated. The joined tuples are stored in the threads local storage area.

Speeding up hash table lookups: In x86_64 the 16 MSB bits of the address are always 0. Every slot in the hash table can contain a chain. The 16 bits of the first pointer in the chain stores effectively a bloom filter for the elements present in the chain. During lookup if the corresponding bit is not set, the chain is not traversed.

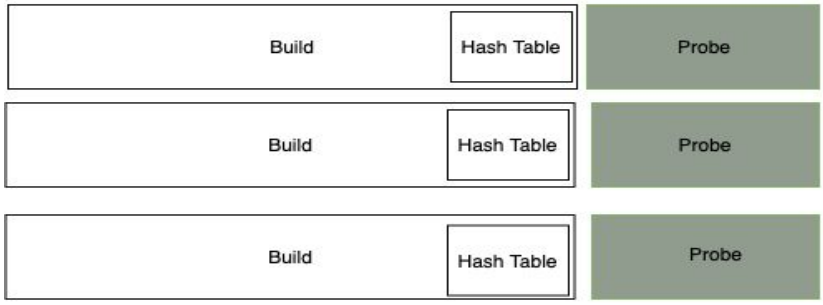
PostgreSQL Implementation:

In PostgreSQL 9, 10 each thread is building their own hash table for an inner table (no parallelism). For outer table, they are probing parallelly and hence reduce time in probing.
In our implementation, we have a shared hash table and each thread divide the work and work parallelly after hash table is complete then probing is done parallelly.

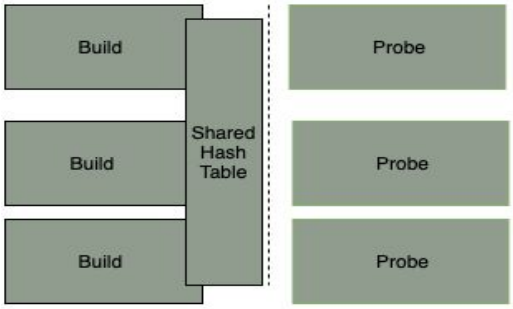
PostgreSQL 11 uses similar implementation which is used by us i.e probe and builds are in parallel.



Without parallel hash join (white color showing without parallelism)



In PostgreSQL 10 only probing is in parallel (gray color showing parallel operation) but not build



PostgresSQL 11 use shared hash table and probe and build are in parallel.

Correctness:

We have generated random relations up to 200000 tuples.
We show correctness of parallel hash join by joining exact same relation by parallel hash join and nested loop join and compare the results.

Evaluation:

Compare execution time for parallel hash join and nested loop join.

- Execution time of parallel hash join:
  For two relations: 33 ms
  For three relations: 130 ms
- Execution time for nested loop join
  For two relations: 135 se

Contribution:

- Neeraj Kerkar :Implementation of parallel hash join and performance optimization.

- Shah rukh Hussian: RnD on Postgres Implementation and implementation of hash table used in parallel hash join.

- Abhishek Singh: Performance and correctness testing.