Yi Liu (999736889)
Daiqing Li (1000795357)

# ECE552 Lab 4: Data Caches

**Question 1:**

**Provide a micro-benchmark and a configuration file that you use to verify that your implementation of the next line prefetcher is correct. Explain your choice.**

In our implementation of next line prefetcher, we first use cache-probe to test if next line block is already fetched in the cache, if it is not, we prefetch it. In our microbenchmark, we allocate a large array and in a loop, we first access an element in the first cache line, and then access the data in the second cache line. Since cache line size is 64 bytes, we make sure we access next 16 element at this time. After running 1000000 times, we should get a very high hit rate because data in the second cache line is already prefetched by the first access.

| | Baseline | nextline prefetcher |
|---|---|---|
| dl1.hits | 47749949 | 47999951 |
| dl1.misses | 250011 | 9 |
| dl1.miss_rate | 0.0052 | 0.0000 |

**Question 2:**

**Provide a micro-benchmark and a configuration file that you use to verify that your implementation of the stride prefetcher is correct. Explain your choice.**

In our implementation of stride prefetcher, we preallocated a RPT table when creating cache and initialized the data structure. We use PC address as index, and after filtering last few zero bits, we choose corresponding #bits as index based on the size of RPT defined in config file. Following the mechanism of RPT table mentioned in the handout, we do prefetching based on current state. In our microbenchmark, we allocate a large array and access it in a constant stride. Since this pattern will be catched by stride predictor, after a long run, we should get a very high hit rate. For the config file, we use RPT table size of 16.

| | Baseline | stride prefetcher |
|---|---|---|
| dl1.hits | 34378585 | 35003776 |
| dl1.misses | 625217 | 26 |
| dl1.miss_rate | 0.0179 | 0.0000 |

Yi Liu (999736889)
Daiqing Li (1000795357)

**Question 3:**
**Using the configuration files provided with the simulator and statistics collected from the simulator, estimate the average memory access time for data accesses for benchmark *compress* for the configurations with no prefetcher, L1 data next line prefetcher and L1 stride prefetcher. In your calculations, assume the following hit times:**
$T_{access-L1\ Data}$ **= 1,** $T_{access-L2}$ **= 10,** $T_{hit-Memory}$ **= 100.**

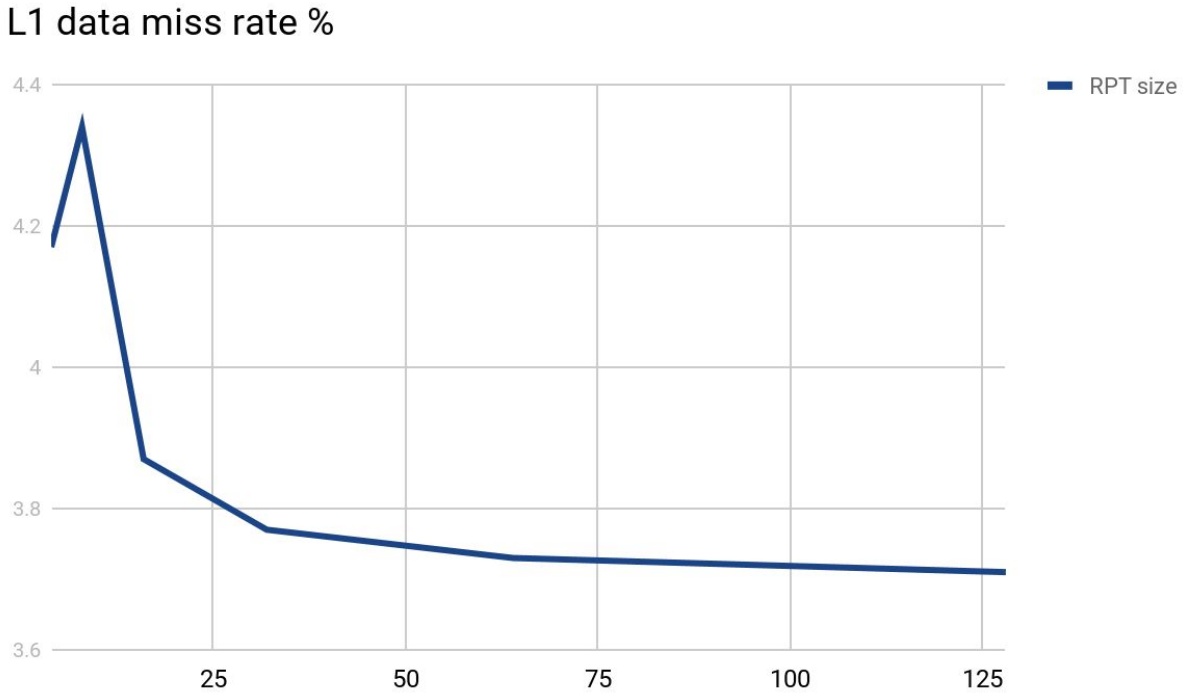
| Config | L1 Miss Rate | L2 Miss Rate | Avg. Access Time |
|---|---|---|---|
| Baseline | 4.16 % | 11.40 % | 1.89 |
| Next-line prefetcher | 4.19 % | 8.38 % | 1.77 |
| Stride prefetcher | 3.85 % | 5.78 % | 1.60 |

Tavg = T access-L1 Data + (L1 miss rate * (T access-L2 + L2 miss rate * T hit-Memory))

**Question 4: For benchmark *compress*, study the performance of the stride prefetcher when varying the number of entries in the RPT. Use the configuration files provided, changing only the number of entries in the RPT. Provide a graph that plots on the x axis the number of entries in the RPT and on the y axis a metric of your choice that measures the performance of the prefetcher. Explain your graph and your conclusions.**

| RPT size | L1 data miss rate |
|---|---|
| 4 | 0.0417 |
| 8 | 0.0434 |
| 16 | 0.0387 |
| 32 | 0.0377 |
| 64 | 0.0373 |
| 128 | 0.0371 |

Yi Liu (999736889)
Daiqing Li (1000795357)




**Question 5: If you were asked to include more statistics in the sim-cache simulator to study the performance of prefetchers in general, which statistics you would consider adding? (No implementation necessary, only an explanation is sufficient.)**

We will add prefetch coverage to count how many misses are covered by prefetching. And prefetch accuracy to count if prefetching evicts useful data. Those two statistics are useful to evaluate prefetching performance.

**Question 6:**

**Provide a micro-benchmark and a configuration file that you use to demonstrate the performance of your open-ended prefetcher. Explain your choice.**

In this microbenchmark, we allocated a large array and try to access it with different strides alternatively. Since we use pc addr as index and we access this index by different strides each time, in a stride prefetcher, it cannot catch such a two strides access pattern. But in our open-ended prefetcher, we can catch such a pattern, therefore comparing the baseline program, we improve the hit rate a lot.

| | Baseline | Open-end prefetcher |
|---|---|---|
| dl1.hits | 44166623 | 44444400 |
| dl1.misses | 277788 | 11 |

| | | |
|---|---|---|
| dl1.miss_rate | 0.0063 | 0.0000 |

**Description of open-end prefetcher implementation**

While the stride prefetcher makes predictions based on a constant stride in between successive memory accesses, the open-ended prefetcher uses a delta-correlation prediction table in attempting to detect a pattern in the strides of memory access sequences. The prefetcher keeps a circular buffer that records the history stride values for the memory access instructions which are indexed to the entries of delta-correlation prediction table. The prefetcher looks for a pattern in reverse order based on the two most recent strides. If there is a match in the buffer of this pair of strides, the prefetcher issues a series of prefetches for the addresses generated by incrementally adding each of the stride values found after the matched pattern to the current memory address. (Note: strides are referred to as deltas in the article) This process of generating the candidate prefetch addresses is known as delta correlation and after that prefetch filtering will check for duplicates in the generated list of candidates. The resultant candidates list will be used to do prefetching.

**Total size in memory for hardware implementation:**

The delta correlation prediction table has a total of 64 entries.

Fields of an entry of delta correlation prediction table:

**PC**: tag for the delta correlation prediction table

Because the last 3 bits of the PC is always 0 for alignment and an additional 8 bits of the PC is used for indexing, only 32 - 3 - 8 = 21 bits of the PC is used for tag

**Last address:** the last memory location accessed by this instruction [32 bits]

**Last prefetch address:** the last address in the traversal of the candidates list. It is only used in the control of the prefetcher hardware.

**Circular buffer:** 8 entries buffer for the recorded history strides [8 * 32 bits = 256 bits]

**Pointer:** the index to the next element in the circular buffer to be inserted [3 bits]

Size of each entry of the delta correlation table = 21 + 32 + 256 + 3 = 312 bits = 39 bytes

Total size needed by the prefetcher = 64 * 39 bytes = 2496 bytes = 2.4375 KB

CACTI values for the open-ended prefetcher:

| | |
|---|---|
| Area | 0.110674 mm x 0.130006 mm |
| Latency | 0.31951 ns |
| Leakage power | 0.999449 mW |

Yi Liu (999736889)
Daiqing Li (1000795357)

**Work completed by each partner**

Daiqing Li:

Microbenchmark for q1, q2, q6, next-line prefetcher, stride prefetcher, preparing for report

Yi Liu:

Checked and verified correctness of next-line and stride prefetcher. Implemented the open-ended prefetcher and consolidated report