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

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

Abstract—In this paper we present HATCH, a novel hash join engine. We follow a new design point which enables us to effectively cache the hash table entries in fast BRAM resources, meanwhile supporting collision resolution in hardware. HATCH enables us to have the best of two worlds: (i) to use the full capacity of the DDR memory to store complete hash tables, and (ii) by employing a cache, to exploit the high access speed of BRAMs. We demonstrate the usefulness of our approach by running hash join operations from 5 TPC-H benchmark queries and report speedups up to 2.8x over a pipeline-optimized baseline.

I. Introduction

Recently, with the rise of Internet of Things and Big Data, faster query processing capabilities has gained significant attention. One of the most time-consuming query operations is the table join. Previous studies have demonstrated that table joins can account for more than 40% of total execution time [1]. The hash join consists of a build phase using the smaller table to make the hash table and a probe phase where all keys in the larger table are probed against the hash table for matches.

II. HATCH: Proposed Design

The most critical issue in table joins is hash collisions. They happen when the hash function hashes different keys to the same index. Collisions are inevitable and need to be handled appropriately. This issue may convert into a bottleneck if it's handled in a non-optimized manner like software fallback mechanisms [2] which can cause extra latencies. Due to the scarcity of on-chip BRAM resources, previous FPGA implementations envisioned building the hash table in high-latency DDR memory. In this paper we propose HATCH, a novel hash join engine that works fully in hardware. HATCH better utilizes the FPGA resources by employing caching of hash table entries in BRAM, meanwhile resolving hash collisions in hardware.

The main components of HATCH are; hash table is made in RAM, cached hash table in BRAM, buckets to store value of tuples, a linear feedback shift register (LFSR)-based hash function and a content addressable memory (CAM) to handle read-after-write hazards in the build phase. In the proposed architecture, bucket overflow issues and nested collided keys are handled by employing a linked list structure. In HATCH, BRAM is used to cache some entries of hash table. All the required look-ups for the hash table go to the cache in BRAM first. If it is a miss, the look-up is forwarded to the main hash table in RAM. Additionally cache entry replacement policy (in both build and probe phases), is considered for all the entries which has missed in the cache.

Table I
Experimental Results for TPC-H Queries

Query	Size (#Tuples)	Baseline (#Cycles)	HATCH (#Cycles)	Speed up
q03	7.50 M	50.2 M	17.7 M	2.8 X
q10	1.10 M	1.10 M	1.10 M	1.0 X
q12	15.3 M	94.9 M	42.6 M	2.2 X
q13	30.0 M	91.8 M	53.6 M	1.7 X
q14	2.70 M	11.6 M	5.90 M	1.9 X

We have used Xilinx ISE version 14.1 and Bluespec System Verilog compiler[3]. Our engine was design to work at 200 MHz on a VC709 Evaluation board with a Virtex-7 xc7vx690tffg1761-2 FPGA and two DDR3 memories of 4 GB each. Simulation results for some TPC-H queries, are reported in Table I for 10G dataset. In the baseline version, BRAM is not employed to cache the hash table entries but its other components are same as HATCH.

III. Acknowledgement

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013), for Advanced Analytics for Extremely Large European Databases (AXLE) project under grant agreement number 318633, and from the Ministry of Economy and Competitiveness of Spain under contract number TIN2012-34557.

References

[1] T. Hayes, O. Palomar, O. Unsal, A. Cristal, and M. Valero. Vector extensions for decision support dbms acceleration. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 166–176, 2012.

[2] L. Woods, J. Teubner, and G. Alonso. Less watts, more performance: An intelligent storage engine for data appliances. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1073–1076, 2013.

[3] Bluespec, Inc. http://www.bluespec.com.