ECE552 Lab 4 : Cache Prefetching

Yi Fan Zhang 1000029284 Jianwei Sun 1000009821

Question 1, 2, 6: Microbenchmarks

Q1: Next-Line Prefetcher

The microbenchmark aims to test the prefetcher using two cases, when it always fails (Case 1), and when it always succeeds (Case 2).

Case 1: The memory access will always be outside the range of the next-line prefetcher. As our prefetcher only fetches the next block size, this case accesses memory 2 block sizes away, causing the prefetcher to miss every time.

Case 2: The memory access will be within the next 1 block-size, so the next-line prefetcher will always accurately fetch the right memory.

We iterated Case 1 100k times, and Case 2 200k times, expecting 100k misses and 200k hits. Our simulations showed 200,343 hits and 100,009 misses, thus verifying our prefetcher

Q2: Stride Prefetcher

The microbenchmark tests two cases of this prefetcher:

Case 1: The memory access is done in an alternating sequence of 1 and 2 block-sizes using the same instruction. As a result, a consistent “stride” cannot be established, causing the Stride Prefetcher to not be able to fetch any correct memory results.

Case 2: We will still fetch unequal strides of alternating 1 and 2 block sizes, each respective memory access is from a different instruction. Because of the RPT, each instruction can keep track of their own strides, and within these entries, a constant stride is established. Thus, “isolating” each block size jumps to their respective instructions, the stride prefetcher can accurately fetch the correct instructions.

We iterated Case 1 to access memory 133k times, and Case 2 400k times, expecting 133k misses and 400k hits. Our simulations showed 400,332 hits and 133,353 misses, thus verifying our prefetcher.

Q6: Open Ended Prefetcher: A GBH-based Prefetcher

The microbenchmark tests three cases of this prefetcher, 2 of which are expected to miss at all times.

Case 1: Similar to Case 1 of Stride Prefetcher, memory access is done in a pattern of 1, 1, 2 block sizes using the same instruction. While this fails Stride Prefetcher, the GBH keeps track of patterns, thus is immune to such alternating changes. In this case, the prefetcher should always hit.

Case 2: Two instructions will access memory at different strides starting from the same memory location. I.E, instruction I will access at 2 block-size intervals, and instruction II will access at 1 block-size intervals. Because of the different “speeds”, the difference between these memory accesses on the GBH will always change. The difference between the “slower” instruction and the “faster” instruction will always increase. Thus, no clear pattern on the GHB exists, and our prefetcher will not be able to

predict anything accurately

Case 3: This case jumps in a pattern of 1 1 1 1 2 block sizes. Because the GHB only keeps track of the last 5 histories, and index tables only remembers the last two histories, the prefetcher will match the pattern in the index table with the latest “memory delta” patterns on the GHB. For example, the prefetcher will see a delta of 1 1 and expect a delta of 2 next, when 4/5 of the time, the next delta is a 1. This would result in a hit only 1/5th of the time.

We expect Case 1 accesses memory 150k times, producing 150k hits, Case 2 accesses memory 200k times, producing 400k misses, and Case 3 accesses memory 1M times, producing 200k hits and 800k misses. Overall, we expect 350k hits, and 1.2M misses. Our simulation showed 350,429 hits, and 1,199,923 misses

Question 3

Average Memory Access Times for Three Prefetcher Configurations for Compress Benchmark

Configuration	L1 Miss Rate	L2 Miss Rate	Average Access Time
Baseline	0.0416	0.114	1.801
Next-line Prefetcher	0.0416	0.114	1.801
Stride Prefetcher	0.0421	0.1109	1.799

It is told that a L1 access takes 1 unit of time, an L2 access takes 10, and a memory access takes 100. With these hit times, the following formula was used to calculate the average access time for each of the pre-fetcher configurations.

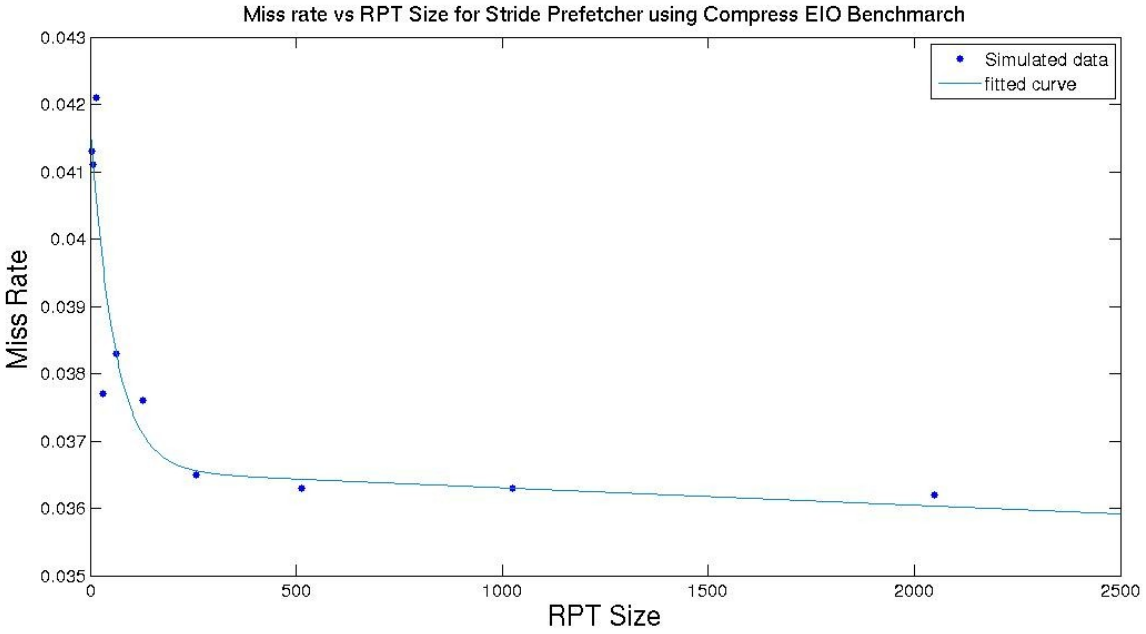
Average Access Time = 1*(1-L1_Miss_Rate) +10*(L1_Miss_Rate)*(1-L2_Miss_Rate) +
100*(L1_Miss_Rate)*(L2_Miss_Rate)

To calculate the average access time, the expected value for all possible hit/miss outcomes must be calculated. If there is an L1 hit, then the access time will be 1 time unit with rate (1-L1_Miss_Rate). However, an access time of 10 units will occur when L2 hits given that L1 misses. This scenario occurs with rate L1_Miss_Rate*(1-L2_Miss_Rate). Finally, the scenario that both caches miss will result in a memory access of 100 time units of latency. This case occurs with rate L1_Miss_Rate*L2_Miss_Rate.

Question 4

The RPT size of the Stride Prefetcher is varied between 4 to 2048 entries to check how the performance of the preference relates to its table size, using the compress benchmark. The miss rate is used as a metric to determine performance, as cache miss and hit rates directly affects the overall speed of memory access instructions of the CPU. The lower the miss rate, the better the performance.

We see that an almost exponential curve can be fitted to the simulated results. The performance improves quite drastically between table sizes of 4 and 256. However, after 256 entries, the table levels off, seeing almost no difference in performance at higher table sizes. This is most because tables bigger than 256 entries already accommodates almost all the instructions with memory access in the program, and further expansion of its size will only be unused space.



Question 5

A statistic to include would be a calculated average access time, given user provided access latencies for each of the caches – similar to question 3. The statistic would provide a high-level performance metric of the cache prefetcher used. Another useful statistic would be to count the number of times that a block was prefetched and not used. This statistic would measure how much the prefetcher is polluting the cache with useless entries, which gauges the accuracy of the prefetcher. Furthermore, this statistic could be extended by also measuring how many times a prefetched block was used. This measure would indicate the amount of improvement of overall performance attributed to the prefetching algorithm.

Open-ended Prefetcher : A Global History Buffer – based design

For the open-ended prefetcher, we used a global history buffer (GHB) inspired approach. Our structure for the prefetcher consists of a global history FIFO queue, implemented as a linked list, and a separate two-element queue containing the most recently observed cache accesses. This prefetcher works on deltas between subsequent memory access addresses in order to account for sequential and incrementing access patterns. On every call to the prefetch algorithm, both the two-element queue and the global FIFO queue are updated with a delta calculated from the current access address. The global FIFO queue is then searched in reverse order for the first pattern that matches the two-element queue. If discovered, the linked list element before the matched two-element queue is taken as the delta to add to the current access address to yield the prefetch address. Essentially, this algorithm works on the principle of pattern matching, partially similar in idea to the GaP branch prediction pattern matcher. This prefetching algorithm is capable of accurately predicting memory access patterns that have alternating deltas, a case that would cause stride to fail due to the irregular location accesses. This algorithm is also scalable because both the global history FIFO queue could be increased or the two-element queue could be made deeper. We have chosen these sizes to work well with the cache specified in cache-lru-openend.cfg. A CACTI simulation was used to analyze the physical overhead introduced

by this prefetcher.

Cache:

Data

Height(mm) : 0.258213
Width(mm) : 0.186667
Area(mm2) : 0.0481999
Total dynamic read energy (nJ) : 0.0181207
Time (ns) : 0.28507

Tag

Height(mm): 0.0369555
Width(mm) : 0.0211796
Area(mm2) : 0.000782704
Total dynamic read energy (nJ) : 0.000409321
Tag (ns) : 0.155352

Prefetcher

Access time(ns) : 0.118515
Total dynamic read energy (nJ) : 0.000312963
Total leakage power of bank (mW) : 0.0286315
Height by width (mm) : 0.0196443 x 0.0130175

Evident in the above results, the prefetcher adds slight overhead. The amount of extra power lost and additional area required is on the order of another tag structure using the specified cache. However, the prefetcher does introduce extra delay also on the order of a tag access, which is comparable to the data access time of the cache. If the prefetcher is activated on every cache access, the total access time would be increased about (0.118515 / (0.28507 + 0.155352)) = 27%. Given that the prefetcher decreases the average miss rate between the three microbenchmarks to below 2.1% from a baseline average miss rate of about 2.3%, it may be more beneficial to include the prefetcher if time penalties are substantial for a miss rate. Hence, the decision to include the prefetcher depends heavily on the data transmission time between the L2 cache, the access time of the L2 cache, and the time penalty resulting from L1 and L2 misses.

Statement of Work Completed by Each Partner

During this lab, Yi Fan created the three microbenchmark test programs used to debug and validate the effectiveness of the three prefetchers: nextline, stride, and the open-ended design. Yi Fan also provided suggestions for the open-ended design as well as produce some of the results and figures shown in this report. Jianwei created the simulation implementation of the stride prefetcher and the open-ended design, along with providing open-ended design suggestions and debugging. Jianwei also wrote a portion of this report.