시스템 프로그래밍 Lab4 Report

경영학과 2017-15108
박지상

1. 실행결과

```
2017-15108@sp2:~/HW4$ ./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           3.4         8        1699
Trans perf 61x67           6.8        10        2319
          Total points    45.2        53
```

2. 구현 방법 : csim.c – (1) Variable 선언

```c
// Jisang Park 2017-15108
#include "cachelab.h"
#include <stdio.h>
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>


// Command Line Arguments
int v; // Optional verbose flag that displays trace info
int s; // Number of set index bits (S = 2^s is the number of sets)
int E; // Associativity (number of lines per set)
int b; // Number of block bits (B = 2^b is the block size)
char* trace; // Name of the valgrind trace to display

// Cache Counts
int hit_count = 0;
int miss_count = 0;
int eviction_count = 0;

// Cache Structure
typedef struct{
    int valid_bit;
    int tag;
    int counter; // Least Recently Used Counter (The Bigger, More Recent)
} cache_line;

cache_line** cache;

int accessCNT = 0; // To manage LRU counter
```

Variable 선언

1) Command Line Arguments

2) Cache Counts

3) Cache Structure
cache는 cache_line[S][B]와 같은 형태의 matrix이므로,
struct cache_line의 2차원 배열을 가리키는 포인터로서
cache변수를 선언한다.

4) LRU counter
Cache에 access된 횟수를 모두 세는 변수로서,
cache line의 counter에 쓰여질 값이다.

0부터 시작하며, cache에 access할 때마다 1씩 증가한다.

2. 구현 방법 : csim.c – (2) main()

```c
int main(int argc, char **argv)
{
    // Parse Command Line
    int opt;
    /* looping over arguments */
    while(-1 != (opt = getopt(argc, argv, "hvs:E:b:t:"))){
        /* determine which argument it's processing */
        switch(opt){
            case 'h': break;
            case 'v': v= 1; break;
            case 's': s = atoi(optarg); break;
            case 'E': E = atoi(optarg); break;
            case 'b': b = atoi(optarg); break;
            case 't': trace = optarg; break;
            default: exit(1);
        }
    }
```

Main함수의 (1) Parse Command Line

getopt()함수를 활용하여 key의 value를 variable에 쓴다.

```c
// Cache Init
int S = 1 << s; // Number of sets

cache = (cache_line**)malloc(sizeof(cache_line*) * S);

for(int i = 0; i < S; i++){
    cache[i] = (cache_line*)malloc(sizeof(cache_line) * E);
    for(int j = 0; j < E; j++){
        cache[i][j].valid_bit = 0;
        cache[i][j].tag = 0;
        cache[i][j].counter = 0;
    }
}
```

Main함수의 (2) Cache init

Cache_line의 2차원 배열인 cache를 초기화한다.

2. 구현 방법 : csim.c – (2) main()

```c
// Open Trace File
FILE *pFile; // pointer to FILE object
pFile = fopen(trace, "r");

char identifier;
unsigned long long address;
int size;
// Reading lines like " M 20, 1" or "L 19, 3"
while(fscanf(pFile, " %c %llx, %d", &identifier, &address, &size) > 0){
    if(v) printf("%c %llx, %d", identifier, address, size);
    switch(identifier){
        case 'I': break;
        case 'L': cacheAccess(address); break;
        case 'M': cacheAccess(address); cacheAccess(address); break;
        case 'S': cacheAccess(address); break;
        default: break;
    }
    printf("\n");
}
```

Main함수의 (3) Open trace File

fscanf()로 trace file을 연 뒤,
instruction을 한 줄씩 읽어 그에 맞게 실행한다.

```c
printSummary(hit_count, miss_count, eviction_count);
fclose(pFile); // remember to close file when done

// Free Malloced Cache
for(int i = 0; i < S; i ++){
    free(cache[i]);
}
free(cache);

return 0;
```

Main함수의 (4) Free Cache

malloc()으로 메모리를 할당했던 cache를 free한다.

2. 구현 방법 : csim.c – (3) cacheAccess()

```c
void cacheAccess(unsigned long long address){
    // Cache : tag (t bits) + set index (s bits) + block offset (b bits)
    int tag = address >> (s+b); // Left-most bits
    int set_index = (address >> b) - (tag << s);


    // Iterate to check Cache Hit
    for(int i = 0; i < E; i++){
        if(cache[set_index][i].valid_bit && cache[set_index][i].tag == tag){
            // Cache Hit
            if(v) printf(" hit");
            cache[set_index][i].counter = accessCNT++;
            hit_count++;
            return;
        }
    }
```

1. Address로부터 tag bit, index bit 추출

2. set_index에 있는 cache_line들에 대해 iterate

   1) cache_hit : counter를 update하고, hit_count++
   2) cache_miss : miss_count++ & LRU line 찾기
      if) LRU line이 valid : eviction_count++
      cache 값 update

```c
    // Cache Miss
    if(v) printf(" miss");
    miss_count++;

    // Variables for LRU replacement method
    accessCNT++;
    int oldest_cnt = accessCNT; // To store the least recently used counter
    int oldest_idx = 0; // To store the least recently used cache line

    // Iterate Update the least recently used cache line
    for(int i = 0; i < E; i++){
        if(cache[set_index][i].counter < oldest_cnt){
            oldest_cnt = cache[set_index][i].counter;
            oldest_idx = i;
        }
    }

    // If LRU cache line is valid, then evict
    if(cache[set_index][oldest_idx].valid_bit){
        if(v) printf(" eviction");
        eviction_count++;
    }

    cache[set_index][oldest_idx].valid_bit = 1;
    cache[set_index][oldest_idx].tag = tag;
    cache[set_index][oldest_idx].counter = accessCNT++;
```

2. 구현 방법 : trans.c – (1) 32 X 32

```
// s = 5, E = 1, b = 5 --> 32bytes per cache line
// 32 bytes per cache line == 8 integers per cache line

// Total 11 Variables for Transpose (Limit: 12)
int i, j, k = 0;
int x1, x2, x3, x4, x5, x6, x7, x8 = 0;
```

평가가 되는 cache의 환경이
s = 5 → 32개의 set
E = 1 → direct-mapped cache
b = 5 → 32bytes per cache line

cache line 하나당 integer 8개

따라서 block size를 8*8로 하여 transpose 진행

```
// Case 1 : M = 32, N = 32
// 1 ROW = 32 integers --> 4 cache lines
// Cache can contain 8 rows
// Use Block Size of 8
if(M == 32 && N == 32){
    for(k = 0; k < N; k += 8){
        for(j = 0; j < M; j += 8){
            for(i = 0; i < 8; i++){
                x1 = A[k+i][j];
                x2 = A[k+i][j+1];
                x3 = A[k+i][j+2];
                x4 = A[k+i][j+3];
                x5 = A[k+i][j+4];
                x6 = A[k+i][j+5];
                x7 = A[k+i][j+6];
                x8 = A[k+i][j+7];
                B[j][k+i] = x1;
                B[j+1][k+i] = x2;
                B[j+2][k+i] = x3;
                B[j+3][k+i] = x4;
                B[j+4][k+i] = x5;
                B[j+5][k+i] = x6;
                B[j+6][k+i] = x7;
                B[j+7][k+i] = x8;
            }
        }
    }
}
```

2. 구현 방법 : trans.c – (2) 64 X 64

```
// Case 2 : M = 64, N = 64
// 1 Row = 64 integers --> 8 cache lines
// Cache can only contain 4 rows --> Conflict
// Use Block Size of 4
else if(M == 64 && N == 64){
    for(k = 0; k < N; k += 4){
        for(j = 0; j < M; j += 4){
            for(i = 0; i < 4; i++){
                x1 = A[k+i][j];
                x2 = A[k+i][j+1];
                x3 = A[k+i][j+2];
                x4 = A[k+i][j+3];
                B[j][k+i] = x1;
                B[j+1][k+i] = x2;
                B[j+2][k+i] = x3;
                B[j+3][k+i] = x4;
            }
        }
    }
}
```

64 X 64에서도 block size를 8로 할 경우

전체 A의 1개의 row가 8 cache lines를 차지한다.

이 경우 32sets만 담을 수 있는 cache는

4개의 row만 담을 수 있다.

B 기준으로 8개의 column으로 모두 담을 수 없으므로 conflict가 발생하게 된다.

따라서 block size를 4로 해서 진행한다.

2. 구현 방법 : trans.c – (3) 61 X 67

```
// Case 3 : M = 61, N = 67
// 4의 배수인 60 * 60 + 1 * 67 + 7 * 60
else if(M == 61 && N == 67){
    // 60 * 60
    for(k = 0; k < 60; k += 4){
        for(j = 0; j < 60; j += 4){
            for(i = 0; i < 4; i++){
                x1 = A[k+i][j];
                x2 = A[k+i][j+1];
                x3 = A[k+i][j+2];
                x4 = A[k+i][j+3];

                B[j][k+i] = x1;
                B[j+1][k+i] = x2;
                B[j+2][k+i] = x3;
                B[j+3][k+i] = x4;
            }
        }
    }
```

```
// 1 * 67
for(i = 0; i < 67; i++){
    x1 = A[i][60];
    B[60][i] = x1;
}
// 7 * 60
for(j = 0; j < 60; j += 4){
    for(i = 0; i < 7; i++){
        x1 = A[60+i][j];
        x2 = A[60+i][j+1];
        x3 = A[60+i][j+2];
        x4 = A[60+i][j+3];

        B[j][60+i] = x1;
        B[j+1][60+i] = x2;
        B[j+2][60+i] = x3;
        B[j+3][60+i] = x4;
    }
}
```

2)와 같이 block size 4로 하기 위해

61 x 67 = 60 x 60 + 1 x 67 + 7 x 60로 한다.

세부적인 구현 방식은 2)와 동일하다.

3. 어려웠던 점

- C의 struct 구조에 익숙하지 않아, cache를 나타내기 위한 자료구조를 구성하기 어려웠다.
- Least Recently Used에 기반한 Replacement를 구현하기 위한 counter의 적용 방법을 설계하기 어려웠다.
- Cache-friendly한 Transpose를 구현하기 위해 Cache의 구조에 대한 깊은 이해가 필요했다.
- Matrix 크기에 따라 block 사이즈를 다르게 하여 구현하였지만, 2 case에 대해서는 optimal하게 구현하지 못했다.

4. 새롭게 배운 점

- LRU replacement method가 구현되는 방식을 알 수 있었다.

- matrix의 transpose를 구현하는 데에 있어서, row와 column을 iterate하는 방식에 따라 cache miss로 인해 성능의 차이가 크게 발생할 수 있음을 알 수 있었다.

- matrix의 size별로 주어진 cache의 구조에 따라 optimal한 block size가 달라질 수 있다.