Lab04

Created by	J Jiping Li
Created time	@November 24, 2023 12:14 AM

Question 1

Micro-benchmark: By accessing every 64th byte, this approach aligns with the 64-byte length of a cache line. The prefetcher, designed to automatically load the next cache line, ensures a high likelihood of cache hits when elements in the newly loaded line are accessed. If the access sequence is smaller or equal to the block size then miss rate will be 0.

With micro-benchmark access jump specified to 64, miss rate is 0%

With micro-benchmark access jump specified to 512, miss rate is 16.67%

Question 2

Micro-benchmark: In this scenario, we accessed every 512th element. Considering that the cache line spans 64 bytes, this approach involves accessing data from every eighth line. For this particular microbenchmark, the next-line prefetcher was not effective. While the stride prefetcher demonstrated a low rate of cache misses in this scenario with it's RPT mechanism. This provides use that if the access sequence are constant, then the miss rate will be 0.

With micro-benchmark access jump specified to 64, miss rate is 0%

With micro-benchmark access jump specified to 512, miss rate is 0%

Question 3

Given that T_L1 = 1, T_L2 = 10, T_Memory = 100

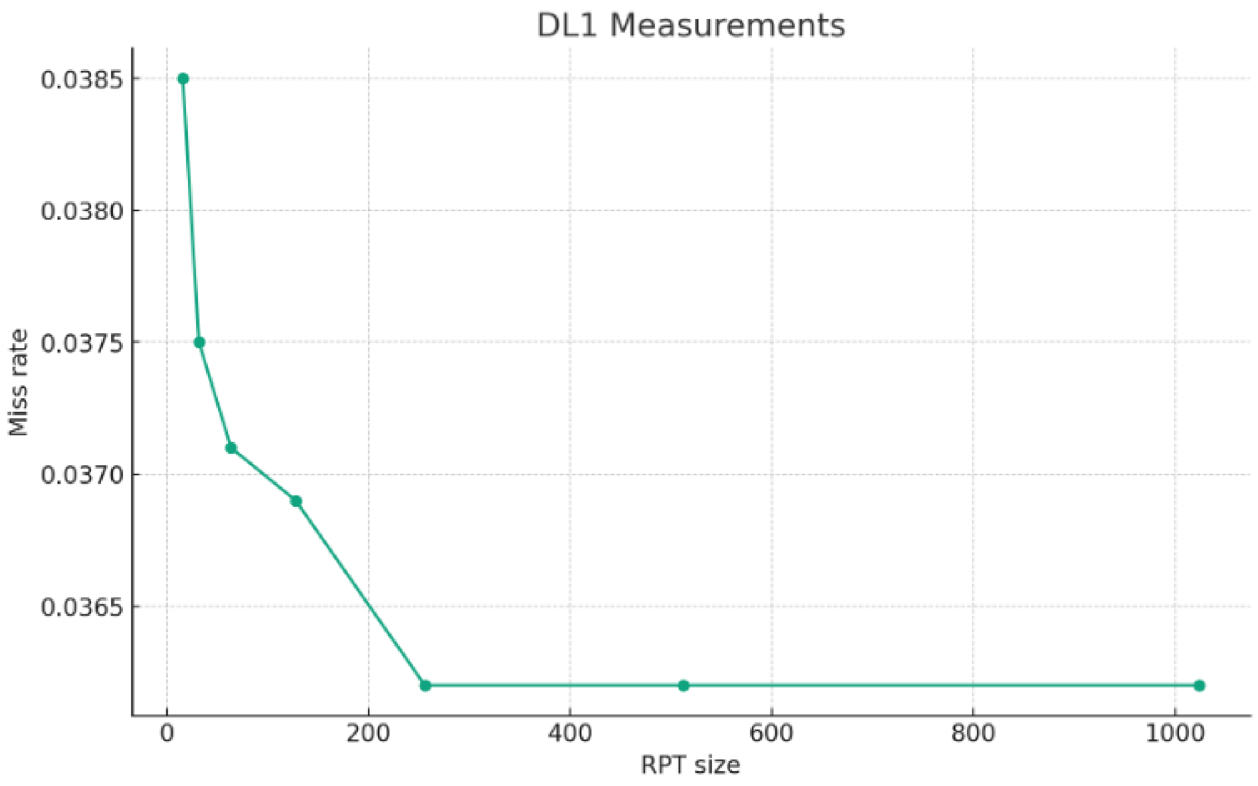
T_avg = T_L1 + L1 Miss Rate * (T_L2 + L2 Miss Rate * T_Memory)

T_avg = 1 + L1 Miss Rate * (10 + L2 Miss Rate * 100)

Config	L1 Miss Rate	L2 Miss Rate	Average Access Time
Baseline	0.0416	0.1140	1.89
Next-line Prefetcher	0.0419	0.0838	1.77
Stride Prefetcher	0.0385	0.0578	1.60

Question 4

To measure performance, we pick compress.eio as a benchmark, and we consider l1 miss rate only. **:** The most significant decrease in miss rate occurs between the RPT sizes of 16 and 64.After reaching an RPT size of 256, the miss rate stabilizes around 0.0362. This plateau suggests that beyond a certain point (in this case, an RPT size of 256), increasing the RPT size does not significantly reduce the miss rate.



Question 5

If the access time for each cache level is available, incorporating these times can offer a broad overview of the overall cache performance. This approach will reveal the extra time incurred due to cache misses, thereby highlighting the impact on overall system efficiency. Essentially, it provides a measure of performance enhancement, emphasizing the time saved or lost in cache operations.

Question 6

In our advanced open-ended implementation, we drew inspiration from the branch predictor utilized in our previous lab2. This involved incorporating a counter with a threshold into the steady state mechanism. The operation of the steady counter is as follows:

Once the program enters the STEADY state, it counts occurrences of identical strides in subsequent iterations. The SteadyCounter increments with each occurrence of the same stride. Conversely, if a different stride is detected, the SteadyCounter decreases until the program exits the Steady State.

Additionally, we adopted another strategy by enlarging the size of the RPT entries. This modification was strategically implemented to achieve a reduction in the miss rate for the “compress” benchmark, successfully decreasing it from 3.84% to 3.62%. This approach highlights our commitment to optimizing performance through thoughtful, data-driven modifications in prefetching mechanisms.

Micro-benchmark

In this scenario, we accessed every 512th element. Considering that the cache line spans 64 bytes, this approach involves accessing data from every eighth line. how ever as we add a change in the access sequence in this case i%32, there will be a stage change back to INIT, which cause the stride in RPT change, thus will lead to a high miss prediction in in the stride-prefetcher. our design of open-ended implementation will solve this