Lab 8 & 9 Cache Lab Report

20220665 유영준

Lab8에서는 Cache Organization에 대해 배웠다. 메모리 주소는 Set Index, Block offset, Tag로 구성되고, Cache 메모리에서는 locality를 이용하여 data 또는 instruction을 저장한다. Cache에 해당 data가 존재하느냐에 따라 hit, miss, eviction이 존재하고, replacement policy로는 LRU, FIFO, LIFO 등이 존재한다.

Lab9에서는 hit ratio를 증가시킬 수 있는 방법인 Blocking에 대해 배웠다. matrix multiplication에서 matrix를 block으로 구분하여 miss가 발생하는 빈도를 줄일 수 있었다.

Lab8과 Lab9와 관련하여 각각 Part A: Building a cache simulator, Part B: Efficient Matrix Transpose에 대한 과제가 나왔고, 풀이 과정은 다음과 같다.

1. Part A: Building a cache simulator

- Cache, Set, Line을 각각 구현하기 위해 struct를 활용했다. Cache는 Set으로 구성되고, 각각의 Set은 Line들로 구성되고, 각각의 Line은 valid, tag, lru로 구성된다. lru는 해당 과제의 Replacement Policy인 LRU 방식을 구현하기 위해 도입한 변수이다.
- 전역 변수로 cache를 생성하고, hit_count, miss_count, eviction_count를 0으로 초기화해주었다.
- main 함수에서는 getopt 함수를 이용하여 인자를 parsing하여 s, E, b에 적절한 값을 넣고, 변수 t에 파일 이름을 저장하였다.
- 입력 받은 s, E, b 값을 바탕으로 cache 공간을 동적할당 해주었고, 각 line들의 valid, tag, lru 값은 0으로 초기화해주었다.
- t에 저장한 파일을 한 줄씩 읽으면서 cache access를 수행한다. operation이 'I'일 경우 건너뛰고, 'M'일 경우 두 번씩 cache access simulation을 수행한다.
- cache access simulation의 구현 방법은 다음과 같다. 먼저 입력 받은 address를 tag부분과 set_idx 부분을 분리하였다. set_idx는 tag부분을 지운 뒤 b만큼 right shift 해주어 분리하였다.
- 해당 set_idx에서 line을 돌며 tag가 같은 line이 있는지 확인하여 만약 존재한다면 hit_count를 1만큼 증가시킨다. 그리고 해당 line의 lru 값을 1로 만든 뒤, set에서 해당 line을 제외한 나머지 line들의 lru값을 1만큼 증가시킨다. 이렇게 하여 가장 예전에 접근

한 line은 가장 큰 lru 값을 가질 수 있도록 하였다.

- hit이 안된 경우, is_set_full 변수를 사용하여 해당 set이 꽉 찼는지 확인한다. 해당 set이 꽉 차지 않은 경우, miss_count를 1만큼 증가시킨 후 line 하나를 채운 뒤, 위와 마찬가지로 lru 값을 update 해준다.
- 해당 set이 꽉 찬 경우, miss_count와 eviction_count를 모두 1만큼 증가시킨 후 lru 값이 가장 큰 line을 찾아 lru_line에 저장한다. 그런 다음 해당 line의 tag 값을 바꿔준 뒤 위와 마찬가지로 lru 값을 update 해준다.
- main 함수에서 cache access를 모두 마친 뒤 동적할당을 해제하고 hit_count, miss_count, eviction_count 수를 출력한 뒤 return한다.
- test-csim을 실행하여 얻은 결과는 다음과 같다.

```
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27

TEST_CSIM_RESULTS=27
```

2. Part B: Efficient Matrix Transpose

- Part B에서는 입력 행렬의 크기가 32*32, 64*64, 61*67인 경우로 나누어 문제를 해결하였다.
- 먼저 32*32 크기의 행렬의 경우 해결 방법은 다음과 같다. 문제에서 b가 5라고 하였으므로 block offset은 2^5= 32byte이고, 이는 int형 정수 8개가 들어가는 크기이므로 block size를 8로 설정하여 blocking을 진행하였다.
- 이때 행렬의 row번째와 column번째가 같을 때, 즉 행렬의 대각선 entry에 접근하는 경우 A, B의 set index가 동일하여 conflict miss를 발생시킨다. 이를 피하기 위해 행렬의 대각선 entry에 접근하는 경우를 tmp 변수에 저장해 두었다가 나중에 따로 처리해주었다.
- 64*64 크기의 행렬의 경우 위와 같은 방법을 사용하면 set index를 공유하는 경우가 많아 cache miss를 최소화할 수 없다. 대신 block size를 4로 blocking을 진행한 뒤, 각 block에 대해 12개의 지역변수 tmp1부터 tmp12까지 사용하여 A와 B 모두 row major order 방식으로 각각의 entry에 접근하여 transpose를 진행하였다.

- 61*67 크기의 행렬의 경우 block size를 16으로 증가시켜 blocking을 하는 것만으로도 cache miss를 2000회보다 적게 나올 수 있다. 이는 n*n 행렬을 block size B로 blocking을 하면 (1/4B) * n^3의 miss가 나오는데, B를 증가시켜 miss의 수가 줄어든 것으로 볼 수 있다.

- driver.py 파일을 활용해 Part A, B의 총 점수를 확인하면 다음과 같다.

```
Part A: Testing cache simulator
Running ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27


Part B: Testing transpose function
Running ./test-trans -M 32 -N 32
Running ./test-trans -M 64 -N 64
Running ./test-trans -M 61 -N 67

Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           3.8         8        1667
Trans perf 61x67          10.0        10        1992
          Total points    48.8        53
```