# Project 2: Understanding Cache Memories

Jingwei Xi, 517030910116, jingweixi@sjtu.edu.cn

June 5, 2019

## 1 Introduction

[In this section you should briefly introduce the task in your own words, and what you've done in this project. A simple copy from project1.pdf is not permitted.]

In this lab, I write two parts for programs. In the first part, I write a program to simulate the behavior of a cache memory. In the program, I use array as the data structure for cache. In the second part, I write a program to optimize cache performance for matrix transpose function which use amazing idea.

## 2 Experiments

[This is the main part of your report. It includes three parts and in each part, you need to write concretely, logically but not in full details.]

### 2.1 Part A

#### 2.1.1 Analysis

[In this part, you should give an overall analysis for the task, like difficult point, core technique and so on.]

The task for part A is to simulates the behavior of a cache with arbitrary size and associativity on a valgrind trace file. To complete this task, I use array to simulate the cache.

In each element of array, I use three variables. First is valid bit, which indicate whether block is in cache. Second is tag, which is used to find the tag. The third one is the usedTime, which record the time that block last be used.

I use the LRU (least-recently used) replacement policy when choosing which cache line to evict. If there is empty line in the set, I will put the block into empty line. If not, I will evict the line with least usedTime number, which indicate it has not been used for a long time.

### 2.1.2 Code

[In this part, you should place your code and make it readable in Microsoft Word, please. Writing necessary comments for codes is a good habit.]

csim.c

```c
//517030910116     Jingwei Xi
//email: jingweixi@sjtu.edu.cn
//This is the program for simulating the behavior of a cache
#include "cachelab.h"
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <memory.h>

typedef struct{
    int valid;
    long unsigned int tag;
    int timeRef;
}line;   // The line in cache array

typedef struct{
    int helpFlag;           //-h
    int verboseFlag;        //-v
    int setBit;             //-s
    int linePerSet;         //-e
    int blockBit;           //-b
    char *fileName;         //-t
}argument;     //The parameters of instruction

line *cache;
argument mainArg;

int setCount;
int blockSize;
int cacheSize;
int hit;
int miss;
int eviction;
int timeClock = 0;

void printHelp(){
    printf("Usage: ./csim-wrc [-hv] -s <s> -E <E> -b <b>
```

```
        -t <tracefile>\n");
    printf("-h: Optional help flag that prints usage info\n");
    printf("-v: Optional verbose flag that displays trace info\n");
    printf("-s <s>: Number of set index bits (S = 2^s is the number of sets)\n");
    printf("-E <E>: Associativity (number of lines per set)\n");
    printf("-b <b>: Number of block bits (B = 2^b is the block size)\n");
    printf("-t <tracefile>: Name of the valgrind trace to replay\n");
}

void initMainArg(){
    mainArg.helpFlag = 0;
    mainArg.verboseFlag = 0;
    mainArg.setBit = 0;
    mainArg.linePerSet = 0;
    mainArg.blockBit = 0;
    mainArg.fileName = NULL;
}

void cacheAccess(char type, long unsigned int addr, int size){
    int hitFlag = 0, hitId = -1, emptyLine = -1, minTime = INT_MAX, evicId;
    int setIndex = 0;
    int dataTag;
    int i;

    //Address: |tag|setIndex|block_offset|
    setIndex = (addr / (blockSize)) % (setCount);
    dataTag = addr / (blockSize * setCount);

    for(i = setIndex * mainArg.linePerSet; i < (setIndex + 1) * mainArg.linePerSet; i++){
        //Hit
        if(cache[i].tag == dataTag){
            hitFlag = 1;
            hitId = i;
            break;
        }
        //Record the empty line id
        if(cache[i].valid == 0 && emptyLine == -1){
```

```
            emptyLine = i;
        }
        //Find the block line with the least timeRef number
        if(cache[i].timeRef < minTime){
            minTime = cache[i].timeRef;
            evicId = i;
        }
    }

    if (mainArg.verboseFlag){
        printf("%c %lx,%x ",type,addr,size);
    }
    if(hitFlag == 1){     //Hit
        cache[hitId].timeRef = timeClock;
        hit++;
        if(type == 'M'){
            hit++;
        }
        if(mainArg.verboseFlag){
            if(type == 'S' || type == 'L'){
                printf("hit\n");
            }
            else{
                printf("hit hit\n");
            }
        }

    }
    else{
        if(emptyLine != -1){//Miss but there is empty line
            cache[emptyLine].valid = 1;
            cache[emptyLine].tag = dataTag;
            cache[emptyLine].timeRef = timeClock;
            miss++;
            if(type == 'M'){
                hit++;
            }
            if(mainArg.verboseFlag){
                if(type == 'S' || type == 'L'){
                    printf("miss\n");
                }
                else{
                    printf("miss hit\n");
                }
```

```c
            }
        }
        else{    //Miss and no empty line, need to evict
            cache[evicId].valid = 1;
            cache[evicId].tag = dataTag;
            cache[evicId].timeRef = timeClock;
            miss++;
            eviction++;
            if(type == 'M'){
                hit++;
            }
            if(mainArg.verboseFlag){
                if(type == 'S' || type == 'L'){
                    printf("miss evition\n");
                }
                else{
                    printf("miss eviction hit\n");
                }
            }
        }
    }
}

int main(int argc, char* argv[])
{
    int opt;
    int i;
    char type;
    int size;
    long unsigned int addr;

    initMainArg();//init each variable in argument struct

    opt = getopt(argc, argv, "s:E:b:t:hv");
    if(opt == -1){      //Invalid arguments
        printHelp();
        return -1;
    }
    while(opt != -1) {
        switch(opt){
            case 'v':
                mainArg.verboseFlag = 1; /* true */
                break;
            case 's':
                mainArg.setBit = atoi(optarg);
                break;
```

```c
            case 'E':
                mainArg.linePerSet = atoi(optarg);
                break;
            case 'b':
                mainArg.blockBit = atoi(optarg);
                break;
            case 't':
                mainArg.fileName = optarg;
                break;
            default:
                printHelp();
                break;
        }
        opt = getopt(argc, argv, "s:E:b:t:hv");
    }

    setCount = 1 << (mainArg.setBit);
    blockSize = 1 << (mainArg.blockBit);

    FILE *file = fopen(mainArg.fileName, "r");
    if (file == NULL){    //Error: File not found
        printf("File not found.");
        return -1;
    }

    cache = (line *) malloc(setCount * mainArg.linePerSet * sizeof(line));
        if (cache == NULL){    //Error: Cache space allocated failed
            printf("Fail to allocate cache.");
            return -1;
        }

    cacheSize = setCount * mainArg.linePerSet;
    //Initialize the cache
    for (i = 0; i < cacheSize; i++) {
        cache[i].valid = 0;
        cache[i].timeRef = 0;
        cache[i].tag = -1;
    }

    while (!feof(file)){
        int tmp = fscanf(file, " %c %lx,%x", &type, &addr, &size);
        if (tmp != 3) continue;
        if (type == 'I') continue;
```

```
        cacheAccess(type, addr, size);
        timeClock++;
    }

    free(cache);    //Free the cache space malloced
    cache = NULL;
    printSummary(hit, miss, eviction);
    return 0;
}
```

### 2.1.3 Evaluation

[In this part, you should place the figures of experiments for your codes, prove the correctness and validate the performance with your own words for each figure's explanation.]

## 2.2 Part B

### 2.2.1 Analysis

[In this part, you should give an overall analysis for the task, like difficult point, core technique and so on.]

The task of part B is to write a transpose function that causes as few cache misses as possible.

For 32*32 matrix, each element in matrix with int type use 4 bytes. As our block size 32 bytes, we have 8 elements in one block. For each line in matrix, it can be placed in 4 blocks and the cache will save 8 lines of matrix at one time.

The distribution of matrix is showed in graph1 below

So we deal with 8*8 matrix each time because all matrix element are in the cache which will cause less miss.

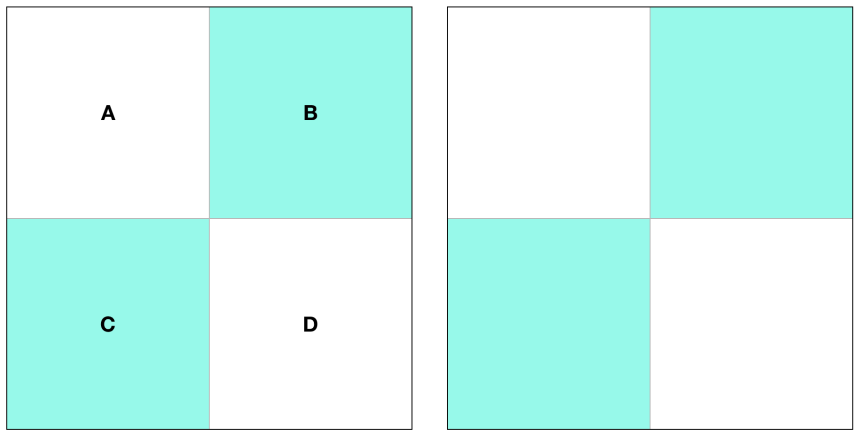
| | | | |
|---|---|---|---|
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 |
| 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | 3 |
| … | | | |

And notice for elements like A[i][i], the lines we'll use in matrix A and matrix B are mapped to the same cache line, which will generate unnecessary conflict misses. So we will use a temporary variant to avoid these misses.

For 64*64 matrix, the statement is different from before. For each line in matrix, it can be placed in 8 blocks and the cache will save 4 lines of matrix at one time. So only 4*4 matrix can be placed in cache at one time. If we transpose 4*4 matrix at on time, it can have little miss but it will waste a lot of space in cache line. The optimization will use 8*8 matrix and fully use of cache line and have little miss.

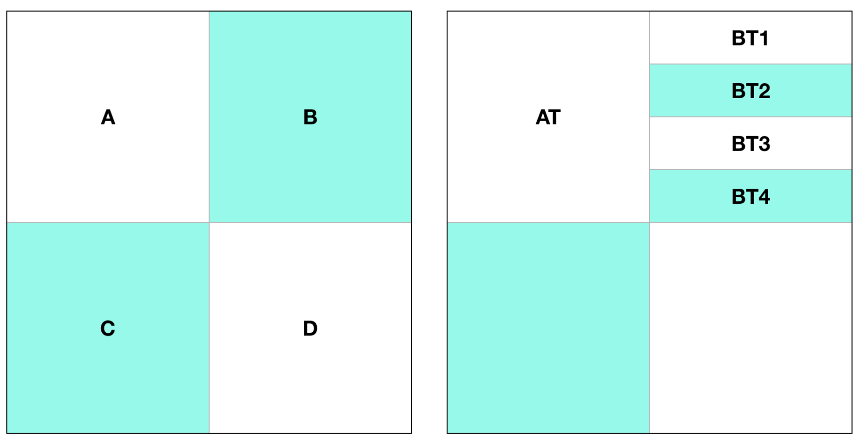
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| 0 | 1 | 2 | … | | | | |

Firstly, there is matrix A and matrix B, and both of them are 8x8. I divide A and B into four 4x4 matrices: A, B, C, D. At this time, the matrix B is empty.



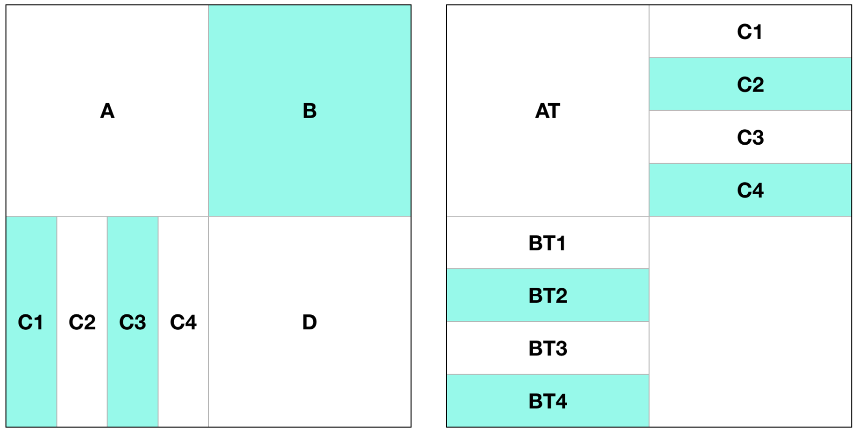

Matrix A Matrix B

Secondly, I put AT andBT into matrix B. Because four lines of matrix can be saved in cache at the same time, there will little miss in transposition.



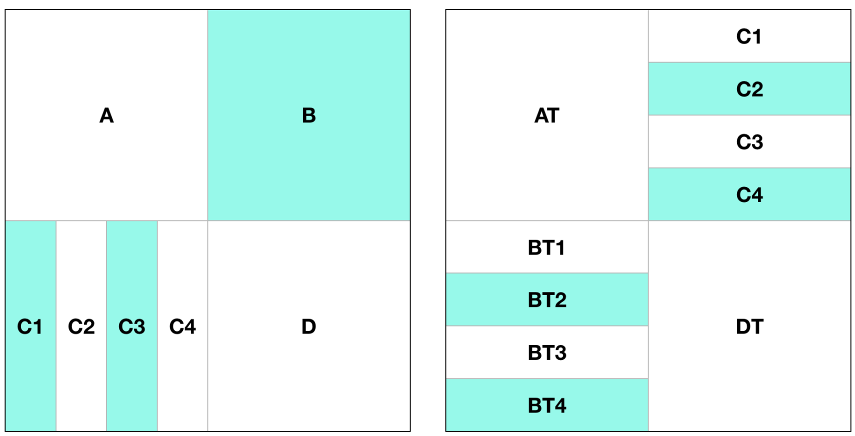

Matrix A Matrix B

Thirdly, I put BT into right place by row. And then I put CT to right place by column.




**Matrix A** **Matrix B**

At last, I put DT to matrix B.




**Matrix A** **Matrix B**

Even though I can not modify matrix A, I can matrix B at any time. When I transpose B and C in matrix A, I use empty space in B to temporarily save BT, which will reduce many conflict misses.

For 61x67 matrix, I can not calculate the best unit size for matrix transposition. So I just test different unit size to handle it and I find that use 16x16 matrix as a unit will meet the requirement.

### 2.2.2 Code

[In this part, you should place your code and make it readable in Latex, please. Writing necessary comments for codes is a good habit.]

trans.c

```c
//517030910116     Jingwei Xi
//email: jingweixi@sjtu.edu.cn
//This is the program for matrix transpose function
/*
 * 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"

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])
{
    int i, j, m, n;
    int a1, a2, a3, a4, a5, a6, a7, a8;
    if(N == 32 && M == 32){    // Matrix 32x32
        for(i = 0; i < 4; i++){
            for(j = 0; j < 4; j++){
                for(m = 0; m < 8; m++){
                    for(n = 0; n < 8; n++){
                        //For A[k][k], handle it later
```

```
                        if(i * 8 + m == j * 8 + n){
                            a1 = i * 8 + m;
                            a2 = A[i * 8 + m][j * 8 + n];
                            continue;
                        }
                        B[j * 8 + n][i * 8 + m] = A[i * 8 + m][j * 8 + n];
                    }
                    if(i == j){
                        //Handle the A[k][k]
                        B[a1][a1] = a2;
                    }
                }
            }
        }
        return;
    }

    if(N == 64 && M == 64){   //Matrix 64x64

        for(i = 0; i < 8; i++){
            for(j = 0; j < 8; j++){
                //Transpose A,B to AT, BT
                for(m = 0; m < 4; m++){
                    a1 = A[i * 8 + m][j * 8];
                    a2 = A[j * 8 + m][j * 8 + 1];
                    a3 = A[j * 8 + m][j * 8 + 2];
                    a4 = A[j * 8 + m][j * 8 + 3];
                    a5 = A[j * 8 + m][j * 8 + 4];
                    a6 = A[j * 8 + m][j * 8 + 5];
                    a7 = A[j * 8 + m][j * 8 + 6];
                    a8 = A[j * 8 + m][j * 8 + 7];

                    B[j * 8][i * 8 + m] = a1;
                    B[j * 8][i * 8 + m + 4] = a5;
                    B[j * 8 + 1][i * 8 + m] = a2;
                    B[j * 8 + 1][i * 8 + m + 4] = a6;
                    B[j * 8 + 2][i * 8 + m] = a3;
                    B[j * 8 + 2][i * 8 + m + 4] = a7;
                    B[j * 8 + 3][i * 8 + m] = a4;
                    B[j * 8 + 3][i * 8 + m + 4] = a8;

                }
                //Transfer BT and CT to right place
                for(m = 0; m < 4; m++){
                    a1 = B[j * 8 + m][i * 8 + 4];
```

```
                    a2 = B[j * 8 + m][i * 8 + 5];
                    a3 = B[j * 8 + m][i * 8 + 6];
                    a4 = B[j * 8 + m][i * 8 + 7];
                    a5 = A[i * 8 + 4][j * 8 + m];
                    a6 = A[i * 8 + 5][j * 8 + m];
                    a7 = A[i * 8 + 6][j * 8 + m];
                    a8 = A[i * 8 + 7][j * 8 + m];

                    B[j * 8 + m][i * 8 + 4] = a5;
                    B[j * 8 + m][i * 8 + 5] = a6;
                    B[j * 8 + m][i * 8 + 6] = a7;
                    B[j * 8 + m][i * 8 + 7] = a8;
                    B[j * 8 + m + 4][i * 8] = a1;
                    B[j * 8 + m + 4][i * 8 + 1] = a2;
                    B[j * 8 + m + 4][i * 8 + 2] = a3;
                    B[j * 8 + m + 4][i * 8 + 3] = a4;
                }
                //Transpose D to DT
                for(m = 0; m < 4; m++){
                    a1 = A[i * 8 + 4 + m][j * 8 + 4];
                    a2 = A[i * 8 + 4 + m][j * 8 + 5];
                    a3 = A[i * 8 + 4 + m][j * 8 + 6];
                    a4 = A[i * 8 + 4 + m][j * 8 + 7];

                    B[j * 8 + 4][i * 8 + m + 4] = a1;
                    B[j * 8 + 5][i * 8 + m + 4] = a2;
                    B[j * 8 + 6][i * 8 + m + 4] = a3;
                    B[j * 8 + 7][i * 8 + m + 4] = a4;
                }
            }
        }
        return;
    }
    //Matrix 61x67, use unit 16x16
    for(i = 0; i < 16; i++){
        for(j = 0; j < 16; j++){
            for(m = 0; m < 16 && m < N; m++){
                for(n = 0; n < 16 && n < M; n++){
                    B[j * 8 + n][i * 8 + m] = A[i * 8 + m
                        ][j * 8 + n];
                }
            }
        }
    }
}
```

```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;
}
```

### 2.2.3 Evaluation

[In this part, you should place the figures of experiments for your codes, prove the correctness and validate the performance with your own words for each figure's explanation.]

# 3 Conclusion

## 3.1 Problems

[In this part you can list the obstacles you met during the project, and better add how you overcome them if you have made it.]

In part A, I meet the problem that I do not know how to get the arguments of command line with uncertain number of arguments. With the help of partner and some technology blogs on the internet, I learned how to use the function getopt() to get the argument value.

The most difficult obstacle I met in this project is in part B. For transposition of matrix 64x64, I think of many ideas to handle it but they all over the 1300. I think a lot of ideas to optimize on handle in a unit matrix of 4x4, but I didn't think about the temporarily use the empty space of matrix B. After learning some best idea on the internet, I solve this problem.

### 3.2 Achievements

[In this part you can list the strength of your project solution, like the performance improvement, coding readability, partner cooperation and so on. You can also write what you have learned if you like.]

In part A, my program takes the same command line arguments and produces the identical output as the reference simulator. In part B, the algorithm I used has a good performance that misses of cache are all meet requirements.

In this project, I discuss the algorithm in part B with my partner several times. During the discuss, we all think about how to improve the performance of our program and talk about ideas with others. I make a good progress in during the optimization of our program.