2019-2학기

컴퓨터 SW시스템 개론

Lab #5

20180085 무은재학부 송수민

Part A : Writing a cache simulator

```
typedef struct {
    bool valid;
    int tag;
    int counter;
}Line;
typedef struct {
    Line *lines;
}Set;
typedef struct {
    Set *set;
    size_t num_set;
    size_t num_line;
}Cache;
```

위와 같이 가상 Cache를 구조체를 사용하여 설계하였다. Line에는 Valid, tag, 그리고 후에 사용 할 LRU Algorithm을 위한 Counter을 member variable로 넣었고, set은 이 라인들의 연속으로 이루어지게 하였다. Cache는 이 set들의 구성이고 set의 수와 line의 수를 member variable로 넣었다.

```
int main(int argc, char *argv[])
{
    FILE *file = 0;
    for (int option; (option = getopt(argc, argv, "s:E:b:t:")) != -1;) {
        switch(option){
            case 's':
                init_set = atoi(optarg);
                cache.num_set = 2 << init_set;
                break;
            case 'E':
                cache.num_line = atoi(optarg);
                break;
            case 'b':
                init_block = atoi(optarg);
                break;
            case 't':
                if (!(file = fopen(optarg, "r")))
                    return 1;
                break;
            default:
                return 1;
        }
    }
    if (!init_set || !(cache.num_line) || !init_block || !file)
        return 1;
    cache.set = malloc(sizeof(Set)*cache.num_set);
    for (int i = 0; i < cache.num_set; i++) {
        cache.set[i].lines = calloc(sizeof(Line), cache.num_line);
    }
```

Optarg method를 사용하여 과제에서 주어지는 Cache의 스펙을 입력하고, 이를 바탕으로 Cache를 malloc을 통해 동적할당한다.

```
char mode;
int address;
while (fscanf(file, " %c %x%*c%*d", &mode, &address) != EOF) {
    if (mode == 'I')
        continue;

    run(address);
    if (mode == 'M')
        run(address);
}
printSummary(hit, miss, eviction);
fclose(file);
for (size_t i = 0; i < cache.num_set; i++)
    free(cache.set[i].lines);
free(cache.set);
return 0;
```

Instruction Mode와 주소를 받고, I면 한번 무시하고, M이면 같은 자리에 2번 Access한다.

```
void run(int address) {
    size_t set_index = (0x7fffffff >> (31 - init_set)) & (address >> init_block);
    int tag = 0xffffffff & (address >> (init_set + init_block));
    Set *set = &cache.set[set_index];
    for (size_t i = 0; i < cache.num_line; i++) {
        Line* line = &set->lines[i];
        if (!line->valid)
            continue;
        if (line->tag != tag)
            continue;
        ++hit;
        update(set, i);
        return;
    }
    ++miss;
    for (size_t i = 0; i < cache.num_line; i++) {
        Line* line = &set->lines[i];
        if (line->valid)
            continue;
        line->valid = true;
        line->tag = tag;
        update(set, i);
        return;
    }
    ++eviction;
    for (size_t i = 0; i < cache.num_line; i++) {
        Line* line = &set->lines[i];
        if (line->counter)
            continue;
        line->valid = true;
        line->tag = tag;
        update(set, i);
        return;
    }
}
```

넘겨진 address를 비트연산자로 masking하여 어떤 셋인지 선택하고, line을 for문을 통해 순회하며 validbit가 1이되어있고, tag가 같은 캐쉬를 찾는다. 찾으면 hit이다. 찾지 못한다면 miss이고 빈 자리의 캐쉬를 찾기 위해 캐쉬를 계속 순회하며, 만약 빈 자리가 없다면 아까 언급하였던 counter의 값이 0인 곳의 캐시 Line을 지우고 캐쉬를 작성한다.

```
void update(Set *set, size_t line_update) {
    Line *line = &set->lines[line_update];
    for (size_t i = 0; i < cache.num_line; i++) {
        Line *iterator = &set->lines[i];
        if (!iterator->valid)
            continue;
        if (iterator->counter <= line->counter)
            continue;

        --iterator->counter;
    }
    line->counter = cache.num_line - 1;
}
```

이 Method는 counter를 update하는 method이다. 어떠한 캐쉬에 Access 할 때, 그 캐쉬 Line보다 덜 쓰인 캐쉬 Line의 counter가 1씩 감소한다.

Result :

```
root@ssm-virtual-machine:/mnt/hgfs/SW/Lab5/cachelab-handout# ./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 : Optimizing Transpose Function

Part B의 핵심은 Miss를 최소화 하는것이다. 과제에서 주어진 Cache는 다음과 같다.

I. Directed Mapped Cache

II. Set = 32

III. Capacity/Line = 32bytes

I. 32X32

32 X 32는 8 x 8으로 나누어 풀이하면 miss를 최소화 할 수 있다. 단 B[I][J] = A[J][I] 는 eviction을 일으키고 i==j 인 경우의 연산을 8x8 블록별 연산 중 마지막 과정에 처리를 하면 연산과정을 줄일 수 있다.

```
for (int i = row; i < row + 8 && i < N; i++)
{
    int tmp = A[i][i];
    for (int j = col; j < col + 8 && j < M; j++)
    {
        if (i == j)
            continue;
        B[j][i] = A[i][j];
    }
    B[i][i] = tmp;
}
```

II. 61 X 67

이 경우의 8X8 로 나누어 연산하면 끝이다.

```
for (int i = col; i < col + 8 && i < M; i++)
{
    for (int j = row; j < row + 8 && j < N; j++)
    {
        B[i][j] = A[j][i];
    }
}
```

III. 64 X 64

이 경우가 처리하기 굉장히 어려웠다.

먼저 4개의 임시 변수를 A의 left lower corner 부분에 복사한다. 이 4개의 변수는 B의 right upper corner부분에 저장된다. 4개의 임시 변수가 저장된 A의 left lower corner 부분의 한 열을 B의 right upper corner로 이동시킨다. 4개의 임시 변수가 저장된 B의 right upper corner 부분의 한 행을 B의 right lower corner 부분으로 이동시킨다. 일련의 과정을 거치면 B의 왼쪽 아래, 위, 오른

쪽 위가 완성되고, 직접 오른쪽 하단 부분을 setting해주면 완성된다.

```
if (M == 64 && N == 64)
{
    int i, j, k, l;
    int temp1, temp2, temp3, temp4, temp5, temp6, temp7, temp8;
    for (i = 0; i < N; i += 8)
    {
    for (j = 0; j < M; j += 8)
    {
        for (k = i; k < i + 4; k++)
        {
            temp1 = A[k][j];
            temp2 = A[k][j + 1];
            temp3 = A[k][j + 2];
            temp4 = A[k][j + 3];
            temp5 = A[k][j + 4];
            temp6 = A[k][j + 5];
            temp7 = A[k][j + 6];
            temp8 = A[k][j + 7];
            B[j][k] = t1;
            B[j + 1][k] = temp2;
            B[j + 2][k] = temp3;
            B[j + 3][k] = temp4;
            B[j][k + 4] = temp5;
            B[j + 1][k + 4] = temp6;
            B[j + 2][k + 4] = temp7;
            B[j + 3][k + 4] = temp8;
        }
        for (l = j + 4; l < j + 8; l++) {

            temp5 = A[i + 4][l - 4];
            temp6 = A[i + 5][l - 4];
            temp7 = A[i + 6][l - 4];
            temp8 = A[i + 7][l - 4];
            t1 = B[l - 4][i + 4];
            temp2 = B[l - 4][i + 5];
            temp3 = B[l - 4][i + 6];
            temp4 = B[l - 4][i + 7];
            B[l - 4][i + 4] = temp5;
            B[l - 4][i + 5] = temp6;
            B[l - 4][i + 6] = temp7;
            B[l - 4][i + 7] = temp8;
            B[l][i] = t1;
            B[l][i + 1] = temp2;
            B[l][i + 2] = temp3;
            B[l][i + 3] = temp4;
            B[l][i + 4] = A[i + 4][l];
            B[l][i + 5] = A[i + 5][l];
            B[l][i + 6] = A[i + 6][l];
            B[l][i + 7] = A[i + 7][l];
        }
    }
}
return;
```

Result :

```
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        1171
Trans perf 61x67          10.0        10        1931
          Total points    53.0        53
```