HATCH: Hash Table Caching in Hardware for Efficient Relational Join on FPGA

Behzad Salami (1,2), Oriol Arcas-Abella(1,2) and Nehir Sonmez (1)
(1) Barcelona Supercomputing Center (BSC), Barcelona, Spain.
(2) Universitat Politecnica de Catalunya- BarcelonaTech (UPC), Barcelona, Spain.
Email: {behzad.salami, oriol.arcas and nehir.sonmez}@bsc.es.

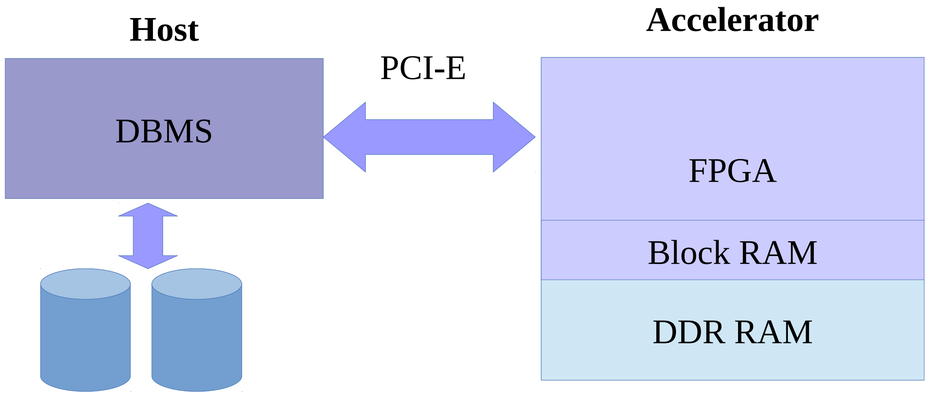
Introduction

Moore's Law: Size of data doubles at regular intervals, favoring the rise of Big Data and the Internet of Things.

Converting Raw Data into meaningful Information: Tools are needed to manage, organize and process data. Database Management Systems (DBMS) are the usual tools, for instance MS-SQL, Oracle, PostgreSQL, etc.

Bottlenecks in DBMS: Performance is a main bottleneck, especially in some operations like Hash Join. Some works report more than 40% of total time for join operations for TPC-H workloads. (TPC-H is a decision support benchmark widely used in database world.)

Solution: One solution is offloading the data from the host machine and accelerating the most time-consuming parts, using hardware devices like FPGAs. In this solution, time-consuming parts of DBMS are executed in the FPGA and data is transferred through a high speed interface, e.g. PCI-E. (In this work, we simulated the accelerator.)



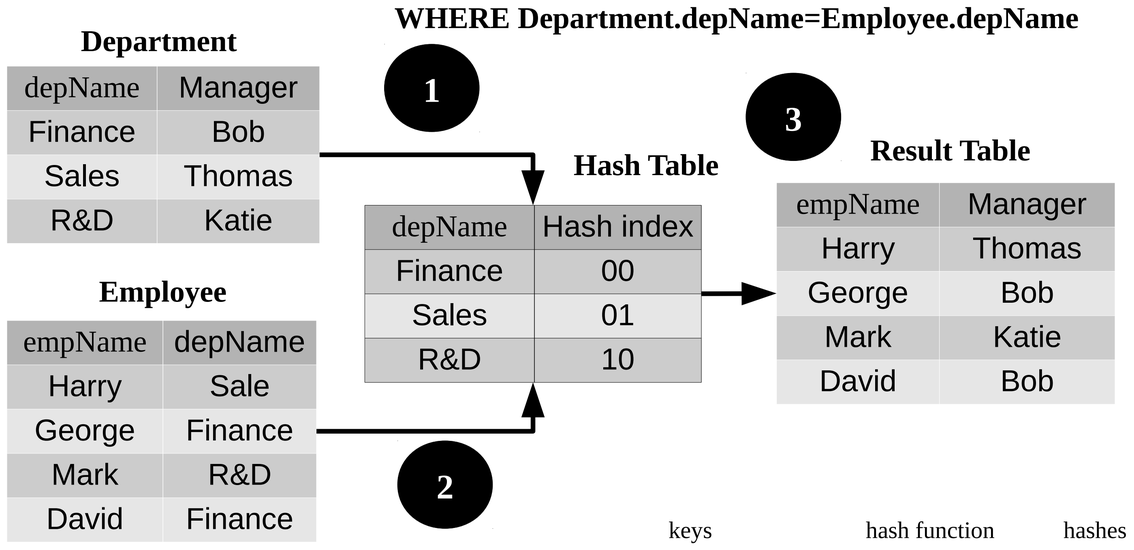
Hash Join

Hash Join: The most common implementation of join is hash join. In the hash join algorithm, the objective is to decrease the search space using a hash table. To index the table, a hash function is used.

Two Phase Hash Join Engine:

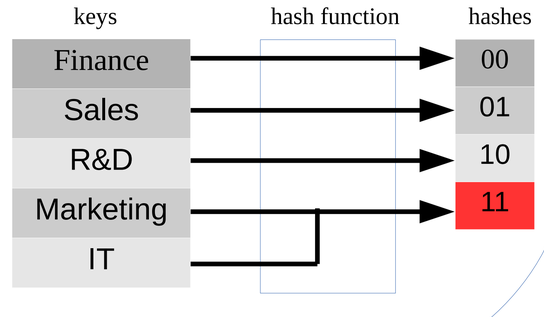
- Build Phase: To build the hash table, using the smaller input table (1).
- Probe Phase: To probe each row of the second table (2) and make the result table (3).

SELECT * FROM Department, Employee
WHERE Department.depName=Employee.depName



Perfect hash function: It makes unique hashes for unique keys.

Hash Collision: It happens when two different keys have the same hash indexes. For collision resolution, usually pointer chaining mechanisms are used.



Motivation

DDR RAM vs. Block RAM:

In the FPGA-Based hash join, the hash table is usually made in the DDR RAM and the on-chip RAMs (BRAM) are not considered. Consequently:

- DDR RAM is large enough to make big hash tables, but it is slow.
- Block RAMs are fast, but too small to make big hash tables.

Caching:

Caching (data or instructions) in the CPU world is a widely used technique to avoid data accesses from the main memory.

IDEA:

In this work, we apply the caching technique to the hash join operation, as it is one of the most time consuming operations in DBMS. The proposed technique avoids long latency memory accesses, by utilizing BRAM resources.

Hash Table Caching

How does our cache works?

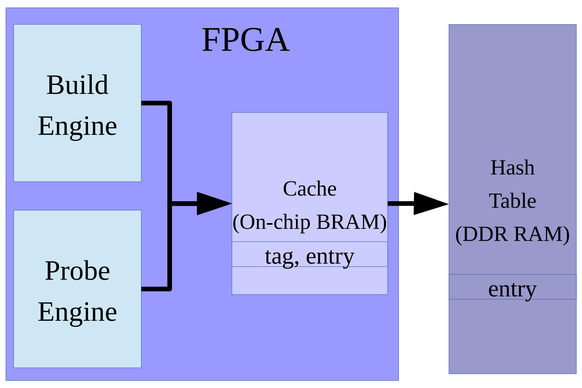
During the hash join (Build & Probe), all the required hash table entries are looked up in the cache first. If not found, the request is forwarded to the main hash table in DDR.

Cache Configuration:

Indexing of the cache: It uses the LSB bits of the hash indexes.

Cache Entry's format: It is same as the hash table entry's format, plus a tag.

Write into Cache: All the written hash table entries, in the build phase, and all the missed entries, in the probe phase, will be cached in BRAMs, using the direct-mapped replacement policy.



Simulation Results and Analysis

Simulation Environment:

Tools: Xilinx Vivado 2014.3, Bluespec 2014 (DDR Latency = 32 cycles).
Benchmarks: TPC-h (q03, q10, q12, q13, q14) for different sizes (1g, 10g, 100g).

Experimental Results (cycle numbers and speed up reports):

Cache hit rate ranges from 0% to 65.5%.
Speed up ranges from 1X to 2.8X, compared to baseline (without cache).

Simulation Results (number of clock cycles for cache-enabled and baseline version, speedup reports)

	1G Dataset			10G Dataset			100G Dataset		
Query	Baseline	Cache	Speedup	Baseline	Cache	Speedup	Baseline	Cache	Speedup
q03	330k	330k	1X	50M	17M	2.8X	896M	554M	1.62X
q10	110k	110k	1X	1.1M	1.1M	1X	11M	11M	1X
q12	4.89M	3.13M	1.5X	94M	42M	2.2X	1304M	631M	2.07X
q13	1.53M	1.53M	1X	91M	53M	1.7X	1422M	908M	1.57X
q14	270k	270k	1X	11M	6M	1.9X	350M	275M	1.27X

Analysis Results:

If the hash table is small enough to fit in the cache, then there is no need to access the memory (this happens in most of the 1G dataset benchmarks).
If there is no collision in build phase the accesses to DDR memory can be pipelined. Therefore there will be no speed up using BRAM to cache the hash table.

Consequently the speed up comes from hiding the memory latency in colliding keys, using caching for the hash table entries.

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Funded by:

European Union FP7 for AXLE Project, contract number: 318633.

Ministry of Economy and Competitiveness of Spain, contract number: TIN2012-34557.