ECE552 PRA0102 - Station 8
Ting-Hao (Tim) Cheng (997289090)
Isaiah Tan (999004937)

Q1

The microbenchmark tests an array accessed by a specified strike, inputted by the user. If the stride is large enough, that the miss rate is significantly increased. If the stride is smaller or equal to the block size, the miss rate will be near 0%. This is because since only the next block is prefetch, a sequential array with small strides will always have the next block prefetched, and reads will not encounter cold misses (except the first access to the array, and other stack variables).

mbq1 output
When the stride is specified to skip 10B, miss rate is 0%.
When the stride is 128B, miss rate is 9.96%.

Q2

The microbenchmark tests an array accessed by a specified strike, inputted by the user. No matter the size of the stride, the miss rate should be near 0%. This is because the stride prefetcher uses PC to index the RPT and only one stride is used. Since the stride is constant, the next address can be prefetched, despite any stride size. Only the first access to the array will lead to a cold miss, or other stack variables.

mbq2 output
When the stride is specified to skip 10B, miss rate is 0%.
When the stride is 128B, miss rate is 0%.

Q3

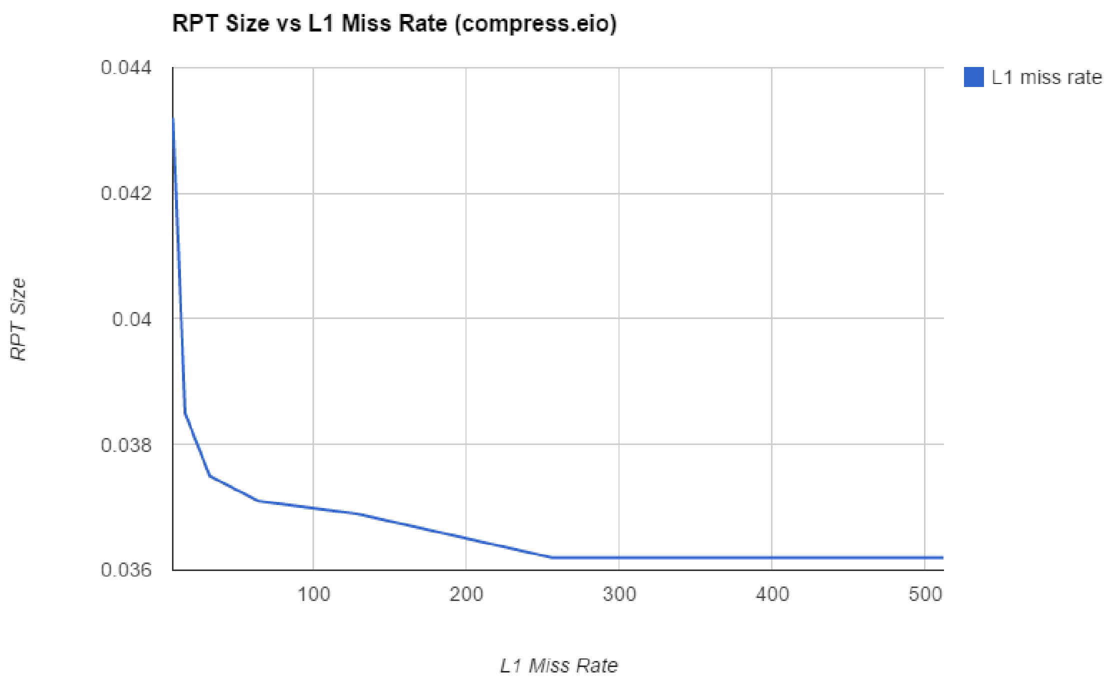
Config	L1 Miss Rate	L2 Miss Rate	Avg. Access Time
No Prefetcher lru	0.0416	0.1140	1.89
Next Line Prefetcher	0.0419	0.0838	1.77
Stride Prefetcher	0.0385	0.0578	1.61

$T_{avg} = T_{access-L1} + (L1\ Miss\ Rate)\, T_{L1-miss}$
$T_{L1-miss} = T_{access-L2} + (L2\ Miss\ Rate)\, T_{hit-memory}$
-> $T_{avg} = T_{access-L1} + (L1\ Miss\ Rate)\,(T_{access-L2} + (L2\ Miss\ Rate)\, T_{hit-memory})$

Q4

Shown on the table and chart below are the increasing RPT size to L1 miss rates. After RPT of size 256, the rates do not improve. A possibility is that the benchmark in use does not require many RPT entries (not many PC addresses require data retrieval), and hence the table cannot be fully filled.

RPT Size	8	16	32	64	128	256	512
L1 miss rate	0.0432	0.0385	0.0375	0.0371	0.0369	0.0362	0.0362



Q5

It would be useful to have a measurement for the number of useless fetches (number of fetched blocks that are never accessed before its eviction). This would allow us to get the average number of useless prefetches over the total prefetches.

ECE552 PRA0102 - Station 8
Ting-Hao (Tim) Cheng (997289090)
Isaiah Tan (999004937)

Q6

Open-ended description
For the open-ended implementation, a stride coupled with a history buffer was implemented. When the RPT state is not stable for a given PC and address access, the buffer was referred to instead. This buffer stores a table of memory accesses with the address as index, referring to the possible 'next' memory address ('next' address used to generate a prefetch address). The 'next' address is populated by filling the 'next address' column of the prior access address. The buffer is evicted by LRU, keeping the most recent history if the buffer is full and needs to add new history.

Example usage
If address 0x00c0 and 0x0100 was previously accessed sequentially, when 0c00c0 is encountered again, 0x100 will be prefetched.

If 0x01c0 was previously access, and memory is current accessing 0x0200, fill 0x0200 as the 'next address' of 0x01c0

History Buffer

Previous address	Next Address
0x00c0	0x0100
0x01c0	0x0200

Open-ended size (assuming 64-bit architecture)
RPT: 256 entries. (2^8)
256 entries * [(64 - 8 - 3) tag bits + 64 previous address bits + 64 stride bits + 2 state bits] = 46848 bits = 5856B

History buffer:
256 entries * (64 'previous' address bits + 64 'next' address bits + 10 bit saturating counter for LRU) = 35328 bits = 4416B

As the total number of bits required for the open-ended implementation is smaller than the L1 cache (16kB), the additional area due to this implementation is feasible.

Open-ended access time

Access time to the history buffer may not be feasible, as the buffer must be fully associative to be timely (to retrieve the ‘next address’ in a timely manner). This is extremely expensive. To evict a spot in the buffer, the LRU saturated counter with the largest value must be found, adding to the need for a fully associative table. As associativity of 1024 is not scalable and immensely power consuming, this implementation may not be feasible.

Microbenchmark

The microbenchmark tests that the history buffer is functional. A sudo-random number is generated to index an array in a tight loop, and a second loop is added to restart the sudo-number generator and iterate the array multiple times. Because of the suo-random numbers, the array is not accessed with constant stride, and the stride generator will not be useful. After one iteration of the array, the history buffer can accurately predict the second iteration onwards. The miss rate of the microbenchmark is near 0% because of the buffer, despite the lack of prefetching from the stride prefetcher.

Work Delegation

Isaiah:
nextline prefetcher
open-ended prefetcher
microbenchmarks
lab report

Tim:
stride prefetcher
open-ended prefetcher