ECE552 Lab Assignment 4: Data Caches Report

Student: Daokun Chen 998699813 Ruyu Fan 999586363

Question 1:
In order to verify the correctness of the next-line prefetcher, a simple microbenchmark are created. The microbenchmark has a char array with the total size of 1000000 * stride. The stride is specified by the user input, which defines the data access stride on the array. The cache configuration used is the original next-line.cfg (64 sets, 2-way, 64 bytes per block and LRU replacement policy). If the stride is set to 1-64 (within a block), the miss rate will be very close to 0.0% (cold misses are inevitable). If the stride is greater than 64, the miss rate will increase drastically.
mbq1.c output:

Stride	40	100	300
L1 Miss Rate	0.0%	28.07%	49.71%

Question 2:
The same method and configuration as Q1 are used to verify the correctness of the stride prefetcher. The output are expected to be close to 0.0% for any stride value because the stride prefetcher are capable of catching any equal stride data access pattern. However, if the stride value are determined randomly for each data access and the rpt table size is set to 4, the miss rate will increase.
mbq2.c output:

Stride	40	100	300
L1 Miss Rate	0.0%	0.0%	0.0%

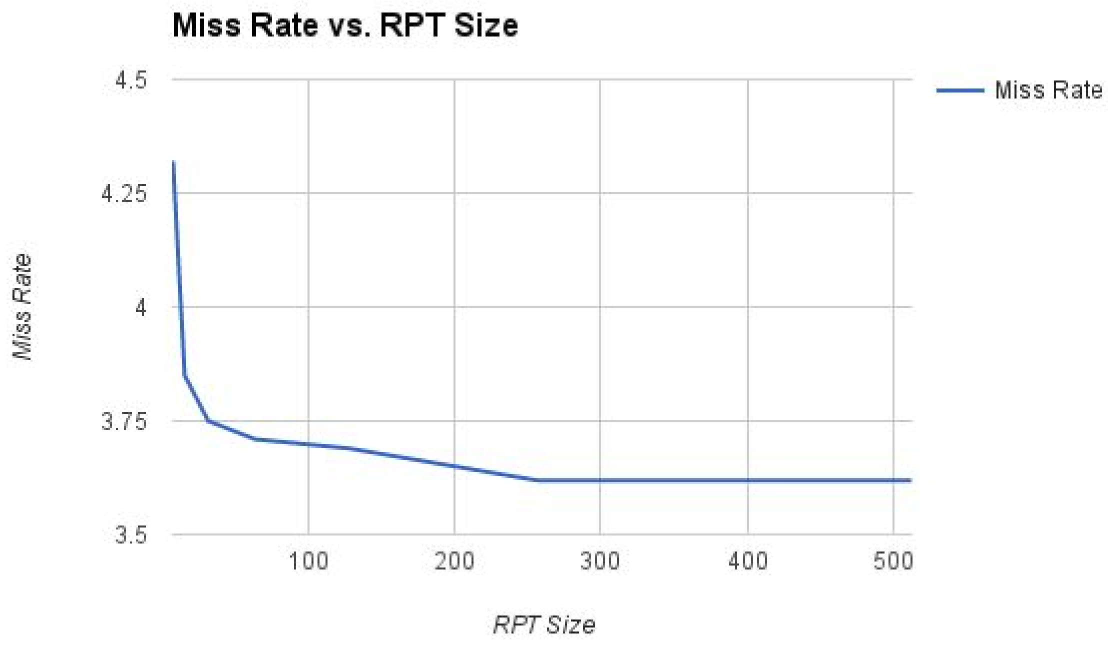
Question 3:

Configuration	L1 Miss Rate	L2 Miss Rate	Avg. Access Time
No Prefetcher	4.16%	11.40%	1.89
Next-Line Prefetcher	4.19%	8.38%	1.77
Stride Prefetcher	3.85%	5.78%	1.61

T(avg) = T(access-L1) + (L1 Miss Rate) * (T(access-L2) + (L2 Miss Rate) * T(hit-memory))

Question 4:

RPT Size	8	16	32	64	128	256	512
Miss Rate	4.32%	3.85%	3.75%	3.71%	3.69%	3.62%	3.62%



Question 5:
We would like to know the useful rate of the prefetched blocks, which is for all prefetched blocks, how many of them are later accessed rather than replacing out the useful blocks. Since a bad prefetcher may kill the performance if it prefetches useless blocks. This statistics would be very helpful for evaluating prefetchers.

Question 6:
To verify the correctness of the open-ended implementation, we use nested for loops to create an organized data access that can be captured by the Czone prefetcher. Since the open-ended prefetcher is a simple stride prefetcher coupled with a Czone prefetcher, and the Czone prefetcher only prefetches when the non-prefetch cache access misses, in other words, cold misses are inevitable. Hence, we compared the open-ended prefetcher design with the single stride prefetcher. The performance does improve dramatically.
mbq6.c output:

Configuration	Open-ended Prefetcher	Stride Prefetcher
L1 Miss Rate	1.15%	6.08%

Open-ended Prefetcher Design:
The open-ended prefetcher we implemented is based on the idea of CZone Delta Prefetching[1]. The global history buffer(GHB) is a link list used to store the most recent cache miss addresses, the index table entry holds pointer that points to the corresponding miss address. The tags are the highest 8 bits of miss address value and the middle 8 bits are used for indexing (representing as the same CZone, here, each CZone is 2^(32-8-8) = 65536). Within the GHB, those entries with the same Czone are linked based on program order. Whenever a cache miss was encountered, it will be inserted at the GHB head, and the head of the same Czone chain as well. Then, a delta buffer will be calculated based on the address difference between each consecutive miss addresses. After the delta buffer has been constructed, the most recent 2 delta will be moved to the correlation key register, and the entire delta buffer will be shifted into the correlation comparison register to compare with the correlation key register. To compute the sequence of prefetch address, the delta values are consecutively added to the miss address. Once there is a correlation hit, the following miss address will be computed, and the corresponding blocks will be prefetched. Figure 1. demonstrates the basic hardware of this implementation.

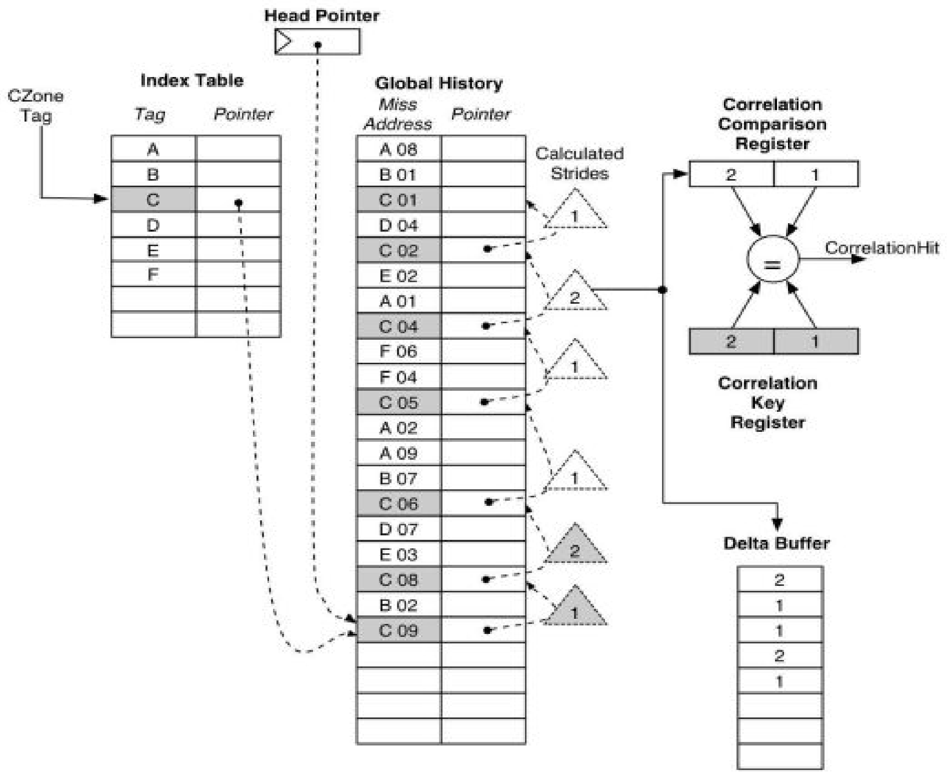


Figure 1: CZone Prefetcher hardware[1]

In addition, this CZone prefetcher prefetches blocks only when cache access miss, this is works not very well in the equal stride data access case because it does not prefetch ahead when the cache hits. Hence, we add a stride prefetcher to handle this situation. When the RPT entry is in steady state, the program prefetches based on the stride prefetcher, otherwise, use the CZone prefetcher.

Performance Summary:

Config / trace	compress	gcc	go
Open-ended Miss Rate	3.70%	1.23%	1.13%

Open-ended Configuration Size:
Stride Prefetcher Size:
16 RPT Table entries * [(64 - 8 - 3) Tag bits + 32 prev addr bits + 32 stride bits + 2 state bits] = 1904 bits = 238 bytes
CZone Prefetcher Size:
256 GHB entries * [32 miss address bits + 32 link list bits] + 256 Index Table entries * [8 Czone Tag bits + 32 link list bits] + 2 * 32 bits correlation key registers + 2 * 32 bits correlation comparison registers = 26752 bits = 3344 bytes

Total Size = 3582 bytes

Open-ended Feasibility:
CACTI Simulation Result:
Tag side (with output driver) (ns): 0.155839
Total dynamic read energy/access (nJ): 0.000382118
Total leakage read/write power of a bank (mW): 0.185118
Tag array: Area (mm2): 0.000855706; Height (mm): 0.0334152;
Width (mm): 0.0256083

This design is realistic because the hardware described in Figure 1. is implementable and the total size is reasonable.

Work Breakdown:

Daokun Chen	**Ruyu Fan**
Open-ended Prefetcher	Nextline & Stride Prefetcher
Report Writing	Report Writing
Microbenchmarks	Microbenchmarks

Reference:
[1] Kyle J. Nesbit, Ashutosh S. Dhodapkar, and James E.Smith "AC/DC: An adaptive Data Cache Prefetcher" Accessed on Nov, 23, 2015:
http://www.eecg.utoronto.ca/~moshovos/ACA05/read/AC_DC.pdf