CSED211 Lab 8 & 9 Report

Cache Lab: Understanding Cache Memories

20220041 김유진

I. Method

이번 실습에서는 cache memory를 구성하고 직접 시뮬레이션해보며 cache를 익히는 것에 의의가 있다. Part A와 Part B로 구분되는데, Part A에서는 cache의 동작 방식을 시뮬레이션하고 Part B에서는 행렬을 전치시키는 연산을 구현하되 최대한 miss 횟수를 줄여 hit ratio를 높이도록 하는 것을 요구한다. 실습에 앞서 몇 가지 개념들을 짚어봐야 한다. Cache memory는 고속의 임시 데이터 보관소로 main memory로부터 데이터를 효과적으로 가져와 프로세서가 더 빠르게 실행되도록 하는 역할을 한다. Cache memory의 최소 단위는 cache line으로, 이 line은 valid, tag, block bit를 포함하고 있다. 이때 tag는 메모리의 주소를 식별하는 역할을 하며 cache는 cache line으로 구성되어 있는 cache set으로 구성되어 있다. Cache가 데이터에 접근하고자 할 때 hit, miss, eviction 이렇게 3가지 경우가 발생할 수 있다. 먼저 cache hit는 cache에 데이터를 요청했을 때 그 데이터가 이미 캐시에 존재하는 경우로, 데이터를 바로 찾아낼 수 있어 메모리에서 읽어오지 않아도 된다. cache miss의 경우에는 반대로 cache에 데이터를 요청했으나 해당 데이터가 캐시에 없어 main memory로부터 가져와야 하는 경우를 말한다. Eviction의 경우, 캐시가 이미 가득 찬 상태에서 새로운 데이터를 캐시에 넣어야할 때 이미 존재하는 데이터를 쫓아내는 과정인데, 일반적으로 LRU 알고리즘에 따라 선택된 데이터를 방출한다. 이때 LRU 알고리즘은 Least Recently Used의 약자로 가장 최근에 사용되지 않은 데이터를 우선적으로 버린다는 방식을 따른다. cache에 데이터가 접근될 때마다 해당 데이터의 사용 시간을 갱신하여, 캐시에 새로운 데이터를 넣어야 하는데 캐시가 가득차 eviction이 필요한 경우, 가장 오랫동안 사용되지 않은 데이터를 선택하여 방출하게 되는 것이다. Part B에서는 최적화된 행렬 전치를 요구한다. 공간 지역성을 지닌 cache의 특징에 따라 cache 데이터를 효율적으로 접근해야 하는데, 이때 cache blocking을 사용해 행렬을 적절한 크기로 나누어 작은 블록 단위로 전치를 수행한다. 이러한 개념들을 참고하여 실습을 수행함으로써 cache memory의 동작과 최적화에 대한 이해를 높인다.

II. Code Explanation

1) csim.c (Part A. Building a cache simulator)

csim.c에서는 파일을 통해 읽어온 정보로 cache를 구성하고, cache memory access를 시뮬레이션하여 그 과정에서 발생하는 hit, miss, eviction을 출력하는 것을 구현하고자 한다.

```c
typedef struct {
    int valid;
    int tag;
    int time;
} CacheLine;
typedef CacheLine* CacheSet;
typedef CacheSet* Cache;
```

CacheLine, CacheSet, Cache 구조체를 위와 같이 정의하여 캐시를 구성하고자 하였다. CacheLine은 valid, tag, time을 요소로 포함하고 있는데, 이때 time은 LRU를 구현하고자 추가하였다. 이를 사용해 데이터가 참조될 때마다 업데이트되도록 하여 eviction이 발생하여 데이터 교체가 필요한 시점에 가장 최근에 참조되지 않은 데이터로 교체할 수 있게끔 하였다.

```c
int hit = 0, miss = 0, evict = 0;
char filename[100];
int s = 0, S = 0, E = 0, b = 0;
unsigned long long time_counter = 0;

Cache cache = NULL;

void init_cache();
void access_cache(unsigned long long address);
void free_cache();
```

전역변수로 몇 가지를 정의하였다. 캐싱 과정에서 발생하는 hit, miss, evict 횟수를 저장할 변수들과, 읽어올 파일의 이름을 담을 filename 변수가 있다. 또한 캐시의 구조를 결정할 s, E, b 정보를 저장할 변수들과 추가로 cache set의 크기를 저장할 S 변수도 선언하고, time을 업데이트할 수 있도록 전역 time_counter를 정의하여 caching 과정에서 꾸준히 갱신해준다. Cache 데이터형으로 정의되어 있는 cache는 이 캐싱 시뮬레이션이 이루어질 전역 변수이고, 그 아래 세가지 함수는 캐시를 구현하는 데에 사용될 함수들이다.

```c
int main(int argc, char* argv[])
{
    int opt;
    while((opt = getopt(argc, argv, "s:E:b:t:")) != -1) {
        switch (opt) {
            case 's':
                s = atoi(optarg);
                S = (1 << s);
                break;
            case 'E':
                E = atoi(optarg);
                break;
            case 'b':
                b = atoi(optarg);
                break;
            case 't':
```

```
                strcpy(filename, optarg);
                break;
        }
    }

    FILE* file = fopen(filename, "r");
    assert(file);
    init_cache();
    char op; //operation
    unsigned long long address; //address
    int size; //number of bytes
    while(fscanf(file, " %c %llx %d", &op, &address, &size) != EOF) {
        switch (op) {
            case 'M': //access twice
                access_cache(address);
                access_cache(address);
                break;
            case 'L': //access once
            case 'S': //access once
                access_cache(address);
                break;
        }
    }
    printSummary(hit, miss, evict);
    free_cache();

    fclose(file);
    return 0;
}
```

main 함수는 위와 같이 구현하였다. getopt 함수와 argc, argc를 이용하여 주어지는 입력을 파싱해준다. s는 캐시의 set index bit 개수로 이를 앞서 선언해두었던 전역변수 s에 넣어주고 set의 개수인 S도 알맞게 값을 부여한다. 마찬가지로 set 하나 당 line 개수인 E, block bits 개수를 나타내는 b, trace file의 이름을 filename에 각각 저장한다. 입력받은 filename의 파일을 열어주고 앞서 입력받았던 정보들을 토대로 cache를 동적할당한다. init_cache() 함수는 아래와 같이 구현하였다.

```
void init_cache() { //allocate cache and initialize
    cache = (Cache)malloc(S * sizeof(CacheSet));
    for(int i = 0; i < S; ++i) {
        cache[i] = (CacheSet)malloc(E * sizeof(CacheLine));
        for(int j = 0; j < E; ++j) {
            cache[i][j].tag = 0;
            cache[i][j].valid = 0;
            cache[i][j].time = 0;
        }
    }
}
```

init_cache()는 이름에서 알 수 있듯이 캐시를 초기화해주는 함수이다. 캐시가 S개의 set으로 구성되어있으며 하나의 set은 E개의 line들로 구성되어있음을 이용하여 malloc 함수를 알맞게 호출하여 동적할당한다. 또한 반복문을 돌며 cache를 구성하는 tag, valid, time도 0으로 초기화해준다.

다시 main 함수를 이어서 살펴보면, 열었던 filename의 파일로부터 option, address, size를 읽어온다. 주어진 option에 따라 알맞은 memory access를 수행하면 된다. 이 코드에서는 L, S, M을 구현하고 있는데 L은 data load, S는 data store, M은 data modify의 기능을 한다. L과 S는 memory access를 한번만 하고 있지만 M은 data load 후 data store까지 하므로 memory access를 두번 한다는 점을 활용해 access_cache() 함수를 횟수에 알맞게 호출한다.

```
void access_cache(unsigned long long address) { //access cache
    int tag = address >> (s + b); //tag bit
    int set_idx = (address >> b) & ((1 << s) - 1); //set index bit
    int hit_flag = 0, evict_flag = 1;
    int replace_idx = 0;

    CacheSet cache_set = cache[set_idx];
    int tmp_idx = 0; //temporary block idx
    for(tmp_idx = 0; tmp_idx < E; ++tmp_idx) {
        if(cache_set[tmp_idx].valid ) { //valid = 1
            if(cache_set[tmp_idx].tag == tag) { //check if tag is same
                hit_flag = 1;
                break;
            }
        }
    }
    /*hit*/
    if(hit_flag) {
        cache_set[tmp_idx].time = time_counter;
        ++hit;
    }
    /*miss*/
    else {
        ++miss;
        for(int i = 0; i < E; i++) { //check if there are empty lines
            if(!cache_set[i].valid) {
                replace_idx = i;
                evict_flag = 0;
                break;
            }
        }
        if(evict_flag) { //line replacement through lru
            ++evict;
            unsigned long long evict_time = -1;
            for(int i = 0; i < E; ++i) {
                if(cache_set[i].time < evict_time) {
```

```
                    evict_time = cache_set[i].time;
                    replace_idx = i;
                }
            }
        }
        cache_set[replace_idx].valid = 1;
        cache_set[replace_idx].tag = tag;
        cache_set[replace_idx].time = time_counter;
    }
    ++time_counter;
}
```

먼저 tag, set_idx 변수를 이용해 address로부터 tag와 set_idx 정보를 추출한다. 쉬프트 연산을 적절히 수행하므로써 구현하였다. 그리고 hit_flag와 evict_flag 변수를 선언하여 hit가 일어났는지의 여부, eviction이 발생하였는지의 여부를 표현하고자 했다. 새롭게 데이터를 넣어줄 line의 인덱스를 저장할 replace_idx도 선언하고, CacheSet 타입의 cache_set을 선언해주어 다루고자 하는 set_idx의 cache set만 담을 수 있게 한다.

첫번째 tmp_idx를 이용한 반복문을 통해 hit가 발생하는지를 확인한다. 해당 set 내의 line들을 돌면서 만약 valid가 1이고, tag가 앞서 추출한 tag 정보와 같다면 hit가 발생한 것이므로 hit_flag을 1로 한 후 break하여 반복문을 나온다. 만약 hit_flag가 1, 즉 hit가 발생했다면 hit 카운트를 1 증가시키고, 해당 line의 time으로 time_counter 값을 넣어주어 업데이트한다.

hit가 발생하지 않은 경우는 miss가 발생하였다는 것이므로 miss 카운트를 1 증가시킨 후, eviction이 발생하는지를 따져보아야한다. 이때는 반복문을 이용해 해당 set의 line들을 확인하면서 valid가 0, 즉 비어있는지를 확인하고 만약 그렇다면 evict_flag를 0으로 설정해주어 eviction이 발생하지 않도록 한다. 반복문을 돌고 난 후 evict_flag가 1, 즉 LRU에 따라 replacement가 필요한 경우, evict 카운트를 1 증가시켜주며, 어떤 line을 교체시킬 것인가를 LRU에 따라 조사해야 한다. 반복문을 돌며 가장 작은 time을 가진 line을 찾아서 이를 replace_idx로 설정해준다.

miss가 났지만 valid가 0인 line이 있어 line replacement가 필요 없는 경우와 valid가 1인 line이 있어 eviction이 발생해 line replacement가 필요한 경우, 이 두가지 경우에 대한 replace_idx를 알맞게 설정해주었으므로 남은 것은 cache_set의 해당 인덱스 line의 valid, tag, time 정보를 업데이트 해주는 일이다. valid는 1로, tag는 tag 정보로, time은 time_counter로 업데이트한다.'

hit 또는 miss(eviction)을 모두 처리한 이후에는 전역 time_counter 변수를 1 증가시킨다.

주어진 option의 memory access를 마치고 나면 기록했던 hit, miss, evict를 출력하고 동적할당해주었던 cache를 free해주는 과정이 필요하다.

```
void free_cache() { //deallocate cache
    for(int i = 0; i < S; ++i) {
        free(cache[i]);
    }
```

```
    free(cache);
}
```

반복문을 돌며 cache를 할당해주고 앞서 열었던 file을 다시 닫음으로써 csim을 종료시킨다.

```
                        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-csim을 실행시키면 다음과 같이 시뮬레이션이 유사하게 돌아갔음을 확인할 수 있다.

2) trans.c (Part B. Efficient Matrix Transpose)

trans.c에서는 행렬을 전치시키는 방식을 구현하여 cache miss가 되도록 적게 나타나도록 하는 것에 목표가 있다. cache의 지역성을 이용한 코드를 작성하는 것이 핵심이다.

이번 실습에서 주어진 cache의 정보는 directly mapped (E=1), block size가 32bytes (b=5), set 개수가 32개 (s=5)라는 것이다. 또한 평가에 사용될 3가지 행렬들은 각각 32x32, 64x64, 61x67 형태의 행렬들로 각 행렬 크기를 고려하여 최대한 최적화할 수 있도록 구현하였다.

먼저 32x32 행렬의 경우는 다음과 같은 연산을 하도록 하였다.

```
if(M == 32 && N == 32) {
        /*copy each values from A to B*/
        int t0, t1, t2, t3, t4, t5, t6, t7;
        for(int row = 0; row < N; row += 8) {
            for(int col = 0; col < M; col += 8) {
                for(int i = row; i < row + 8; ++i) {
                    t0 = A[i][col];
                    t1 = A[i][col+1];
                    t2 = A[i][col+2];
                    t3 = A[i][col+3];
                    t4 = A[i][col+4];
                    t5 = A[i][col+5];
                    t6 = A[i][col+6];
                    t7 = A[i][col+7];

                    B[col][i] = t0;
                    B[col+1][i] = t1;
                    B[col+2][i] = t2;
                    B[col+3][i] = t3;
```

```
                    B[col+4][i] = t4;
                    B[col+5][i] = t5;
                    B[col+6][i] = t6;
                    B[col+7][i] = t7;
                }
            }
        }
    }
```

앞서 주어진 cache의 정보에 따르면 block의 크기는 32bytes로 총 8개의 int를 저장할 수 있다. 이 점을 이용해 8x8 형태의 블록으로 나누어 전치를 진행하도록 하였다. t0~t7이라는 임시변수를 선언하여 A에서 값을 읽어 저장한 후, 이를 알맞게 B에 옮겨주는 방식으로 구현하였다.

```
[kimyujin1224@programming2 cachelab-handout]$ ./test-trans -M 32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=1 misses=287

TEST_TRANS_RESULTS=1:287
```

test-trans 프로그램을 통해 287 miss가 발생하여 평가 기준을 통과한 것을 확인하였다.

그다음 64x64 행렬의 경우이다.

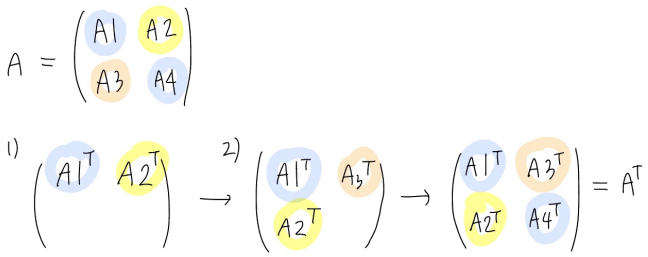
```
    else if(M == 64 && N == 64) {
        /* 64X64 matrix A = | A1 A2 |
                            | A3 A4 |  */
        int t0, t1, t2, t3, t4, t5, t6, t7;
        int row, col, i, j;
        for(row = 0; row < N; row += 8) {
            for(col = 0; col < M; col += 8) {
                for(i = row; i < row + 4; ++i) { //A1, A2 transpose
                    t0 = A[i][col];
                    t1 = A[i][col+1];
                    t2 = A[i][col+2];
                    t3 = A[i][col+3];
                    t4 = A[i][col+4];
                    t5 = A[i][col+5];
                    t6 = A[i][col+6];
                    t7 = A[i][col+7];
                    B[col][i] = t0;
                    B[col+1][i] = t1;
```

```
                    B[col+2][i] = t2;
                    B[col+3][i] = t3;
                    B[col][i+4] = t4;
                    B[col+1][i+4] = t5;
                    B[col+2][i+4] = t6;
                    B[col+3][i+4] = t7;
                }
                for(j = 0; j < 4; ++j) { //A2, A3 transpose(change location)
                    t0 = A[row+4][col+j];
                    t1 = A[row+5][col+j];
                    t2 = A[row+6][col+j];
                    t3 = A[row+7][col+j];

                    t4 = B[col+j][row+4];
                    t5 = B[col+j][row+5];
                    t6 = B[col+j][row+6];
                    t7 = B[col+j][row+7];

                    B[col+j][row+4] = t0;
                    B[col+j][row+5] = t1;
                    B[col+j][row+6] = t2;
                    B[col+j][row+7] = t3;

                    B[col+j+4][row] = t4;
                    B[col+j+4][row+1] = t5;
                    B[col+j+4][row+2] = t6;
                    B[col+j+4][row+3] = t7;

                }
                for(i = row + 4; i < row + 8 && i < N; ++i) { //A4 transpose
                    t4 = A[i][col+4];
                    t5 = A[i][col+5];
                    t6 = A[i][col+6];
                    t7 = A[i][col+7];
                    B[col+4][i] = t4;
                    B[col+5][i] = t5;
                    B[col+6][i] = t6;
                    B[col+7][i] = t7;
                }
            }
        }
    }
```

이 경우를 아까 32x32 행렬을 전치시켰던 방식과 비슷한 방식으로 전치하였다면 충돌이 발생하는 부분이 생긴다. 따라서 새롭게 고안한 방식은 A 행렬을 4x4 형태의 블록 4개(A1, A2, A3, A4)로 나누어 각 블록마다 전치를 시키는 방식이다.

row, col 이중 반복문을 돌면서 각 블록마다 알맞은 전치를 할 수 있게 한다. 이중 반복문 내에

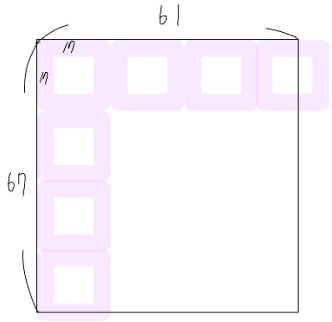
첫번째 반복문은 A1과 A2를 전치시키고, 두번째 반복문은 전치시켰던 A2^T를 올바른 위치에 배치하고 동시에 A3을 전치시킨다. 마지막 반복문은 A4^T는 기존 A4 위치와 같은 곳에 있으므로 전치만 시켜준다. 간략한 그림으로 나타내면 다음과 같다.



```
[kimyujin1224@programming2 cachelab-handout]$ ./test-trans -M 64 -N 64

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:9066, misses:1179, evictions:1147

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3474, misses:4723, evictions:4691

Summary for official submission (func 0): correctness=1 misses=1179

TEST_TRANS_RESULTS=1:1179
```

총 1179 miss가 발생하여 평가 기준을 통과하였다.

마지막 61x67 행렬의 경우이다.



61x67 행렬은 정사각 행렬이 아닌 형태를 띠고 있어 block size를 다른 방식으로 가정해야 한다. 61과 64는 17*4의 값인 68과 가까운 크기이기 때문에 cache miss를 최소화하도록 활용하기에 적합하다. block size를 17로 설정하면 다음과 같이 블록을 나눌 수 있다.

A의 row와 col을 도는 이중반복문 내에 또 하나의 이중반복문을 통해 17x17 형태의 블록을 사용하여 전치를 하도록 했다.

```
else if(M == 61 && N == 67) {
    int tmp;
    for(int row = 0; row < N; row += 17) {
        for(int col = 0; col < M; col += 17) {
            for(int i = row; i < row + 17 && i < N; ++i) {
                for(int j = col; j < col + 17 && j < M; ++j) {
                    tmp = A[i][j];
                    B[j][i] = tmp;
```

```
                    }
                }
            }
        }
    }
    return;

}
```

```
[kimyujin1224@programming2 cachelab-handout]$ ./test-trans -M 61 -N 67

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6229, misses:1950, evictions:1918

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3756, misses:4423, evictions:4391

Summary for official submission (func 0): correctness=1 misses=1950

TEST_TRANS_RESULTS=1:1950
```

총 1950 miss가 발생하여 평가 기준을 통과하였다.

최종적으로 driver.py를 이용해 test-csim과 test-trans를 실행시킨 결과는 다음과 같다.

```
[kimyujin1224@programming2 cachelab-handout]$ python2 driver.py
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           8.0         8        1179
Trans perf 61x67          10.0        10        1950
          Total points    53.0        53
```