ECE552 Lab4 report
Francis Guo 1005931397
Rongbo Zhang 1005273393

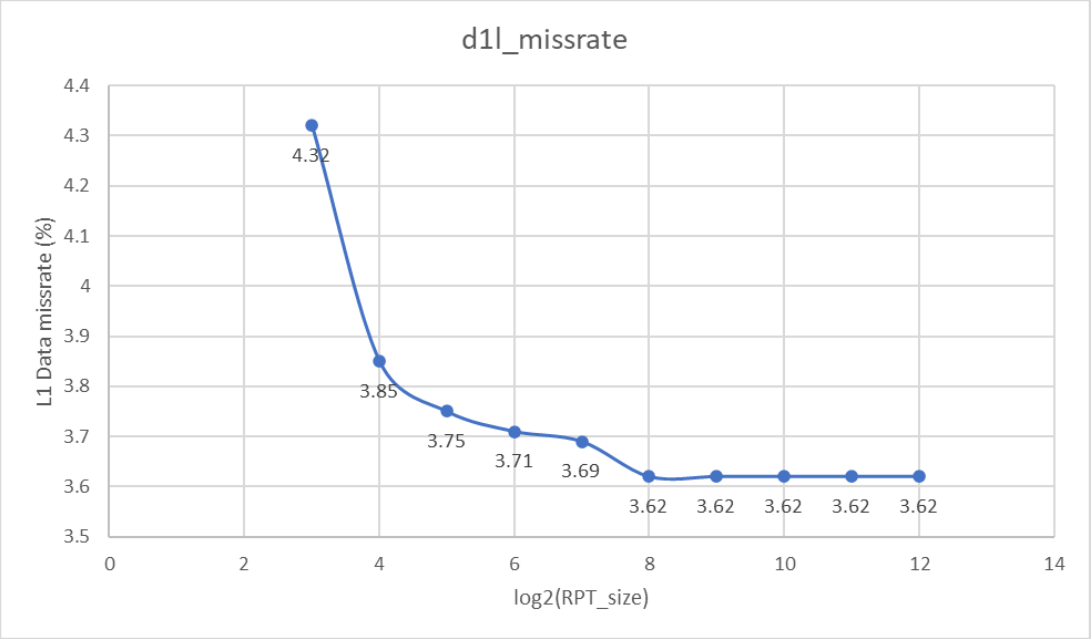
Q1.
For the micro benchmark for the next-line prefetcher. For the choice of the microbenchmark, we had an array of 1,000,000 elements. The elements are int type. Then when we are looping through this array, when we set the step of looping equal to 1, which means that we are stepping 4 bytes everytime we loop, we are getting a very small miss rate. In the tests we got 0% of the missing rate. (0% is rounded number, there are some cache misses) And when we are stepping 128 elements which are 4*128 = 512 bytes. So we are getting a higher cache miss rate which is 67.51%. This shows that the next-line prefetcher is working correctly.

Q2
For the micro benchmark for the stride prefetcher. We had an array of 1,000,000 elements of int. Then we are looping through this array. In this micro benchmark, we used a switching step, which means when we are looping through the array, if the first time we used a step of 64 elements which are 256 byte, then, the next time we will use step of 128 which is 512 byte. In this case, we are getting a high miss rate which is 11.98%. Then, we change it to a constant step of 64 element which is 128*4 = 512 byte. In this case we have a miss rate of 0.5%. With these results, we could confirm that the stride prefetcher is working properly.

Q3
$$T_{avg} = T_{L1-hit} + MR_{L1} * (T_{L2-hit} + MR_{L2} * T_{Memory})$$
$$T_{avg} = 1 + MR_{L1} * (10 + MR_{L2} * 100)$$

Config	L1 Miss Rate	L2 Miss Rate	Average access time
baseline	4.16%	11.40%	1.89
next-line	4.19%	8.38%	1.77
stride	3.85%	5.78%	1.61

Q4



Q5

I am considering adding the accuracy and coverage of the prefetcher, and the average access time as a result of the prefetcher. By looking at the accuracy we can study if the prefetcher pollutes the cache with unnecessary data by evicting useful data. By looking at the coverage we can study how many misses did it save. By looking at the average access time we can study the performance gain of adding the prefetcher.

Q6

For the micro benchmark for the open-ended prefetcher. We had an array of 1,000,000 elements of int. Then we are looping through this array. In this micro benchmark, we used a patterned switching step, which means when we are looping through the array, we will have a pattern of 10 iter of the array. The steps are shown in a pattern of {10, 10, 64, 64…}. In this case, we could see that our open prefetcher has 0% of miss rate and the stride prefetcher causes a 1.25% of miss rate.

For the open-ended prefetcher, the open-ended prefetcher we implemented a modified stride prefetcher. Here, we have implemented a stride prefetcher that we change the size of the rpt table to 1K. We have also changed the output of the state for the stride predictor. We change to prefetcher when we are at the steady state of the steady. This means that we are only allowed when the prefetcher is very confident about the prediction for the address of prediction.

For this open predictor, we have the RPT table size of 1024, we are having the RPT table size is 16 Byte * 1024 = 16,384 Byte. This is 16K bytes. This is larger than the accrual L1 data cache. So, this open prefetcher is not feasible to be applied to the processor. From the CACTIA result, we have the following for the RPT table:

```
Access time (ns): 0.492951
Cycle time (ns):  0.443181
Total dynamic read energy per access (nJ): 0.00433307
Total leakage power of a bank (mW): 5.33019
Cache height x width (mm): 0.134653 x 0.206732
```

The size is too large and the leak power is high on the RPT table as shown in the graph.