Project 2: Understanding Cache Memories

519021911174, Haoyuan Tian, thy0210@sjtu.edu.cn

June 3, 2021

1 Introduction

The objective of the project is to help us develop an understanding of how the cache memory works and how it affects the performance of our programs. The first part of the project requires writing a simple cache simulator. The second part requires optimizing the performance of a matrix transpose program, where the main target is to minimize the number of cache misses.

2 Experiments

2.1 Part A

2.1.1 Analysis

In **Part A**, we are required to write a C program to simulate the mechanism of the cache memory. The program takes input from the *.trace* file and the three parameters, respectively S, E and B are read from the *valgrind* command. The replacement strategy is LRU, and the program output the number of hits, misses, and evictions.
The main difficulties here are listed below:

1. Analyze the input of instructions properly.
2. Arrange the miss and hit situation properly on the basis.

2.1.2 Functions in detail

- ***getopt()***

Prototype:

```c
int getopt(int argc, char * const argv[], const char * optstring);
```

As shown in the complete code below, the function is used to analyze command line arguments.

- ***readCache()***

The function is used to imitate the behaviour of the cache memory. According to the reference, we divide the situations that the cache may encounter into 5 cases, each corresponding to a different number of hits, misses and evictions.

2.1.3 Code

```
// Haoyuan Tian, 519021911174

#include "cachelab.h"
#include <getopt.h>
#include <unistd.h>
#include <math.h>
#include <limits.h>
#include <memory.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>
#include <math.h>

typedef struct
{
    int valid;
    long address;
    int time;
} Cache;

Cache *cache;

int S, E, B;
int hits = 0;
int misses = 0;
int evictions = 0;

int readCache(Cache *cache, char type, long address)
{
    int flag = 0;
    int line = (int)((address / B) % S);
    int pos = 0;
    int minTime = INT_MAX;
    int current = -1;
    int exist = 0;
    int insert = 0;
    for (int i = 0; i < E; i++)
    {
        Cache *element = (cache + line * E + i);
        if (insert == 0 && flag == 0)
        {
            if (!element->valid)
            {
                element->valid = 1;
                element->address = address;
                insert = 1;
                exist = 0;
                flag = 2;
                pos = i;
                if (type == 'M')
                {
                    flag = 3;
```

```
                }
            }
            else
            {
                if ((element->address / B) == (address / B))
                {
                    exist = 1;
                    pos = i;
                    flag = 1;
                }
                if (element->time < minTime)
                {
                    minTime = element->time;
                    if (exist == 0)
                    {
                        pos = i;
                    }
                }
            }
        }
        if (element->time > current)
            current = element->time;
    }
    Cache *element = (cache + line * E + pos);
    if (insert == 0 && exist == 0)
    {
        element->address = address;
        flag = 4;
        if (type == 'M')
        {
            flag = 5;
        }
    }
    element->time = current + 1;
    return flag;
}

void printHelp()
{
printf("Usage: ./csim [-hv] -s <s> -E <E> -b <b> -t <tracefile>\n");
}

int main(int argc, char *argv[])
{
    opterr = 0;
    int verbose = 0;
    int para;
    char *name = NULL;
    while ((para = getopt(argc, argv, "s:E:b:t:hv")) != -1)
    {
        switch (para)
        {
        case 's':
            S = (int)pow(2, atoi(optarg));
            break;
```

```
            case 'E':
                E = atoi(optarg);
                break;
            case 'b':
                B = (int)pow(2, atoi(optarg));
                break;
            case 't':
                name = optarg;
                break;
            case 'v':
                verbose = 1;
                break;
            case 'h':
                printHelp();
                break;
            }
        }

        FILE *file = fopen(name, "r");

        if (file == NULL)
        {
            printf("NO SUCH FILE!\n");
            return -1;
        }

        cache = (Cache *)malloc(S * E * sizeof(Cache));
        if (cache == NULL)
        {
            printf("MEMORY ALLOCATION FAILED!\n");
            return -1;
        }

        memset(cache, 0, S * E * sizeof(Cache));
        char type;
        int size;
        long address;
        while (!feof(file))
        {
            int tmp = fscanf(file, " %c %lx,%x", &type, &address, &size);
            if (tmp != 3)
                continue;
            if (type != 'I')
            {
                int ret = readCache(cache, type, address);
                char *tmp1 = NULL;
                switch (ret)
                {
                case 1:
                    hits++;
                    tmp1 = "hit";
                    if (type == 'M')
                        hits++;
                    break;
                case 2:
```

```
                misses++;
                tmp1 = "miss";
                break;

            case 3:
                misses++;
                hits++;
                tmp1 = "miss";
                break;
            case 4:
                misses++;
                evictions++;
                tmp1 = "miss";
                break;
            case 5:
            default:
                misses++;
                evictions++;
                hits++;
                tmp1 = "miss";
                break;
            }
            if (verbose)
            {
                printf("%c %lx,%x %s", type, address, size, tmp1);
                if (ret == 4)
                {
                    printf(" eviction");
                }
                else if (ret == 3)
                {
                    printf(" hit");
                }
                else if (ret == 5)
                {
                    printf(" eviction hit");
                }
                else if (ret == 1 && type == 'M')
                {
                    printf(" hit");
                }

                printf("\n");
            }
        }
    }
    free(cache);
    printSummary(hits, misses, evictions);
}
```

2.1.4 Evaluation

```
thy@thy-virtual-machine:~/Desktop/arch/project2-handout$ make
# Generate a handin tar file each time you compile
tar -cvf thy-handin.tar  csim.c trans.c
csim.c
trans.c
```

Figure 1: Use *make* to compile the project

In the project, the warnings will be considered errors when compiling the project. **Figure 1** shows both **Part A** and **Part B** are free of warnings.

```
thy@thy-virtual-machine:~/Desktop/arch/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: Use *./test-csim* to test the result

Figure 2 shows that the simulator we realized produces exactly the same results as the reference simulator. It reaches the full mark which is 27 here.

```
thy@thy-virtual-machine:~/Desktop/arch/project2-handout$ ./csim -v -s 1 -E 1 -b 1 -t ./tra
ces/yi2.trace
L 0,1 miss
L 1,1 hit
L 2,1 miss
L 3,1 hit
S 4,1 miss eviction
L 5,1 hit
S 6,1 miss eviction
L 7,1 hit
S 8,1 miss eviction
L 9,1 hit
S a,1 miss eviction
L b,1 hit
S c,1 miss eviction
L d,1 hit
S e,1 miss eviction
M f,1 hit hit
hits:9 misses:8 evictions:6
```

Figure 3: Test the *verbose* output

The verbose output displays the hits, misses, and evictions that occur as a result of each memory access. The feature is not required but extremely helpful when debugging.

2.2 Part B

2.2.1 Analysis

In **Part B**, we are required to write a matrix-transpose function that causes as few cache misses as possible. The most crucial part here is understanding how cache handles the elements inside a matrix. In simple term, when a cache miss happens, the particular element will be stored into cache together with the surrounding elements on the same row. Due to the given specifications of cache and the different matrix sizes, we have to make special arrangement for each of the three conditions.

- M=32, N=32
  According to the cache size, each line of the cache can store 8 integers, which means that the cache will be filled with every 8 rows of the matrix A. Therefore, we can divide B into 8×8 matrices. Arrange the elements on the diagonal line will further reduce the misses.

- M=64, N=64
  The idea used here will be discussed later in **3.1 Problems**.

- M=61, N=67
  Neither M nor N is 2 to the power. However, the criteria is not strict here. Dividing A into 16×16 matrices is a feasible scheme.

2.2.2 Code

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, n, m;
    int tmp, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;
    if (M == 32)
    {
        for (n = 0; n < N; n += 8)
        {
            for (m = 0; m < M; m += 8)
            {
                for (i = n; i < n + 8; i++)
                {
                    for (j = m; j < m + 8; j++)
                    {
                        if (i != j)
                        {
                            B[j][i] = A[i][j];
                        }
                        else
                        {
                            tmp = A[i][j];
                        }
                    }
                    if (m == n)
                    {
                        B[i][i] = tmp;
                    }
                }
            }
        }
    }
    else if (M == 64)
```

```
    {
        for (m = 0; m < 64; m += 8)
        {
            for (n = 0; n < 64; n += 8)
            {
                for (i = n; i < n + 4; ++i)
                {
                    tmp = A[i][m];
                    tmp1 = A[i][m + 1];
                    tmp2 = A[i][m + 2];
                    tmp3 = A[i][m + 3];
                    tmp4 = A[i][m + 4];
                    tmp5 = A[i][m + 5];
                    tmp6 = A[i][m + 6];
                    tmp7 = A[i][m + 7];

                    B[m][i] = tmp;
                    B[m][i + 4] = tmp4;
                    B[m + 1][i] = tmp1;
                    B[m + 1][i + 4] = tmp5;
                    B[m + 2][i] = tmp2;
                    B[m + 2][i + 4] = tmp6;
                    B[m + 3][i] = tmp3;
                    B[m + 3][i + 4] = tmp7;
                }
                for (i = m; i < m + 4; ++i)
                {
                    tmp = B[i][n + 4];
                    tmp1 = B[i][n + 5];
                    tmp2 = B[i][n + 6];
                    tmp3 = B[i][n + 7];
                    tmp4 = A[n + 4][i];
                    tmp5 = A[n + 5][i];
                    tmp6 = A[n + 6][i];
                    tmp7 = A[n + 7][i];


                    B[i][n + 4] = tmp4;
                    B[i][n + 5] = tmp5;
                    B[i][n + 6] = tmp6;
                    B[i][n + 7] = tmp7;
                    B[i + 4][n] = tmp;
                    B[i + 4][n + 1] = tmp1;
                    B[i + 4][n + 2] = tmp2;
                    B[i + 4][n + 3] = tmp3;

                }
                for (i = m + 4; i < m + 8; ++i)
                {
                    tmp = A[n + 4][i];
                    tmp1 = A[n + 5][i];
                    tmp2 = A[n + 6][i];
                    tmp3 = A[n + 7][i];

                    B[i][n + 4] = tmp;
```

```
                    B[i][n + 5] = tmp1;
                    B[i][n + 6] = tmp2;
                    B[i][n + 7] = tmp3;
                }
            }
        }
    }
    else
    {
        for (n = 0; n < N; n += 16)
        {
            for (m = 0; m < M; m += 16)
            {
                for (i = n; (i < n + 16) && (i < N); i++)
                {
                    for (j = m; (j < m + 16) && (j < M); j++)
                    {
                        if (i != j)
                        {
                            B[j][i] = A[i][j];
                        }
                        else
                        {
                            tmp = A[i][j];
                        }
                    }
                    if (m == n)
                    {
                        B[i][i] = tmp;
                    }
                }
            }
        }
    }
}
```

2.2.3 Evaluation

Since the correctness during the compile stage has been proved in **Part A**, we won't bother to show it again here.

```
thy@thy-virtual-machine:~/Desktop/arch/project2-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
```

Figure 4: M=32, N=32

```
thy@thy-virtual-machine:~/Desktop/arch/project2-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:9026, misses:1219, evictions:1187

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=1219

TEST_TRANS_RESULTS=1:1219
```

Figure 5: M=64, N=64

```
thy@thy-virtual-machine:~/Desktop/arch/project2-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:6193, misses:1989, evictions:1957

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=1989

TEST_TRANS_RESULTS=1:1989
```

Figure 6: M=61, N=67

1. M=32, N=32: 287 misses in total.
2. M=64, N=64: 1219 misses in total.
3. M=61, N=67: 1989 misses in total.

All reach the full mark!

3 Conclusion

3.1 Problems

3.1.1 Problems in Part A

The first obstacle comes from the unfamiliarity with the function *getopt()*. After reading some materials on the Internet, I manage to apply it properly.
The second obstacle is the arrangement of different cache situations. Once I figured out the logic of the cache and the impact of the instructions from the *.trace* file, all left to be done is debugging. Using the reference simulator *csim-ref* and verbose feature wisely helps a lot.

3.1.2 Problems in Part B

This part is particularly focused on the $M = 64, N = 64$ situation.
The cache will be filled with every 4 rows of the matrix A, so the initial idea was to divide B into 4×4 matrices. The total number of misses using such method is 1699, which is not ideal.
Next, I tried the division plan of first 8×8, then 4×4. By arranging the sequence properly, the number of misses went down to 1451. Effective, but still not satisfying enough.
The final idea was inspired by the reading material provided and blogs on the Internet. The previous ideas only focused on dividing A and B, but ignored that we could make changes to matrix B. The figure below illustrated the idea in a brief way.

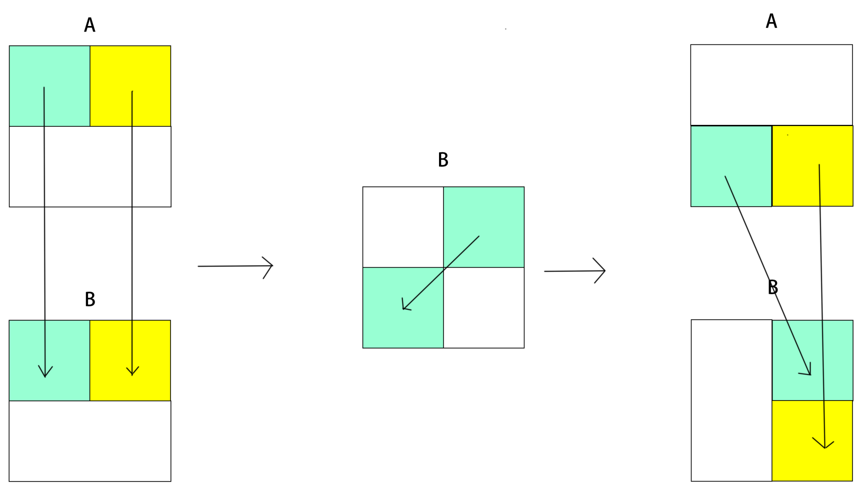

Figure 7: The final idea

The yellow block of A doesn't go to the right place in the first round of transposition, and will be dealt with later. In this way, the number of misses was eventually cut down to 1219.

3.2 Achievements

In **Part A**, I managed to realize a cache simulator in only 212 lines. The handling of the LRU replacement strategy might seem confusing but it worked just well. The situations were divided precisely and tackled successfully. I also paid effort to improve the robustness of the program by rejecting the wrong input and avoiding memory leaks.
In **Part B**, I strictly followed the restriction about the number of local variables. In the mean time, the variables defined locally were used wisely and none of them was redundant. The loop structure is clear and easy to understand.

3.3 Feelings

The project puts emphasis on cache and cache alone. It provides a golden opportunity to put knowledge learnt in the course into practice, and therefore deepen our understanding of the mechanism of cache. Without exaggeration, I can say that I've realized a significant optimization and produced a marvellous result. The process of trial and error might be miserable, especially when the result of **Part B** didn't live up to my expectation. But finally accomplishing the full mark with my effort brought more delight. Cache is a very interesting and important part of the computer architecture, and I hope that I will have the chance to explore more about it later.

Great thanks to teacher Shen and the teaching assistants for your patience and guidance.