Shawn Zhai 1006979389
Victor Wu 1007069039

Question 1:

The microbenchmark verifies the next-line prefetcher by sequentially accessing memory in increments equal to the cache block size (BLOCK_SIZE). The benchmark demonstrates near 100% prefetch hit rate, as all accesses after the initial miss are prefetched correctly. Deviating beyond BLOCK_SIZE would result in cache misses, confirming the prefetcher's expected behavior. Ran using config **cache-lru-nextline.cfg.**

Question 2:

The microbenchmark verifies the stride prefetcher by accessing an array with a fixed stride of 128 bytes across iterations. This pattern exploits regular address strides, which the stride prefetcher can detect and prefetch efficiently. The benchmark highlights the stride prefetcher working as intended as it had significantly fewer misses (only 0.001) compared to the next-line prefetcher. Ran using **cache-lru-stride.cfg**.

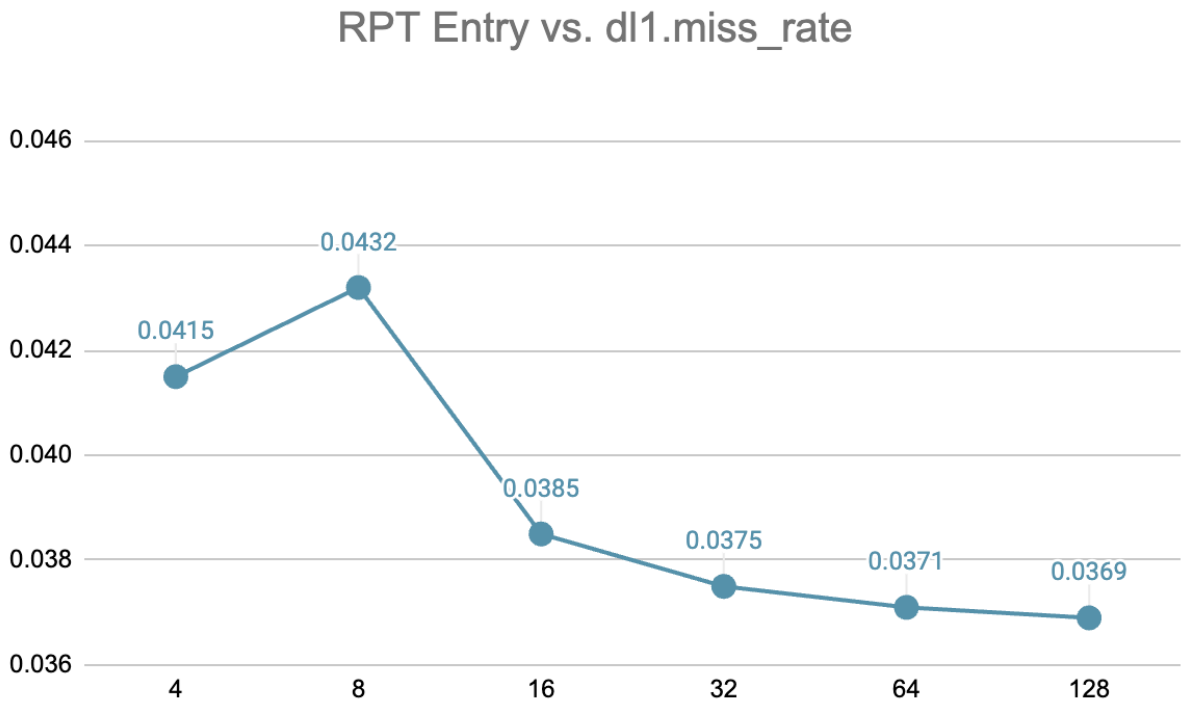
Question 3:

Config	**L1 Miss Rate**	**L2 Miss Rate**	**Average Access Time**
cache-lru.cfg (base)	0.0416	0.1140	1.89024
cache-lru-nextline.cfg	0.0419	0.0838	1.770122
cache-lru-stride.cfg (16 RPT entries default)	0.0385	0.0578	1.60753

average access time = T_L1 + %miss_L1*(T_L2 + %miss_L2 *(T_memory))

Question 4:

RPT entry	**L1 data miss rate**
4	0.0415
8	0.0432
16 (default)	0.0385
32	0.0375
64	0.0371
128	0.0369



The graph illustrates the effect of varying RPT sizes on the L1 data cache miss rate for the compress benchmark. Smaller RPT sizes (4 and 8 entries) result in higher miss rates due to insufficient tracking of memory streams, while the most significant improvements occur up to **16 entries (0.0385 miss rate)**. Beyond 16 entries, the improvement diminishes, indicating that the working set of this benchmark does not require tracking more than 64 streams for optimal prefetching.

Question 5:
Prefetch-Induced Misses: Cache misses caused by evicting useful blocks to accommodate prefetches, identifying potential drawbacks.
Latency Reduction: Average memory access latency improvement attributable to prefetching, showing practical performance benefits.
CPU Stall Time Reduction: Quantifies the reduction in CPU stall cycles caused by memory access delays due to prefetching. This gives a practical view of how prefetching improves CPU throughput.

Question 6:
Our open-end prefetcher, with enhanced multiple-prefetching and a more capable 32-entry RPT. The microbenchmark accesses <addr + stride> and <addr + 2*stride> in the same loop cycle, which is the scenario our open end prefetcher should perform well at with multiple prefetches. Overall it achieves a very low miss rate on the microbenchmark as expected. We do not have a microbenchmark to test the confidence mechanism, but we found this practice yields lower misrate. Ran using config **cache-lru-open.cfg**.

Open-ended Prefetcher Implementation

Our open-ended prefetcher is an extension to the stride prefetcher. Two mechanisms, prefetch depth and stride confidence, are introduced to improve the performance. The prefetcher has a prefetch depth of 2, which means that both memory <addr + stride> and <addr + 2 * stride> are considered when prefetch happens. This prefetch depth mechanism further exploits spatial locality of memory access. Moreover, each RPT entry now has a stride confidence value between 0 and 2. it increases when the new stride is the same as the old stride and is reset to 0 otherwise. Unlike the stride prefetcher, where prefetching happens when the state is not no-pred, our open-ended prefetcher only prefetches when the state is stable or the confidence value is 2. This confidence mechanism reduces the risk of prefetching useless data into the cache.

	compress.eio	gcc.eio	go.eio	average
L1 data miss rate	0.0368	0.0122	0.0111	0.02

Open-ended Prefetcher Feasibility

UG machine is 64-bit
RPT entry: previous address 64 bits + stride 32 bits + state 2 bits + confidence 2 bits + tag 56 bits (64 - 3 least significant bits - 5 index bits) = 156 bits
32 RPT entries, total size = 156 x 32 = 4992bits = 624 bytes

According to CACTI, our open-ended prefetcher has an area of 0.00243066mm2 and an access time of 0.345969ns. A typical 32 KB L1 data cache occupies around 0.2–0.5 mm². Our prefetcher occupies less than 0.7% of the area of a small L1 cache, making the area overhead negligible. L1 cache access times are typically 0.5–1 ns, which means our prefetcher operates significantly faster than the cache. These characteristics confirm that the prefetcher is both area-efficient and high-performance, making it highly feasible.

Work completed by each partner

Shawn: implementation of next-line prefetcher, implementation of stride prefetcher, implementation and parameter-tuning of open-ended prefetcher, report writing.
Victor: Microbenchmarks, configuration files, implementation and parameter-tuning of open-ended prefetcher, data collection, report writing.

Reference

Jiajun Wang, Reena Panda, and Lizy K. John. 2018. SelSMaP: A Selective Stride Masking Prefetching Scheme. ACM Trans. Archit. Code Optim. 15, 4, Article 42 (December 2018), 21 pages. https://doi.org/10.1145/3274650

Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V. Gratz, and Daniel A. Jiménez. 2019. Perceptron-based prefetch filtering. In Proceedings of the 46th International Symposium on Computer Architecture (ISCA '19). Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3307650.3322207