Lab Report

Harshita Huria: 1000980398
Angel Serah: 1000491773

Q1) 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.

For our microbenchmark for next-line, we had an int array of 1000 elements. We looped over these 1000 elements 10000 times to simulate a higher number of accesses. The cache line is 64 bytes long. When we access every 16th element i.e. every 64th byte, next line will have a very low miss rate (0.03%), as the next 64 bytes are prefetched with every access. However, if we access every 32nd element, next line prefetcher will fail. We get a miss rate of 16.61% in this case, which is much worse. We used the same configuration file as was supplied for the other benchmarks i.e. with 16KB 4-way associative data L1 cache.

Q2) 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.

For our microbenchmark for stride prefetcher, we had an int array of 1000 elements. We looped over these 1000 elements 10000 times to simulate a higher number of accesses. We tested the case which was failing for next line i.e. accessed every 128th byte. Stride prefetcher had a miss rate of 0.0%. We also tested with different strides i.e. 64, 128 and still got a very low miss rate. However, when we created a benchmark which accessed a linked list which was dynamically allocated, the miss rate was much worse (5.8%), proving that when the stride was unpredictable, the prefetcher performed worse. We used the same configuration file as was supplied for the other benchmarks i.e. with 16KB 4-way associative data L1 cache.

Q3) 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: Taccess−L1Data = 1, Taccess−L2 = 10, Thit−Memory = 100

Config	**L1 Miss Rate**	**L2 Miss Rate**	**Average 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

Average Access Time Calculation:

$T_{L2\text{-}Miss} = T_{Memory} = 100$

$T_{L1\text{-}Miss} = T_{L2\text{-}Hit} + T_{L2\text{-}Miss} * MR_{L2}$

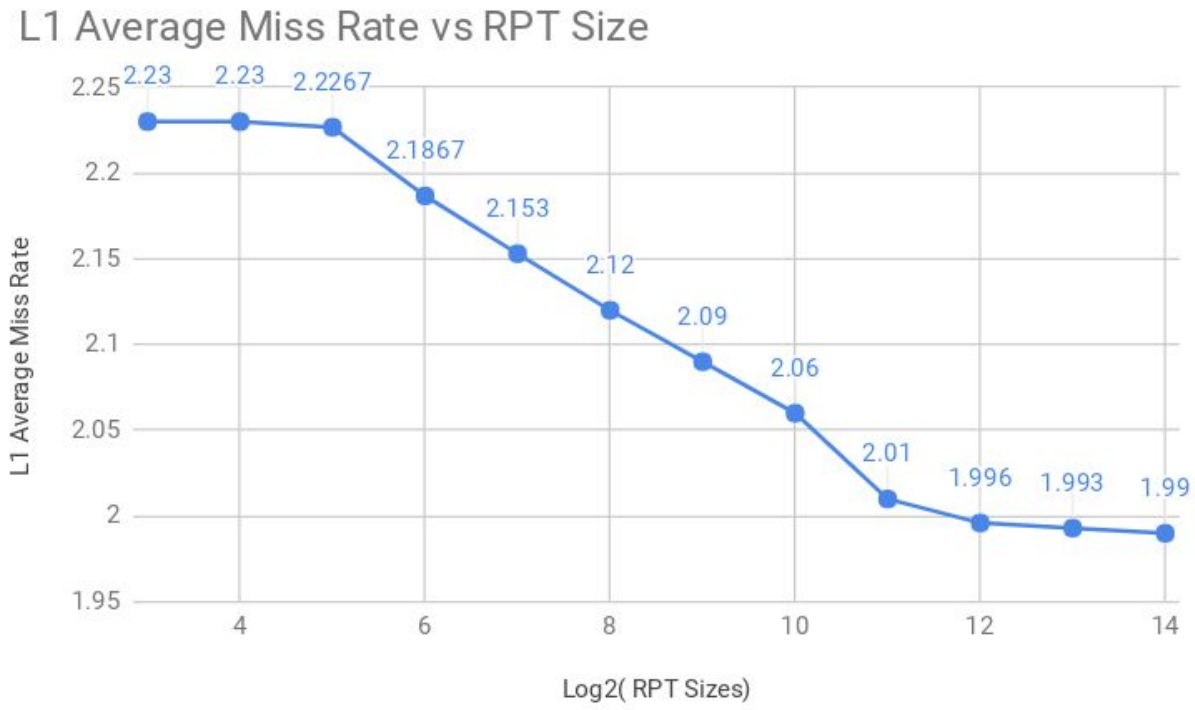
$= 10 + 100*MR_{L2}$

$T_{L1} = T_{L1\text{-}Hit} + T_{L1\text{-}Miss}*MR_{L1}$

$= 1 + (10 + 100*MR_{L2})*MR_{L1}$

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.

The metric that we selected for the y axis was the average L1 miss rate for the three benchmarks that were given i.e. gcc.eio, go.eio and compress.eio



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.)

Being able to calculate average access time (i.e. latencies) would be a useful parameter to study and analyse cache 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.

To confirm that the open-ended prefetcher worked, we used the microbenchmark which performed badly on the stride prefetcher. This was because we accessed a dynamically allocated linked list with no predictable stride. However, with the open ended prefetcher, the miss rate was reduced down to 0.03%, which was a large difference from the stride prefetcher's 5.8%. This improvement was because we kept a stream buffer of accessed addresses and used that to prefetch addresses when stride failed.

• *Describe your open-ended data prefetcher implementation. Reason about how realistic your data prefetcher is in terms of area overhead and access time.*

We decided to combine the Stride Prefetcher with the Stream Buffer Prefetcher technique. It stores the missed addresses in a separate cyclical buffer, which is different from the cache. With each new address access, we try to prefetch using the stride prefetcher and if there is no steady stride which we can use, we try to find the current address in the stream buffer. If the address is in the stream buffer, we prefetch the address in the buffer immediately after the given address. To meet the lab constraints, we used an RPT table of size 1024. This increased our area by 16B*1024 = 16,384B i.e. 16KB. Moreover, we used a stream buffer of type m_addr_t of size 4096 which had a size of 4B*4096 = 16,384B i.e. 16 KB. So, we had to use an extra 32KB to implement this prefetcher. Realistically, this would not be a feasible technique as we could have used that area to increase the size of the L1 cache instead which would have an effect in decreasing miss rates.

• Include a brief statement of work completed by each partner.
Harshita Huria : 50% of workload
Angel Serah : 50% of workload