Probing Based Hash Table Accelerator

Yaodong Sheng
Lehigh University
Bethlehem, PA, U.S.A
yas616@lehigh.edu

Zhankai Feng
Lehigh University
Bethlehem, PA, U.S.A
zhf218@lehigh.edu

Anlan Yu
Lehigh University
Bethlehem, PA, U.S.A
any218@lehigh.edu

ABSTRACT

Hash table is a widely used data structure in many applications. However, hash value lookup is a time consuming problem because of multiple-time index comparison and it cannot be avoided. In this case, following instructions dependent on this instruction will be delayed and the overall execution efficiency cannot be guaranteed. In this proposal, we design a hash lookaside buffer (HLB) for hash value lookup to increase the throughput as well as decrease the latency of operations on probing hash table. Used as a buffer dedicated for hash table, HLB fetches the values from L1 cache. Without fetching an entire cache line in L1 cache, values are fetched to HLB only as use bit is 1 inside cache line. In this case, several values can be fetched to HLB simultaneously and unrelated data will not be fetched to HLB, which increases MLP. When doing hash lookup, one can directly use key to compare the key value pair to find the data in HLB. Different from traditional index lookup, HLB provides 4-way lookup simultaneously. In this way, the efficiency of hash lookup will be highly improved.

CCS CONCEPTS

• **Computer Architecture → Hash Accelerator**.

KEYWORDS

hash table, accelerator, table lookup

ACM Reference Format:
Yaodong Sheng, Zhankai Feng, and Anlan Yu. 2019. Probing Based Hash Table Accelerator. In *ECE 401: Advanced Computer Architecture, Sep. - Dec., 2019, Bethlehem, PA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Alg.1 shows the pseudo code for traditional hash lookup. We can notice that there is a loop to compare the two indexes. Every time when hash lookup is executed, program has to do index comparison for several times. One can imagine that the efficiency cannot be promised.

Based on the drawback of traditional hash lookup, we propose to increase the throughput while decrease the latency of operations on probing based hash table. We add a special unit called HLB in following article which is between CPU and L1 cache. The working environment is in 5-stage RISC pipeline CPU and the normal DRAM.

Algorithm 1 Hash Lookup Pseudo Code

```
1: $Vector < T > hash\_table$
2: $index \leftarrow hash(key)$
3: while $hash\_table.index\ != index\ \&\&\ index\ != hash\_table.size()$ do
4:     $index++$
5: end while
6: if $index == hash\_table.size()$ then
7:     $return\ -1$ // key not exist
8: else
9:     $return\ index$
10: end if
```

1.1 Instruction Set

HLB is used to fetch the value of a corresponding key in hashmap which is a pointer to the real data. Instead of going to L1 cache to fetch the value from cache line, now such data is fetched from HLB directly. Only when the HLB miss happens, cache will be utilized to refill value to the HLB. The ISA should be expanded to identify such instructions and the electric logic in IF and ID stages should also be modified to translate the 'hash' instructions correctly.

1.2 Data Structure Description

1.2.1 Hash Table. The hash table is a vector, each item is combined with key value pair and a use bit. The value is a pointer to the actual data. Each cache line is exactly filled with several number of items, so that HLB can easily determine which items in a cache line are valid by comparing the use bit.

1.2.2 Data Store. The data store is used to store the real data, and is implemented as a vector. The data type of each item is the same as hash table's value type.

1.3 Data Flow Description

Given a key, CPU calculate the index from the hash function also its virtual address and then search from the HLB to check if this virtual address exists, in case it is not a 'miss', then the HLB will compare this item's key with the given key. At the mean time 3 following addresses are also compared in parallel by HLB which means with one instruction for possible items are checked simultaneously. If either of them matches, then it means the key-value pair is found return the pointer, otherwise, return -1. If the given key is not found, CPU will check next four following address after the 4 address compared in the last iteration.

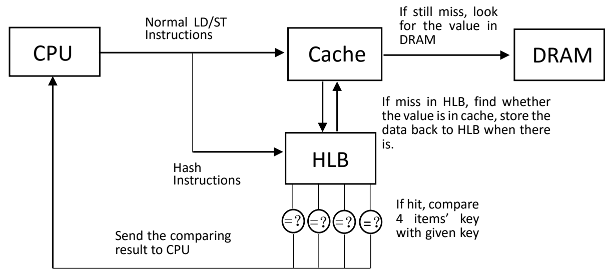
CPU
Normal LD/ST Instructions
Cache
If still miss, look for the value in DRAM
DRAM
If miss in HLB, find whether the value is in cache, store the data back to HLB when there is.
HLB
Hash Instructions
If hit, compare 4 items' key with given key
=?
Send the comparing result to CPU

Figure 1: Data Flow.

2 LITERATURE REVIEW

2.1 Hardware Accelerator for Software Data Structure [2]

The main idea of this paper is to exploit MLP (memory level parallelism) on data-centric applications (e.g iterate over vector) which addresses the energy efficiency problem. To achieve MLP, the hardware accelerator has its own 4 stage pipelines (PE) which uses 'key' to fetch the element that will be operated from a specific cache called 'obj-store', the obj-store is refilled by the management of 'collector'. The 'collector' sends prefetch request to DRAM and refill the LLC (last level cache) which overlaps pipeline execution time. So the collector is responsible to generate memory address and refill 'obj-store'. Once PE starts running, it will unroll the iterative instructions and do operations on it, obj-store ensures that there would never be cache misses.

2.2 Fine Grain Checkpointing In-Cache-Line-Logging[1]

The main idea of this paper is to utilize the 'natural atomicity' by putting 'data + log' in one cache line, although external log may be utilized to record the whole object for complex situation (which is rarely happens). On the other hand, instead of using transaction, it uses frequent checking point for recovery and crash consistency. Combining the idea of 'in cache line log + external log' and 'frequent checking point', it can make the operations on data structure easy to be persistent and recover after crash.

3 PROCEDURE

3.1

Determine the new ISA, and according to it modifying the electric elements in ID and IF stage so that make CPU realize 'hash' instructions coming.

3.2

Design the hash table data structure to maximum utilize the cache line.

3.3

Determine the HLB structure.

3.4

Determine the logic between HLB with cache and CPU.

3.5

Describe the whole architecture by Verilog language.

3.6

Simulate the performance by ModelSim.

REFERENCES

[1] Nachshon Cohen, David T Aksun, Hillel Avni, and James R Larus. 2019. Fine-Grain Checkpointing with In-Cache-Line Logging. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 441–454.

[2] Snehasish Kumar, Naveen Vedula, Arrvindh Shriraman, and Vijayalakshmi Srinivasan. 2015. DASX: Hardware accelerator for software data structures. In *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 361–372.