Ringo Cheung - 1000463955
Yassir Solomah - 1000458284

Lab 4 - Data Caches

1 Results

Config	L1 Miss Rate (%)	L2 Miss Rate	Average access time
baseline	4.16	11.40	1.89
next-line	4.19	2.92	1.54
stride	3.85	5.78	1.61

Table 1: Average access times for three different prefetcher configurations

Average access time was computed as follows:

$$T_{cache_access} = T_{access-L1Data} + \%_{miss-L1} * T_{access-L2}$$
$$T_{access-L2} = T_{access-L2Data} + \%_{miss-L2} * T_{hit-Memory}$$

2 Next-line Prefetcher

2.1 Micro-benchmark

```
#define BLOCK_SIZE 16 // 64B blocks
#define ARRAY_SIZE 8192
#define ITERATION_SIZE 409600

int main() {
    ...
    // Miss only on loop around 8192/16 = 512, 409600/512 = 800 misses
    for(i = 0; i < ITERATION_SIZE; i++) {
        dummy += array[(i*16) % ARRAY_SIZE];
    }

    // Miss always, two blocks over, 409600/100 = 4096 misses
    for(i = 0; i < ITERATION_SIZE / 100; i++) {
        dummy += array[(i*64) % ARRAY_SIZE];
    }

    // Estimated Direct Mapped = 4896
    // Direct Mapped - dl1.misses                    4939 # total number of misses
    ...
}
```

This benchmark was compiled using `ssbig-na-sstrix-gcc -O1 mbq1.c -o mbq1` and then run using `./sim-cache -config cache-config/cache-lru-nextline.cfg mbq1`.

As expected from the benchmark, next-line predictor successfully captures the next block, but fails on any other scenario.

3 Stride Prefetcher

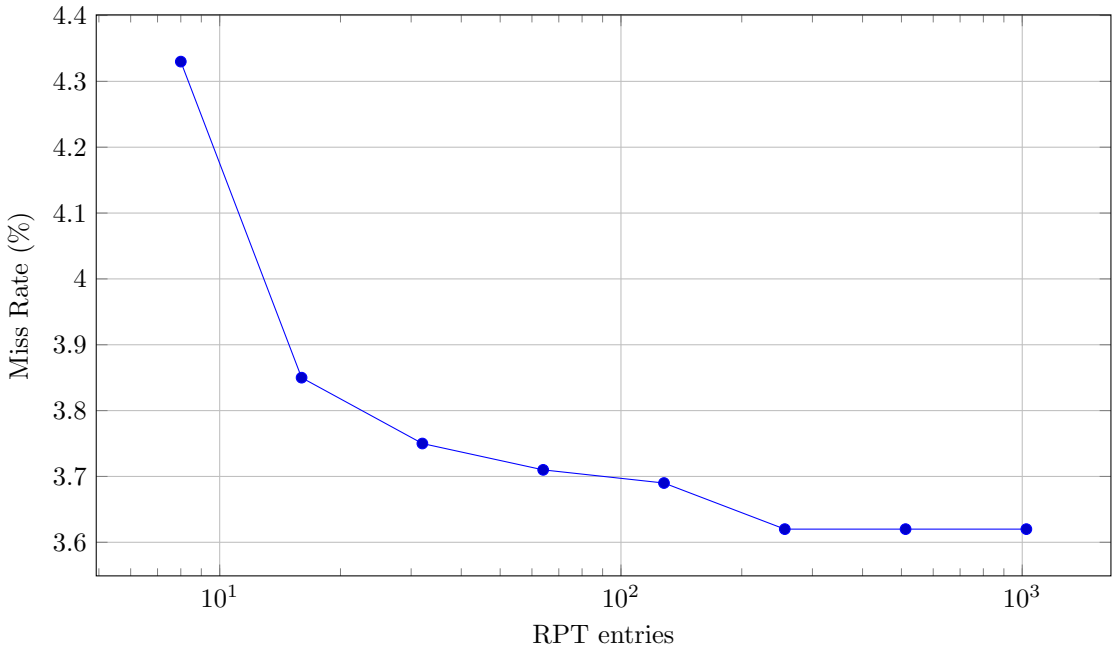
```
#define BLOCK_SIZE 16 // 64B blocks
#define ARRAY_SIZE 8192
#define ITERATION_SIZE 409600

int main() {
    ...
```

```
    // Next Block - Fails on wrap around 8192/16 = 512; 409600/512 = 800 misses
    for(i = 0; i < ITERATION_SIZE; i++) {
        dummy += array[(i*16) % ARRAY_SIZE];
    }

    // Two Blocks Away - Fails on wrap around 8192/32 = 256; 409600/256 = 1600 misses
    for(i = 0; i < ITERATION_SIZE; i++) {
        dummy += array_two[(i*32) % ARRAY_SIZE];
    }

    // Different Blocks, Aliasing Index - 409600*2/100 = 8192 misses
    for(i = 0; i < ITERATION_SIZE / 100; i++) {
        dummy += array_three[(i*32) % ARRAY_SIZE];
        __asm__("nop");
        ... // nop repeated to alias PC bits [7-4]
        __asm__("nop");
        dummy += array_four[(i*63) % ARRAY_SIZE];
    }


    // Estimated Direct Mapped - 800+1600+8192 = 10592
    // Direct Mapped - dl1.misses                     10636 # total number of misses
    ...
}
```

This benchmark was compiled using `ssbig-na-sstrix-gcc -O0 mbq2.c -o mbq2` and then run using `./sim-cache -config cache-config/cache-lru-stride.cfg mbq2`.

As expected, stride-predictor correctly prefetches when theres a constant different (regardless of block distance). This implies that stride predictor only will achieve optimal conditions up to a constant derivative of the block distance over time. We also demonstrate the effect of *destructive* aliasing, which results in corruption of the RPT entries.

3.1 Performance with varying entries in the RPT



The metric chosen to represent prefetcher performance is miss rate. The graph shows that as we

increase the number of entries in the RPT table, we do see a decrease/improvement in the miss rate. However, it appears to cap off at 3.62% for the compress benchmark. This could be because there are not more than 256 loads with unique PC's so the aliasing of the RPT is not an issue. One conclusion to be drawn from this graph is that it is insufficient to simply increase the size of the RPT to get a better miss rate as the miss rate does not improve consistently with more RPT entries. The feasibility of the stride prefetcher decreases as the RPT size increases because more area is needed for the prefetcher which takes up valuable hardware space and increases the latency of a prefetch.

4 Open-ended Prefetcher

The open-ended prefetcher is a combination of miss-queue prefetching and stride prefetching. On every prefetch, the miss queue is searched for a valid entry before the stride prefetcher is used. The miss queue is a data structure that is updated every time a cache miss occurs. The miss queue entries are the memory addresses being read or written to when a cache miss occurs. When the miss queue is searched, the memory address after the one that is currently being prefetched for is returned. Intuitively, the miss queue returns the next address accessed the last time the current address was a miss.

We were able to achieve an average dl1 cache miss rate of 2.05% using 4 way set associativity in the cache, a miss queue size of 512 and an RPT with 256 entries.

4.1 Benchmark

```
#define BLOCK_SIZE 16 // 64B blocks
#define ARRAY_SIZE 16384
#define ITERATION_SIZE 40960

int main() {
    ...
    int * array = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    int * array_two = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    int * array_three = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    int * array_four = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    int * array_five = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    int * array_six = (int *) malloc(sizeof(int) * ARRAY_SIZE);
    ...

    // Stride always fails, open-ended will always prefetch correctly
    for(i = 0; i < ITERATION_SIZE; i++) {
        j = (int) pow(2, i % 14);
        dummy += array[j % ARRAY_SIZE];
        dummy += array_two[j % ARRAY_SIZE];
        dummy += array_three[j % ARRAY_SIZE];
        dummy += array_four[j % ARRAY_SIZE];
        dummy += array_five[j % ARRAY_SIZE];
        dummy += array_six[j % ARRAY_SIZE];
    }

    // Direct Mapped Estimate STRIDE - 40960*6 = 245760
    // Direct Mapped Estimate OPEN - Bias + 6*15

    // Direct Mapped STRIDE - dl1.misses                  245818 # total number of misses
    // Direct Mapped OPEN - dl1.misses                     573 # total number of misses
    ...
}
```

This benchmark was compiled using `ssbig-na-sstrix-gcc -O0 mbq6.c -o mbq6` and then run using `./sim-cache -config cache-config/cache-lru-open.cfg mbq6`.

As shown in this scenario, we vary the step size by an exponential variation, resuting in a non-constant block-distance derivative over time. This implies that stride will not be able to correctly to prefetch. As also expected, our open-ended predictor always prefetches correctly (up to a degree with some bias).

4.2 Feasibility

In order to evaluate the feasibility of the open-ended predictor, CACTI was used to determine the access time and area of the three data structures involved in the dl1 cache with the open-ended predictor: the dl1 cache, the RPT and the miss queue. The access time and area for the three structures are given below.

Structure	Access time (ns)	Area (μm^2)
DL1 cache	0.049	27 945
RPT	0.235	9682
Miss queue	0.393	8400

Table 2: Access times and area of the open-ended prefetcher implementation analyzed with CACTI

The DL1 cache was modelled in CACTI as a 16KB 4-way set associative cache with 64 byte lines, the RPT was modelled as a $256 * (64 * 3)/8 = 6144$ byte fully associative cache with $(64 * 3)/8 = 24$ byte blocks and the miss queue was modelled as a $512 * 64/8 = 4096$ byte fully associative cache with $64/8 = 8$ byte blocks. Using this analysis, the open-ended prefetcher implementation is feasible because the access time of the RPT and miss queue are both lower than the DL1 cache meaning the the timeliness of the prefetcher is not a concern since the RPT and miss queue can be accessed in parallel by the prefetcher with a decision maker deciding which of the two to use. The areas of the prefetcher is also only 64% of the area of the DL1 cache.

5 Additional statistics

Some additional statistics that would have been useful to study the performance of the prefetcher are the access time of the prefetcher and the area of the prefetcher. The access time is important because the prefetcher must be fast enough to prefetch the value from memory before the value is needed by the program but also not too fast that it will pollute the catch by evicting still useful values. The area is important because it tells us how feasible the prefetchers to implement in hardware.

6 Statement of Work

Equal work was completed by both partners.