A Locality-Aware Hash-Join Algorithm

Jian Fang*, Jan Hidders†, Koen Bertels*,
Jinho Lee‡, Peter Hofstee‡*

* *Delft University of Technology, The Netherlands*
† *Vrije Universiteit Brussel, Belgium*
‡ *IBM Austin Research Lab, USA*

ABSTRACT

The join is a commonly used operation in databases systems. As data volumes explode, join operations between two large relations become challenging. To overcome this challenge, some research adopts FPGAs (field programmable gate arrays) to accelerate this operation. However, increasing the bandwidth between accelerators and main memory provides additional opportunity for improvement. In this paper, we propose a locality-aware hash-join algorithm, and apply this to explore the potential of using FPGAs to accelerate hash-join operations in databases. The tuples in the input tables are partitioned into small buckets such that each bucket can fit into BRAM (Block RAM) of the FPGA. Hash-join probe operations are only performed on buckets with the same bucket number in these two tables. Analysis shows that the proposed method can take advantage of large memory bandwidth and thus provide a high throughput.

KEYWORDS: Databases; Join; FPGAs; Acceleration; Locality-Aware

1 Introduction

Even though the concept "big data" has been studied for years, data are still getting "bigger". A recent IDC report [IDC] shows that global data increased from 6.2ZB in 2014 to 8.6ZB in 2015, and it is expected to keep increasing at 40% per year, reaching 40ZB in 2020. To handle such large data requires computer systems to process data faster. FPGAs excel in domain-specific functions with high parallelism, and have been used as accelerators in the databases community [SMT$^+$12]. New hardware technology brings large available bandwidth between FPGAs and memory [OPE], breaking one of the main limitations of FPGA-based design. This provides FPGAs with new chances to enhance performance of computer systems.

To show the acceleration potential, this paper illustrates a method to accelerate databases using FPGAs. In databases, join operations between two or more large tables are widely used. Accelerating joins can improve the performance of the system. Naively speaking, performing joins is matching tuples in one table with tuples in other tables by a common value of the join keys. There are many join algorithms including nested-loop join, sort-merge join and hash-join. Among these algorithms, the hash-join is an efficient way due to a constant time complexity of accessing hash tables on a reasonable distribution of keys. Some research implements FPGA-based join algorithms, and gains speedups compared with the software algorithms. However, prior work did not deal with very large datasets due to the limitations of low memory bandwidth or long memory latency. In this paper, we propose a locality-aware hash-join algorithm to handle join operations between two large tables. In this algorithm, tuples in two tables are first divided into buckets that fit within BRAM. After the partition, build operations follow, building hash tables for each bucket. Finally, probe operations are run between two buckets in each table with a same bucket number.

The rest of this paper is organized as follows: Section 2 describes related work, while our proposed method is presented in Section 3. Section 4 discusses future work followed by conclusions in Section 5.

2 Join Algorithms

Running a join on two or more tables consists of combining tuples from these tables that have a common key. Among various join algorithms, the hash-join is illustrated to be one of the efficient join algorithms since it is a linear scalable algorithm. The simplest hash-join algorithm is the classical hash-join [KTMO83], which is a single node algorithm with two phases, build and probe. As we know, the complexity of this algorithm is $O(|R|+|S|)$, where $|R|$ stands for the number of tuples in table R, and $|S|$ stands for the number of tuples in table S. However, this can be improved by utilizing more processing elements. By dividing both input tables into portions and assigning them to p different workers, it can gain an ideal speedup of p, compared with the classical hash-join. However, this may introduce a large number of cache misses leading to a longer latency.

An efficient way to solve the cache miss problem is partitioning the relations to fit the size of the cache [SKN94]. Figure 1 gives an overview of this idea. The main idea of this algorithm is to add an extra phase to partition relations into small chunks with each size fitting in the cache by hashing on their key values before the build phase. Consequently, tuples in one bucket of relation R can only match tuples in one bucket with a same bucket number of relation S. Thus, the hash table of one bucket can be stored in the cache, leading to reduce cache misses. An improved algorithm, radix hash-join, further splits the partition phase into multiple passes to reduce the possibility of TLB (Translation Look-aside Buffer) misses introduced by the partition phase.

3 Architecture

An overview of the proposed architecture is shown in Figure 2. There are three main components including the partition engine, the build engine, and the probe engine, representing the three phases of this algorithm. All these three components are implemented in the FPGA and connected to memory.

Our algorithm is described in Figure 3. It contains three phases which are the partition phase, the build phase, and the probe phase. In the partition phase, all tuples in table R are scanned and assigned to different buckets using a partition hash function. The hash function is used to partition the data and assign the bucket number. The same partition operation is

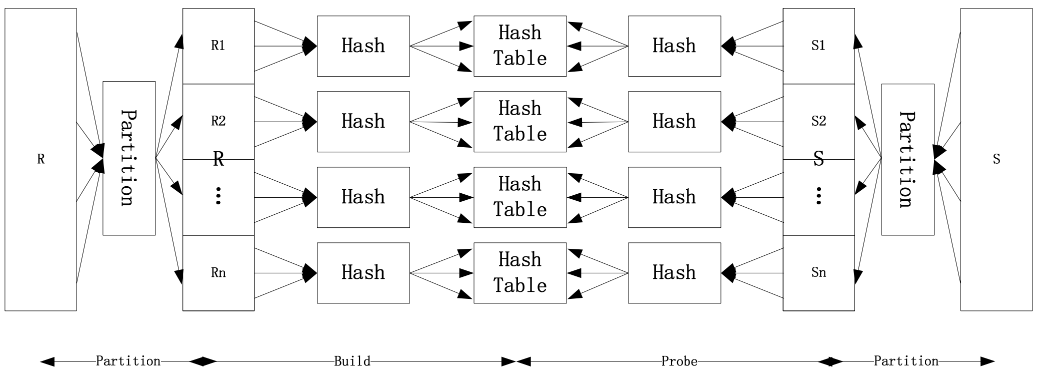

Figure 1: Cache Conscious Hash-Join.

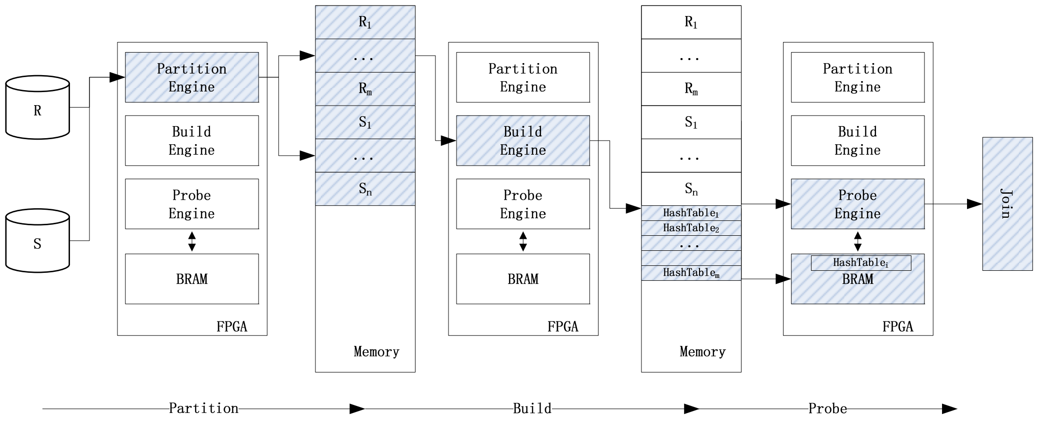

Figure 3: Three Phases of the Locality-Aware Hash-Join Algorithm.

run on table S. To avoid long main memory latencies in the build phase and the probe phase, we partition data into buckets that fit into BRAM. Consequently, the hash table of a whole bucket can be stored inside the FPGAs. The build phase performs a hash function on tuples in table R bucket by bucket to build their hash tables. Each bucket has its own hash table, and this table will be read and stored in BRAM. The probe phase is also performed at a bucket granularity. The tuples in bucket i of table S only probe the hash table of bucket j, where bucket i and bucket j share a same bucket number. So the whole hash table is stored in BRAM, reducing the number of memory accesses when not using cache or partitioning data in a larger size.

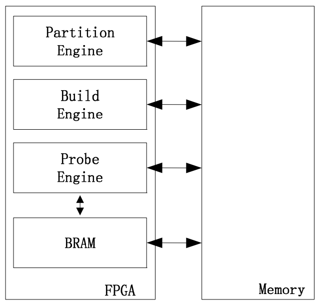

Figure 2: Overview Architecture

When designing an FPGA-based algorithm, we need to consider all the resources in FPGAs including the logic resource, buffers, memory bandwidth, and latency. Among these constraints, buffers and memory latency are two main limitations. Our proposed method partitions data into a size that meets the BRAM limitations. When having a larger BRAM, we use a larger granularity, leading to less buckets. Utilizing BRAM also shows benefits from low latency and large bandwidth. As we know, the latency of accessing memory is much larger than that of accessing BRAM. Our method uses BRAM to store hash tables, resulting in a low latency in the probe phase.

4 Future Work

Our algorithm aims to take advantage of the large available main memory bandwidth. Benefit from this is shown in reducing the time for transferring data between FPGAs and memory. However, there are some remaining issues that need to be resolved, which we intend to address in future work.

- Handling hash collisions. The two most commonly used techniques for this are *separate chaining with linked lists* and *open addressing*. We want to investigate which type of solution is efficient when using the FPGA. Since our algorithm stores the whole hash table for a bucket and processes probes inside the FPGA, open addressing should be an more effective choice. This is because it can save more BRAM for a larger hash table since it does not need to represent there the linked lists.
- Design an efficient parallel algorithm that performs multiple partitions, builds, and probes. A simple way to scale this algorithm into a parallel one is having more engines. However, we should ensure that each engine reads and writes the right data.
- Increases of main memory bandwidth change the design of using FPGAs. Two important questions are how to trade off between different resources and how to make full use of the larger memory bandwidth.

5 Conclusions

In databases, the join is a commonly used operation. Enhancing the join performance can improve the computation capacity of the whole computer system. In this paper, we propose a locality-aware hash-join algorithm that partitions data into fragments whose size matches the size of BRAM. We expect that our proposed method can achieve a high performance, but many aspects need to be researched further.

References

[IDC] `http://www.idc.com`.

[KTMO83] Masaru Kitsuregawa, Hidehiko Tanaka, and Tohru Moto-Oka. Application of hash to data base machine and its architecture. *New Generation Computing*, 1(1):63–74, 1983.

[OPE] `http://openpowerfoundation.org/presentations/brad-mccredie-board-advisor-ibm/`.

[SKN94] Ambuj Shatdal, Chander Kant, and Jeffrey F Naughton. *Cache conscious algorithms for relational query processing*. University of Wisconsin-Madison, Computer Sciences Department, 1994.

[SMT$^+$12] Bharat Sukhwani, Hong Min, Mathew Thoennes, Parijat Dube, Balakrishna Iyer, Bernard Brezzo, Donna Dillenberger, and Sameh Asaad. Database analytics acceleration using FPGAs. *Proceedings of the 21st international conference on Parallel architectures and compilation techniques*, pages 411–420, 2012.