Project 2: Understanding Cache Memories

519021910366, Dou Yiming, douyiming@sjtu.edu.cn

June 5, 2021

1 Introduction

This project aims at making us understand cache memories better. The whole project can be divided into two parts.

Firstly, we are required to simulate a cache including sets, ways, tags and valid bits. It also covers cache replacement algorithm LRU that we have learned in operating system class. I complete this part with reference to slides of the course and CSAPP book.

Secondly, on the basis of the cache simulator, we are required to implement an optimization. When transposing the matrix, the cache may not play to its full capability if without enough design. Therefore, we need to improve the cache hit rate by blocking the matrix and transfer each block with the help of cache in order to speed up.

2 Experiments

2.1 Part A

2.1.1 Analysis

The cache we are required to simulate contains 2^s sets, E ways and 2^b block offsets, whose structure can be shown in Fig. 1.

In the process of a single entry, we begin with traversing each set, checking whether it's valid or not and whether the tag is the same as the data we want. If a qualified set is found, we define this as a cache hit. Otherwise, if the cache is full, we define this situation as eviction, in which we should swap a new element with the existing element in the cache.

The next problem is how we choose the place we want to evict. In this project, we use the Least-Recently-Used (LRU) strategy, meaning that we need to seek the data that we have not used for the longest time. Therefore, we attach a time stamp for each set, each time the set is visited, we update the time stamp of it to the current time, and when eviction happens, we choose the set whose time stamp is the smallest.

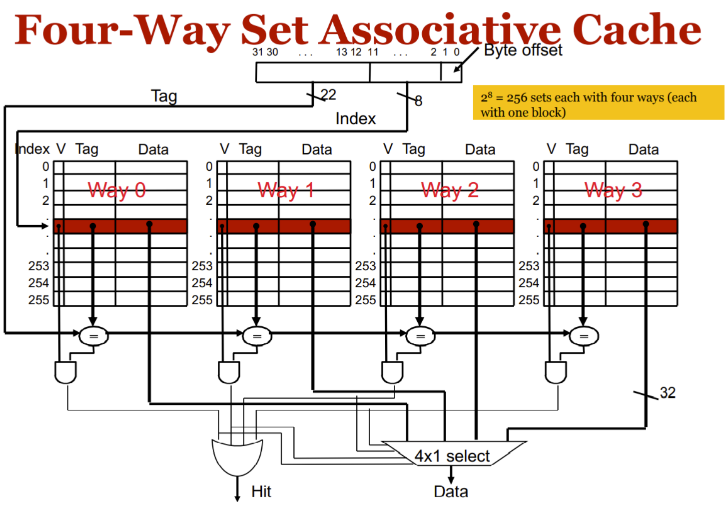


Figure 1: Cache Structure

2.1.2 Code

```
/*
 * Name: DouYiming
 * ID: 519021910366
 */
#include <getopt.h>
#include <stdlib.h>
#include <stdio.h>
#include "cachelab.h"

const int m = 64;

typedef struct Arg
{
    int s; // Number of set index bits
    int E; // number of lines per set
    int b; // Number of block bits
    FILE *f; // valgrind trace to replay
} Arg;

typedef struct summary
{
    int hits;
    int misses;
    int evictions;
} summary;

typedef struct Line
{
    int valid;
    int tag;
    int time_stamp;
```

```c
} Line;

typedef struct Set
{
    Line* lines;
} Set;

Arg arg_parser(int argc, char ** argv)
{
    Arg arg;
    if(getopt(argc, argv, "s:E:b:t:") == 's')
        arg.s = atoi(optarg);
    if(getopt(argc, argv, "s:E:b:t:") == 'E')
        arg.E = atoi(optarg);
    if(getopt(argc, argv, "s:E:b:t:") == 'b')
        arg.b = atoi(optarg);
    if(getopt(argc, argv, "s:E:b:t:") == 't')
        arg.f = fopen(optarg, "r");
    return arg;
}

Set* init_cache(Arg arg)
{
    int set_num = 1 << arg.s;
    Set* caches=(Set*)malloc(set_num * sizeof(Set));
    for(int i = 0;i < set_num; ++i)
    {
        caches[i].lines = (Line*)malloc(arg.E * sizeof(Line));
        for(int j = 0; j < arg.E; ++j)
        {
            caches[i].lines[j].valid = 0;
            caches[i].lines[j].time_stamp = 0;
        }
    }
    return caches;
}

summary simulator(Arg arg, Set* caches)
{
    // information to display
    summary ans;
    ans.hits = 0;
    ans.misses = 0;
    ans.evictions = 0;
    // compute tag bits
    int t = m - arg.b - arg.s;
    // simulator time
    int cur_time = 0;
    // information of each instruction
    char op;
    long addr;
    int len;
    int cycle_time;

    // read from file
    while(!feof(arg.f))
    {
```

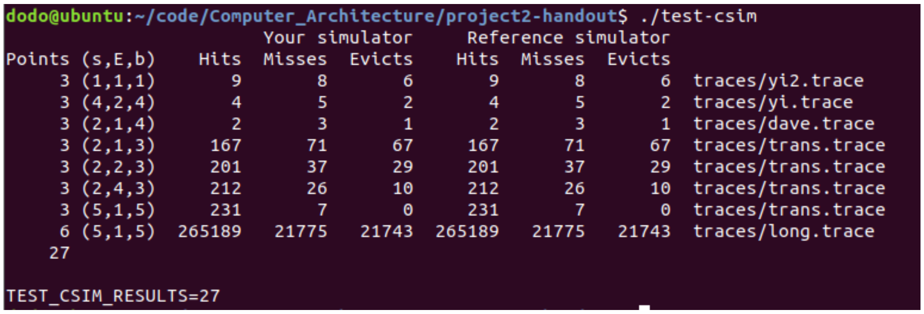
```
        if (fscanf(arg.f, " %c %lx,%d", &op, &addr, &len) != 3)
            continue;

        if (op == 'L' || op == 'S')
            cycle_time = 1;
        else if (op == 'M')
            cycle_time = 2;
        else
            continue;
        while(cycle_time--)
        {
            ++cur_time;
            int set_num = (addr >> arg.b) & ((1 << arg.s) - 1);
            int tag = (addr >> (arg.s + arg.b)) & ((1 << t) - 1);
            int is_hit = 0;
            int invalid_line = -1;
            int LRU_line = -1;
            int min_time = 0x7fffffff;
            // search each way
            for(int i = 0;i < arg.E; ++i)
            {
                Line* line = &caches[set_num].lines[i];
                if(line->valid && line->tag == tag) // hit
                {
                    is_hit = 1;
                    line->time_stamp = cur_time;
                    break;
                }
                else if(!line->valid)
                {
                    invalid_line = i;
                    break;
                }
                else if(line->valid && line->time_stamp<min_time)
                {
                    min_time = line->time_stamp;
                    LRU_line = i;
                }
            }
            if(is_hit)//hit
                ++ans.hits;
            else
            {
                ++ans.misses;
                if(invalid_line != -1)//miss
                {
                    Line*l=&caches[set_num].lines[invalid_line];
                    l->tag = tag;
                    l->time_stamp = cur_time;
                    l->valid = 1;
                }
                else//evict
                {
                    ++ans.evictions;
                    Line*l=&caches[set_num].lines[LRU_line];
                    l->tag = tag;
                    l->time_stamp = cur_time;
```

```c
                }
            }
        }
    }
    return ans;
}

void terminate(Arg arg, Set* caches)
{
    fclose(arg.f);
    for (int i = 0; i < (1 << arg.s); ++i)
        free(caches[i].lines);
    free(caches);
}

int main(int argc, char **argv)
{
    // arg parsing
    Arg arg = arg_parser(argc, argv);
    // cache initialization
    Set* caches = init_cache(arg);
    // simulation
    summary s = simulator(arg, caches);
    // ending
    terminate(arg,caches);
    // print ans
    printSummary(s.hits, s.misses, s.evictions);
    return 0;
}
```



```
dodo@ubuntu:~/code/Computer_Architecture/project2-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
```

Figure 2: The result of part A

2.1.3 Evaluation

The result of part A is shown in Fig. 2.

2.2 Part B

2.2.1 Analysis

In this part, we need to optimize the process of transposing matrix of different sizes.

The basic idea is that we should use the elements in the cache as much as possible for each time. Therefore, simply traversing the rows and columns of a large matrix may not be a good idea due to large amounts of evictions. In order to make full use of the cache, we separate the matrix into blocks and move a block into the cache each time, ensuring that most elements in cache are used.

Furthermore, our method to separate the matrix should vary when facing different sizes of matrix by calculating the address of elements in the matrix. Therefore, we respectively implement a optimization method for each matrix transposing situation.

2.2.2 Code

```c
/*
 * Name: Dou Yi-ming
 * ID: 519021910366
 * trans.c - Matrix transpose B = A^T
 *
 * Each transpose function must have a prototype of the form:
 * void trans(int M, int N, int A[N][M], int B[M][N]);
 *
 * A transpose function is evaluated by counting the number of
    misses
 * on a 1KB direct mapped cache with a block size of 32 bytes.
 */
#include <stdio.h>
#include "cachelab.h"
```

```c

int is_transpose(int M, int N, int A[N][M], int B[M][N]);

/*
 * transpose_submit - This is the solution transpose function that you
 *     will be graded on for Part B of the assignment. Do not change
 *     the description string "Transpose submission", as the driver
 *     searches for that string to identify the transpose function to
 *     be graded.
 */
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    // local variables
    int ii, jj, i, j;
    int tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    if (M == 32 && N == 32)
    {
        // 8 * 8 blocks
        for(ii = 0;ii < 4;++ii)
            for(jj = 0;jj < 4;++jj)
                for(i = 0;i < 8;++i)
                {
                    for(j = 0;j < 8;++j)
                        if(i != j)
                            B[8 * jj + j][8 * ii + i] = A[8 * ii + i][8 * jj + j];
                    B[8 * jj + i][8 * ii + i] = A[8 * ii + i][8 * jj + i];
                }
    }
    else if (M == 64 && N == 64)
    {
        for(ii = 0;ii < 8;++ii)
            for(jj = 0;jj < 8;++jj)
            {
                for(i = 0;i < 4;++i)
                {
                    tmp0 = A[ii * 8 + i][jj * 8 + 0];
                    tmp1 = A[ii * 8 + i][jj * 8 + 1];
                    tmp2 = A[ii * 8 + i][jj * 8 + 2];
                    tmp3 = A[ii * 8 + i][jj * 8 + 3];
                    tmp4 = A[ii * 8 + i][jj * 8 + 4];
                    tmp5 = A[ii * 8 + i][jj * 8 + 5];
                    tmp6 = A[ii * 8 + i][jj * 8 + 6];
                    tmp7 = A[ii * 8 + i][jj * 8 + 7];

                    B[jj * 8 + 0][ii * 8 + i] = tmp0;
                    B[jj * 8 + 1][ii * 8 + i] = tmp1;
                    B[jj * 8 + 2][ii * 8 + i] = tmp2;
                    B[jj * 8 + 3][ii * 8 + i] = tmp3;

                    B[jj * 8 + 0][ii * 8 + i + 4] = tmp4;
                    B[jj * 8 + 1][ii * 8 + i + 4] = tmp5;
```

```c
                    B[jj * 8 + 2][ii * 8 + i + 4] = tmp6;
                    B[jj * 8 + 3][ii * 8 + i + 4] = tmp7;
                }
                for(j = 0; j < 4; ++j)
                {
                    tmp0 = B[jj * 8 + j][ii * 8 + 4];
                    tmp1 = B[jj * 8 + j][ii * 8 + 5];
                    tmp2 = B[jj * 8 + j][ii * 8 + 6];
                    tmp3 = B[jj * 8 + j][ii * 8 + 7];

                    tmp4 = A[ii * 8 + 4][jj * 8 + j];
                    tmp5 = A[ii * 8 + 5][jj * 8 + j];
                    tmp6 = A[ii * 8 + 6][jj * 8 + j];
                    tmp7 = A[ii * 8 + 7][jj * 8 + j];

                    B[jj * 8 + j][ii * 8 + 4] = tmp4;
                    B[jj * 8 + j][ii * 8 + 5] = tmp5;
                    B[jj * 8 + j][ii * 8 + 6] = tmp6;
                    B[jj * 8 + j][ii * 8 + 7] = tmp7;

                    tmp4 = A[ii * 8 + 4][jj * 8 + j + 4];
                    tmp5 = A[ii * 8 + 5][jj * 8 + j + 4];
                    tmp6 = A[ii * 8 + 6][jj * 8 + j + 4];
                    tmp7 = A[ii * 8 + 7][jj * 8 + j + 4];

                    B[jj * 8 + j + 4][ii * 8 + 0] = tmp0;
                    B[jj * 8 + j + 4][ii * 8 + 1] = tmp1;
                    B[jj * 8 + j + 4][ii * 8 + 2] = tmp2;
                    B[jj * 8 + j + 4][ii * 8 + 3] = tmp3;
                    B[jj * 8 + j + 4][ii * 8 + 4] = tmp4;
                    B[jj * 8 + j + 4][ii * 8 + 5] = tmp5;
                    B[jj * 8 + j + 4][ii * 8 + 6] = tmp6;
                    B[jj * 8 + j + 4][ii * 8 + 7] = tmp7;
                }
            }
    }
    else if (M == 61 && N == 67)
    {
        for(ii = 0; ii < 5; ++ii)
            for(jj = 0; jj < 4; ++jj)
                for(i = 0; i < 16 && ii * 16 + i < N; ++i)
                    for(j = 0; j < 16 && jj * 16 + j < M; ++j)
                        B[16 * jj + j][16 * ii + i] = A[16 * ii + i][16 * jj + j];
    }
    else
    {
        int tmp;
        for (i = 0; i < N; i++)
            for (j = 0; j < M; j++)
            {
                tmp = A[i][j];
                B[j][i] = tmp;
            }
    }

}
```

```c

/*
 * You can define additional transpose functions below. We've defined
 * a simple one below to help you get started.
 */

/*
 * trans - A simple baseline transpose function, not optimized for the cache.
 */
char trans_desc[] = "Simple row-wise scan transpose";
void trans(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, tmp;

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            tmp = A[i][j];
            B[j][i] = tmp;
        }
    }

}

/*
 * registerFunctions - This function registers your transpose
 *     functions with the driver.  At runtime, the driver will
 *     evaluate each of the registered functions and summarize their
 *     performance. This is a handy way to experiment with different
 *     transpose strategies.
 */
void registerFunctions()
{
    /* Register your solution function */
    registerTransFunction(transpose_submit, transpose_submit_desc);

    /* Register any additional transpose functions */
    registerTransFunction(trans, trans_desc);

}

/*
 * is_transpose - This helper function checks if B is the transpose of
 *     A. You can check the correctness of your transpose by calling
 *     it before returning from the transpose function.
 */
int is_transpose(int M, int N, int A[N][M], int B[M][N])
{
    int i, j;

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; ++j) {
```

```
                if (A[i][j] != B[j][i]) {
                    return 0;
                }
            }
        }
    return 1;
}
```

```
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        1187
Trans perf 61x67          10.0        10        1992
          Total points    53.0        53
```

Figure 3: Result of part B

2.2.3 Evaluation

The result of the test is shown in Fig. 3

3 Conclusion

3.1 Problems

1. The name of each part of the cache is different from the name we use in class and slides, which cause lots of confusion. With reference to CSAPP book, we finally solve this problem.

2. The optimization of the matrix transposing problem is far from our intuition, which is simply traversing the rows and columns. After referring to lots of materials, we eventually understand the separation method.

3.2 Achievements

1. My results are excellent. All of the requirements have been satisfied and i get full score in each test.

2. When designing the cache simulator, i paid a lot of attention to the structure of the code, and optimize the structure to improve readability and conciseness for a long time. Finally, i reduced the length of main function to 6 lines and i am very proud of it.

3. I figured out and successfully implemented the optimization method of transposing the matrix, which is very complicated.