2023 Fall CSED211

Lab 5(Cache Lab) Report

학번 : 20220312
이름 : 박준혁

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

1. csim.c

A. 캐시 구조

본 과제에서는 구조체를 활용, 포인터 변수를 사용해 라인, 과 세트로 구성되어 있는 캐시를 구현하였다.

```
//cache struct: sets - lines - {valid,tag,lru}
typedef struct{
    bool valid;
    int tag;
    int lru;
} line_t;

typedef struct{
    line_t* lines;
} set_t;

typedef struct{
    set_t* sets;
} cache_t;

//initialize
cache_t cache = {};
```

B. 캐시 초기화: 동적 할당

```
//cache init; dynamic allocation and initialize
void InitCache(){
    cache.sets = (set_t*)malloc(sizeof(set_t) * S);
    for (int i=0; i<S; i++){
        cache.sets[i].lines = (line_t*)malloc(sizeof(line_t)*E);
        for (int j=0; j<E; j++){
            cache.sets[i].lines[j].valid=0;
            cache.sets[i].lines[j].tag=0;
            cache.sets[i].lines[j].lru=0;
        }
    }
}
```

C. 트레이스 파일 파싱

I로 시작하는 줄은 무시하고, L,S,M에 따라 시뮬레이션을 한다.

이때 M은 두번 시뮬레이션 한다.

```c
//Parse Trace: read trace file, simulate cache
void ParseTrace(){
    FILE* traceFile = fopen(TraceFile, "r");
    char cmd;
    int address;
    int size;
    while(fscanf(traceFile, " %c %x,%d", &cmd, &address, &size)!=EOF){
        //'I' line is ignored
        switch(cmd){
            case 'L':
                Simulate(address);
                break;
            case 'S':
                Simulate(address);
                break;
            case 'M':
                //simulate twice
                Simulate(address);
                Simulate(address);
                break;
            default:
                break;
        }
    }
    fclose(traceFile);
}
```

```
1    S 00600aa0,1
2   I  004005b6,5
3   I  004005bb,5
4   I  004005c0,5
5    S 7ff000398,8
6   I  0040051e,1
7    S 7ff000390,8
8   I  0040051f,3
9   I  00400522,4
```

D. 시뮬레이션

i. 주소 읽기

set index, tag bit을 구한다. 입력받은 s자리, b자리를 기반으로 계산한다.

```
void Simulate(int address){
    //extract information from address
    size_t setIndex = (address >> b_bit) & (0x7fffffff>>(31-s_bit));
    size_t tag = (address >> (s_bit+b_bit)) & (0x7fffffff);
    //set
    set_t set_c = cache.sets[setIndex];
```

ii. hit, miss

일치하는 태그가 존재하고 valid 한 라인이 있으면 hit이다.

hit에 실패하면 miss이다.

```
    //cache hit
    for (int i=0; i < E ; i++){
        line_t line_c = set_c.lines[i];
        if (!line_c.valid) {
            continue;
        }
        if(tag!=line_c.tag){
            continue;
        }
        LRUCounter++;
        HitCount++;
        set_c.lines[i].lru=LRUCounter;
        return;
    }

    //cache miss
    MissCount++;
```

iii. eviction

본 프로그램에서는 LRU Policy를 적용, 매 hit마다 lru를 1씩 올리고 라인에 그 값을 저장해 놓는다. 이러면 오래 될수록 작은 lru를 가지게 되고, 오래된 라인부터 교체할 수 있다. 따라서 가장 작은 lru를 가진 라인을 찾아 교체해 준다.

```
//replace line based on LRU
unsigned int evictionLine = 0;
unsigned long long int evictionLRU = ULONG_MAX;
for (int i=0; i < E ; i++){
    line_t line_c = set_c.lines[i];
    if (evictionLRU > line_c.lru){
        evictionLRU = line_c.lru;
        evictionLine = i;
    }
}

//cache eviction
if(set_c.lines[evictionLine].valid){
    EvictionCount++;
}

LRUCounter++;
set_c.lines[evictionLine].valid=1;
set_c.lines[evictionLine].tag=tag;
set_c.lines[evictionLine].lru=LRUCounter;
```

E. 메모리 할당 해제: 동적할당 해제

```
//deallocate dynamic allocation
void FreeCache(){
    for(int i=0; i<S;i++){
        free(cache.sets[i].lines);
    }
    free(cache.sets);
}
```

F. main 함수

main에서는 getopt를 사용해 instruction을 읽고, 이에 따라 S,E,B를 정하고 trace 파일을 정한다. 맨 마지막에 hit, miss, eviction 개수를 출력해 준다.

최종 결과

```
[joon363@programming2 cachelab-handout]$ ./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

TEST CSIM RESULTS=27
```

2. trans.c

A. 32*32 matrix

캐시 라인의 크기가 32바이트 이므로, 4바이트 int 8개씩 끊어서 접근해주면 miss rate를 줄일 수 있다.

```
void transpose_32_v2(int M, int N, int A[N][M], int B[M][N])
{
    int i,j,k;
    for(i=0; i<M; i+=8)
    {
        for(j=0; j<N; j+=8)
        {
            for(k=0;k<8;k++){
                int temp1=A[i+k][j+0];
                int temp2=A[i+k][j+1];
                int temp3=A[i+k][j+2];
                int temp4=A[i+k][j+3];
                int temp5=A[i+k][j+4];
                int temp6=A[i+k][j+5];
                int temp7=A[i+k][j+6];
                int temp8=A[i+k][j+7];

                B[j+0][i+k]=temp1;
                B[j+1][i+k]=temp2;
                B[j+2][i+k]=temp3;
                B[j+3][i+k]=temp4;
                B[j+4][i+k]=temp5;
                B[j+5][i+k]=temp6;
                B[j+6][i+k]=temp7;
                B[j+7][i+k]=temp8;
            }
        }
    }
}
```

B에 바로 A를 대입하는 것보다 temp를 거쳐서 대입하는 것이 더 miss가 적어서 위와 같이 짰다.

B. 64*64

32*32와 비슷하나, 크기 8의 블록으로 쪼개는 것 보다 크기 4의 블록으로 쪼개는 것이 더 miss가 적게 나왔다.

따라서 4*4의 행렬 전치를 통째로 loop에 돌린다.

```
void transpose_64(int M, int N, int A[N][M], int B[M][N]){
    int r, blockSize, c;
    int temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8;
    int temp9, temp10, temp11, temp12, temp13, temp14, temp15, temp16;

    blockSize = 4;
    for(r = 0; r < N; r += blockSize)
    {
        for(c = 0; c < M; c += blockSize)
        {
            temp1 = A[r+0][c+3];
            temp2 = A[r+1][c+3];
            temp3 = A[r+2][c+3];
            temp4 = A[r+0][c+2];
            temp5 = A[r+1][c+2];
            temp6 = A[r+2][c+2];
            temp7 = A[r+0][c+1];
            temp8 = A[r+1][c+1];
            temp9 = A[r+2][c+1];
            temp10 = A[r+0][c+0];
            temp11 = A[r+1][c+0];
            temp12 = A[r+2][c+0];
            temp13 = A[r+3][c+0];
            temp14 = A[r+3][c+1];
            temp15 = A[r+3][c+2];
            temp16 = A[r+3][c+3];
```

```
            B[c+3][r+0] = temp1;
            B[c+3][r+1] = temp2;
            B[c+3][r+2] = temp3;
            B[c+2][r+0] = temp4;
            B[c+2][r+1] = temp5;
            B[c+2][r+2] = temp6;
            B[c+1][r+0] = temp7;
            B[c+1][r+1] = temp8;
            B[c+1][r+2] = temp9;
            B[c+0][r+0] = temp10;
            B[c+0][r+1] = temp11;
            B[c+0][r+2] = temp12;
            B[c+0][r+3] = temp13;
            B[c+1][r+3] = temp14;
            B[c+2][r+3] = temp15;
            B[c+3][r+3] = temp16;
        }
    }
}
```

C. 61*67

블록 사이즈를 4,8,12,16,17,18,19,20,32로 바꿔가면서 브루트 포스를 진행, 17과 18에서 최저 miss 를 달성할 수 있었다.

```
char transpose_61_v1_desc[] = "61 v1"; //2425
void transpose_61_v1(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(4,M,N,A,B);
}
char transpose_61_v2_desc[] = "61 v2"; //2215
void transpose_61_v2(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(8,M,N,A,B);
}
char transpose_61_v3_desc[] = "61 v3"; //2088
void transpose_61_v3(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(12,M,N,A,B);
}
char transpose_61_v4_desc[] = "61 v4"; //2002
void transpose_61_v4(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(16,M,N,A,B);
}
char transpose_61_v5_desc[] = "61 v5"; //1970
void transpose_61_v5(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(17,M,N,A,B);
}
char transpose_61_v6_desc[] = "61 v6"; //1970
void transpose_61_v6(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(18,M,N,A,B);
}
char transpose_61_v7_desc[] = "61 v7"; //1983
void transpose_61_v7(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(19,M,N,A,B);
}
```

```
char transpose_61_v8_desc[] = "61 v8"; //1984
void transpose_61_v8(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(20,M,N,A,B);
}
char transpose_61_v9_desc[] = "61 v9"; //3856
void transpose_61_v9(int M, int N, int A[N][M], int B[M][N]){
    transpose_61(32,M,N,A,B);
}
```

```
void transpose_61(int b, int M, int N, int A[N][M], int B[M][N]){
    int i,j,p,q;
    int blocksize=b;
    for(i=0; i<M; i+=blocksize)
    {
        for(j=0; j<N; j+=blocksize)
        {
            for(p=i; p<(i+blocksize) && (p<M); p++)
            {
                for(q=j; q<(j+blocksize) && q<N; q++)
                {
                    B[p][q]=A[q][p];
                }
            }
        }
    }
}
```

D. 성능 테스트 결과

i. 32*32: 287 miss

```
[joon363@programming2 cachelab-handout]$ ./test-trans -M 32 -N 32

Function 0 (1 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

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

TEST_TRANS_RESULTS=1:287
```

ii. 64*64: 1603 miss

```
[joon363@programming2 cachelab-handout]$ ./test-trans -M 64 -N 64

Function 0 (1 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6594, misses:1603, evictions:1571

Summary for official submission (func 0): correctness=1 misses=1603

TEST_TRANS_RESULTS=1:1603
```

iii. 61*67: 1970

```
[joon363@programming2 cachelab-handout]$ ./test-trans -M 61 -N 67

Function 0 (1 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6209, misses:1970, evictions:1938

Summary for official submission (func 0): correctness=1 misses=1970

TEST_TRANS_RESULTS=1:1970
```

3. 참고 문헌

- Cache Memories, Cache Complexity, Marc Moreno Maza, University of Western Ontario, London
  url: https://www.csd.uwo.ca/~mmorenom/HPC-Slides/Cache_Complexity.pdf