Project 2: Understanding Cache Memories

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

June 5, 2019

1 Introduction

In this lab, I write two programs for two parts. In the first part, I write a program to simulate the hit, miss and evict behaviors of an LRU cache memory. In the program, I use array to be the data structure for cache. In the second part, I write a program to optimize cache performance for matrix transpose function.

2 Experiments

2.1 Part A

2.1.1 Analysis

The task for part A is to simulate 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 the data of block in cache is valid. The second variable is tag, which is used to check whether block is in the cache. The third one is the timeRef, which stores the time that the block recently be accessed.

I use the LRU (least-recently used) replacement policy when choosing which cache line to evict. For each data access, there will be three situation:

a. The data block is in the cache.

b. The data block is not in the cache but there is empty line in the set. It will put the block into empty line.

c. The data block is not in the cache and there is not empty line in the set. It will evict the line with minimum timeRef number, which indicate it has not been accessed for a long time.

2.1.2 Code

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");
```

```
41      printf("-v: Optional verbose flag that displays trace
            info\n");
42      printf("-s <s>: Number of set index bits (S = 2^s is
            the number of sets)\n");
43      printf("-E <E>: Associativity (number of lines per
            set)\n");
44      printf("-b <b>: Number of block bits (B = 2^b is the
            block size)\n");
45      printf("-t <tracefile>: Name of the valgrind trace to
            replay\n");
46  }
47
48  void initMainArg(){
49      mainArg.helpFlag = 0;
50      mainArg.verboseFlag = 0;
51      mainArg.setBit = 0;
52      mainArg.linePerSet = 0;
53      mainArg.blockBit = 0;
54      mainArg.fileName = NULL;
55  }
56
57  void cacheAccess(char type, long unsigned int addr, int
        size){
58      int hitFlag = 0, hitId = -1, emptyLine = -1, minTime
            = INT_MAX, evicId;
59      int setIndex = 0;
60      int dataTag;
61      int i;
62
63      //Address: |tag|setIndex|block_offset|
64      setIndex = (addr / (blockSize)) % (setCount);
65      dataTag = addr / (blockSize * setCount);
66
67      for(i = setIndex * mainArg.linePerSet; i < (setIndex
            + 1) * mainArg.linePerSet; i++){
68          //Hit
69          if(cache[i].tag == dataTag){
70              hitFlag = 1;
71              hitId = i;
72              break;
73          }
74          //Record the empty line id
75          if(cache[i].valid == 0 && emptyLine == -1){
76              emptyLine = i;
77          }
```

```
78          //Find the block line with the minimum timeRef
                number
79          if(cache[i].timeRef < minTime){
80              minTime = cache[i].timeRef;
81              evicId = i;
82          }
83      }
84
85      if (mainArg.verboseFlag){
86          printf("%c %lx,%x ",type,addr,size);
87      }
88      if(hitFlag == 1){     //Hit
89          cache[hitId].timeRef = timeClock;
90          hit++;
91          if(type == 'M'){
92              hit++;
93          }
94          if(mainArg.verboseFlag){
95              if(type == 'S' || type == 'L'){
96                  printf("hit\n");
97              }
98              else{
99                  printf("hit hit\n");
100             }
101         }
102
103     }
104     else{
105         if(emptyLine != -1){//Miss but there is empty
                line
106             cache[emptyLine].valid = 1;
107             cache[emptyLine].tag = dataTag;
108             cache[emptyLine].timeRef = timeClock;
109             miss++;
110             if(type == 'M'){
111                 hit++;
112             }
113             if(mainArg.verboseFlag){
114                 if(type == 'S' || type == 'L'){
115                     printf("miss\n");
116                 }
117                 else{
118                     printf("miss hit\n");
119                 }
120             }
121         }
```

```
122         else{    //Miss and no empty line, need to evict
123             cache[evicId].valid = 1;
124             cache[evicId].tag = dataTag;
125             cache[evicId].timeRef = timeClock;
126             miss++;
127             eviction++;
128             if(type == 'M'){
129                 hit++;
130             }
131             if(mainArg.verboseFlag){
132                 if(type == 'S' || type == 'L'){
133                     printf("miss evition\n");
134                 }
135                 else{
136                     printf("miss eviction hit\n");
137                 }
138             }
139         }
140     }
141 }
142
143 int main(int argc, char* argv[])
144 {
145     int opt;
146     int i;
147     char type;
148     int size;
149     long unsigned int addr;
150
151     initMainArg();//init each variable in argument struct
152
153     opt = getopt(argc, argv, "s:E:b:t:hv");
154     if(opt == -1){      //Invalid arguments
155         printHelp();
156         return -1;
157     }
158     while(opt != -1) {
159         switch(opt){
160             case 'v':
161                 mainArg.verboseFlag = 1; /* true */
162                 break;
163             case 's':
164                 mainArg.setBit = atoi(optarg);
165                 break;
166             case 'E':
167                 mainArg.linePerSet = atoi(optarg);
```

```
168                 break;
169             case 'b':
170                 mainArg.blockBit = atoi(optarg);
171                 break;
172             case 't':
173                 mainArg.fileName = optarg;
174                 break;
175             default:
176                 printHelp();
177                 break;
178         }
179         opt = getopt(argc, argv, "s:E:b:t:hv");
180     }
181
182     setCount = 1 << (mainArg.setBit);
183     blockSize = 1 << (mainArg.blockBit);
184
185     FILE *file = fopen(mainArg.fileName, "r");
186     if (file == NULL){    //Error: File not found
187         printf("File not found.");
188         return -1;
189     }
190
191     cache = (line *) malloc(setCount * mainArg.linePerSet
        * sizeof(line));
192         if (cache == NULL){    //Error: Cache space
            allocated failed
193             printf("Fail to allocate cache.");
194             return -1;
195         }
196
197     cacheSize = setCount * mainArg.linePerSet;
198     //Initialize the cache
199     for (i = 0; i < cacheSize; i++) {
200         cache[i].valid = 0;
201         cache[i].timeRef = 0;
202         cache[i].tag = -1;
203     }
204
205     while (!feof(file)){
206         int tmp = fscanf(file, " %c %lx,%x", &type, &addr
            , &size);
207         if (tmp != 3) continue;
208         if (type == 'I') continue;
209         cacheAccess(type, addr, size);
210         timeClock++;
```

```
211     }
212
213     free(cache);     //Free the cache space malloced
214     cache = NULL;
215     printSummary(hit, miss, eviction);
216     return 0;
217 }
```

2.1.3 Evaluation

Here is the result.

```
jingweixi@jingweixi-VirtualBox:~/Documents/archi2$ ./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
```

2.2 Part B

2.2.1 Analysis

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

For 32x32 matrix, each element in matrix with int type use 4 bytes. As our block size is 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 cache block distribution of matrix elements is showed in the picture.

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
...			

From the picture, we can see that we can deal with 8x8 matrix each time with few conflict misses because all matrix element are in the cache.

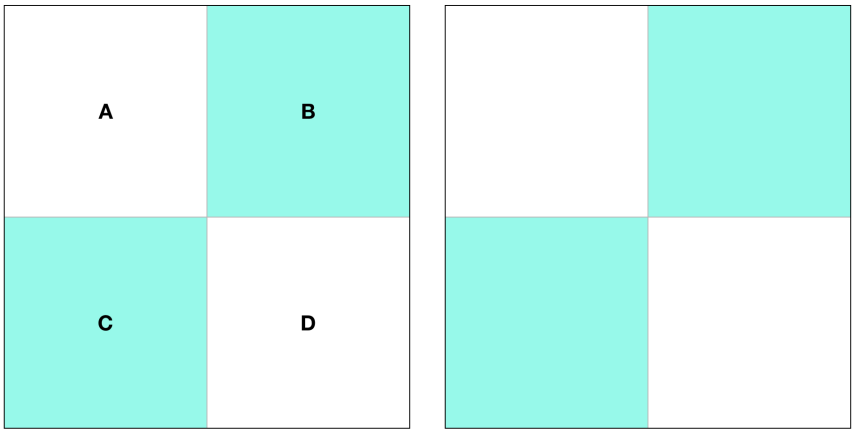
And for elements like A[i][i], the lines we'll use in matrix A and matrix B are mapped to the same cache line, which will generate unnecessary conflict misses. So we will use a temporary variant to store the value temporarily and transfer to matrix B later.

For 64x64 matrix, the situation is different from 32x32 before.

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	...				

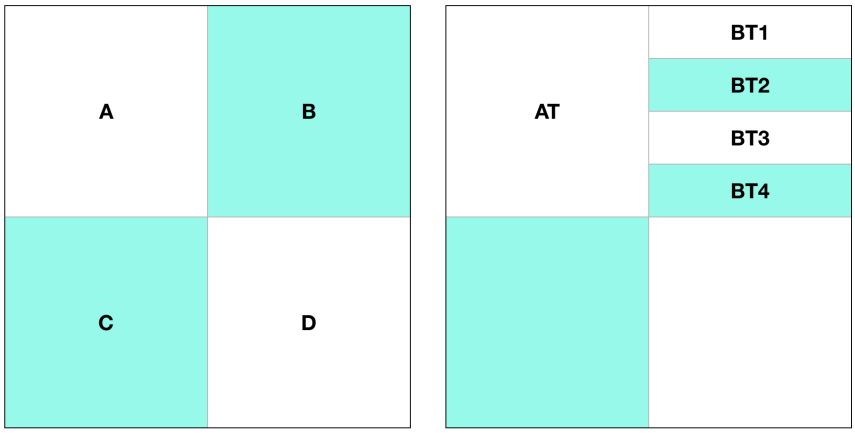
From the picture we can see that each line in matrix can be placed in 8 blocks and the cache will save 4 lines of matrix at one time. So only 4*4 matrix can be placed in cache at one time. If we transpose 4*4 matrix at on time, it can have little miss but it will waste a lot of space in cache line. The algorithm I developed will use 8*8 matrix which can fully use of cache line and have a good performance.

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.



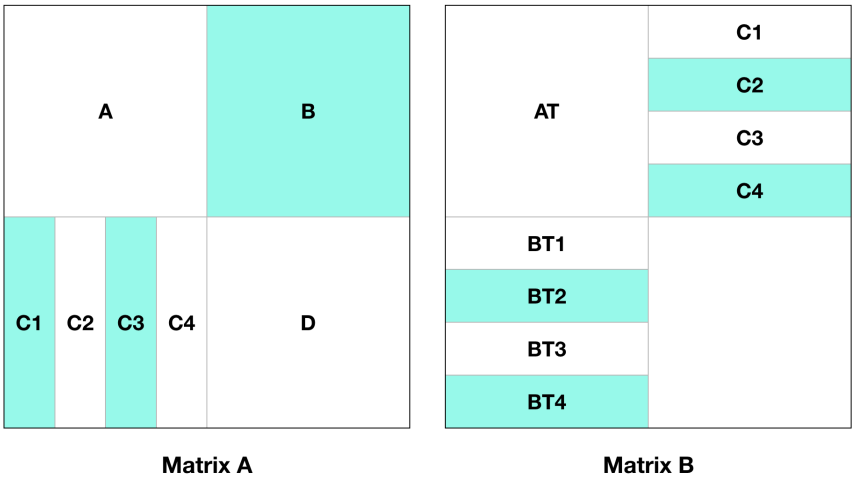
Matrix A Matrix B

Secondly, I put AT andBT into matrix B. Because four lines of matrix can be saved in cache at the same time, we can handle A and B at the same time with little misses.

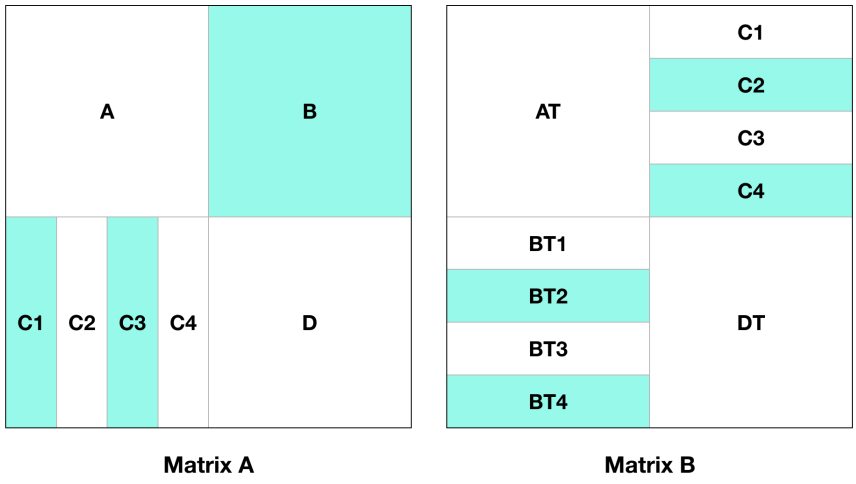


Matrix A Matrix B

Thirdly, I put BT into right place by row. And then I put CT to right place by column. The transfer algorithm is showed in the picture.



At last, I put DT to 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 sizes to handle it and I find that use 16x16 size matrix as the unit will meet the requirement.

2.2.2 Code

trans.c

```
1 //517030910116     Jingwei Xi
2 //email: jingweixi@sjtu.edu.cn
```

```
3  //This is the program for matrix transpose function
4  /*
5   * trans.c - Matrix transpose B = A^T
6   *
7   * Each transpose function must have a prototype of the
        form:
8   * void trans(int M, int N, int A[N][M], int B[M][N]);
9   *
10  * A transpose function is evaluated by counting the
        number of misses
11  * on a 1KB direct mapped cache with a block size of 32
        bytes.
12  */
13
14 #include <stdio.h>
15 #include "cachelab.h"
16
17 int is_transpose(int M, int N, int A[N][M], int B[M][N]);
18
19 /*
20  * transpose_submit - This is the solution transpose
        function that you
21  *      will be graded on for Part B of the assignment. Do
         not change
22  *      the description string "Transpose submission", as
        the driver
23  *      searches for that string to identify the transpose
         function to
24  *      be graded.
25  */
26 char transpose_submit_desc[] = "Transpose submission";
27 void transpose_submit(int M, int N, int A[N][M], int B[M
    ][N])
28 {
29     int i, j, m, n;
30     int a1, a2, a3, a4, a5, a6, a7, a8;
31     if(N == 32 && M == 32){    // Matrix 32x32
32         for(i = 0; i < 4; i++){
33             for(j = 0; j < 4; j++){
34                 for(m = 0; m < 8; m++){
35                     for(n = 0; n < 8; n++){
36                         //For A[k][k], handle it later
37                         if(i * 8 + m == j * 8 + n){
38                             a1 = i * 8 + m;
39                             a2 = A[i * 8 + m][j * 8 + n];
40                             continue;
```

```
41                          }
42                          B[j * 8 + n][i * 8 + m] = A[i * 8
                                + m][j * 8 + n];
43                      }
44                      if(i == j){
45                          //Handle the A[k][k]
46                          B[a1][a1] = a2;
47                      }
48                  }
49              }
50          }
51          return;
52      }
53
54      if(N == 64 && M == 64){   //Matrix 64x64
55
56          for(i = 0; i < 8; i++){
57              for(j = 0; j < 8; j++){
58                  //Transpose A,B to AT, BT
59                  for(m = 0; m < 4; m++){
60                      a1 = A[i * 8 + m][j * 8];
61                      a2 = A[j * 8 + m][j * 8 + 1];
62                      a3 = A[j * 8 + m][j * 8 + 2];
63                      a4 = A[j * 8 + m][j * 8 + 3];
64                      a5 = A[j * 8 + m][j * 8 + 4];
65                      a6 = A[j * 8 + m][j * 8 + 5];
66                      a7 = A[j * 8 + m][j * 8 + 6];
67                      a8 = A[j * 8 + m][j * 8 + 7];
68
69                      B[j * 8][i * 8 + m] = a1;
70                      B[j * 8][i * 8 + m + 4] = a5;
71                      B[j * 8 + 1][i * 8 + m] = a2;
72                      B[j * 8 + 1][i * 8 + m + 4] = a6;
73                      B[j * 8 + 2][i * 8 + m] = a3;
74                      B[j * 8 + 2][i * 8 + m + 4] = a7;
75                      B[j * 8 + 3][i * 8 + m] = a4;
76                      B[j * 8 + 3][i * 8 + m + 4] = a8;
77
78                  }
79                  //Transfer BT and CT to right place
80                  for(m = 0; m < 4; m++){
81                      a1 = B[j * 8 + m][i * 8 + 4];
82                      a2 = B[j * 8 + m][i * 8 + 5];
83                      a3 = B[j * 8 + m][i * 8 + 6];
84                      a4 = B[j * 8 + m][i * 8 + 7];
85                      a5 = A[i * 8 + 4][j * 8 + m];
```

```
86                          a6 = A[i * 8 + 5][j * 8 + m];
87                          a7 = A[i * 8 + 6][j * 8 + m];
88                          a8 = A[i * 8 + 7][j * 8 + m];
89
90                          B[j * 8 + m][i * 8 + 4] = a5;
91                          B[j * 8 + m][i * 8 + 5] = a6;
92                          B[j * 8 + m][i * 8 + 6] = a7;
93                          B[j * 8 + m][i * 8 + 7] = a8;
94                          B[j * 8 + m + 4][i * 8] = a1;
95                          B[j * 8 + m + 4][i * 8 + 1] = a2;
96                          B[j * 8 + m + 4][i * 8 + 2] = a3;
97                          B[j * 8 + m + 4][i * 8 + 3] = a4;
98                      }
99                      //Transpose D to DT
100                     for(m = 0; m < 4; m++){
101                         a1 = A[i * 8 + 4 + m][j * 8 + 4];
102                         a2 = A[i * 8 + 4 + m][j * 8 + 5];
103                         a3 = A[i * 8 + 4 + m][j * 8 + 6];
104                         a4 = A[i * 8 + 4 + m][j * 8 + 7];
105
106                         B[j * 8 + 4][i * 8 + m + 4] = a1;
107                         B[j * 8 + 5][i * 8 + m + 4] = a2;
108                         B[j * 8 + 6][i * 8 + m + 4] = a3;
109                         B[j * 8 + 7][i * 8 + m + 4] = a4;
110                     }
111                 }
112             }
113             return;
114         }
115         //Matrix 61x67, use unit 16x16
116         for(i = 0; i < 16; i++){
117             for(j = 0; j < 16; j++){
118                 for(m = 0; m < 16 && m < N; m++){
119                     for(n = 0; n < 16 && n < M; n++){
120                         B[j * 8 + n][i * 8 + m] = A[i * 8 + m
                                ][j * 8 + n];
121                     }
122                 }
123             }
124         }
125     }
126
127     /*
128      * You can define additional transpose functions below.
            We've defined
129      * a simple one below to help you get started.
```

```
130  */
131
132 /*
133  * trans - A simple baseline transpose function, not
         optimized for the cache.
134  */
135 char trans_desc[] = "Simple row-wise scan transpose";
136 void trans(int M, int N, int A[N][M], int B[M][N])
137 {
138     int i, j, tmp;
139
140     for (i = 0; i < N; i++) {
141         for (j = 0; j < M; j++) {
142             tmp = A[i][j];
143             B[j][i] = tmp;
144         }
145     }
146
147 }
148
149 /*
150  * registerFunctions - This function registers your
         transpose
151  *     functions with the driver.  At runtime, the driver
          will
152  *     evaluate each of the registered functions and
         summarize their
153  *     performance. This is a handy way to experiment
         with different
154  *     transpose strategies.
155  */
156 void registerFunctions()
157 {
158     /* Register your solution function */
159     registerTransFunction(transpose_submit,
            transpose_submit_desc);
160
161     /* Register any additional transpose functions */
162     registerTransFunction(trans, trans_desc);
163
164 }
165
166 /*
167  * is_transpose - This helper function checks if B is the
          transpose of
```

```
168  *     A. You can check the correctness of your transpose
         by calling
169  *     it before returning from the transpose function.
170  */
171 int is_transpose(int M, int N, int A[N][M], int B[M][N])
172 {
173     int i, j;
174
175     for (i = 0; i < N; i++) {
176         for (j = 0; j < M; ++j) {
177             if (A[i][j] != B[j][i]) {
178                 return 0;
179             }
180         }
181     }
182     return 1;
183 }
```

2.2.3 Evaluation

Here are the results and correctness of my program.

```
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        1227
Trans perf 61x67          10.0        10        1992
          Total points    53.0        53
```

3 Conclusion

3.1 Problems

In part A, I met the problem that I didn't 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, the optimization of matrix 32x32 and 61x67 are both not work well. I think a lot of ideas to optimize on handling in a unit matrix of 4x4, but they all did not meet requirements. After learning some best idea on the internet and discussing with partners , I solve this problem by temporarily using the empty space in matrix B.

3.2 Achievements

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 performance of our program and talk about ideas with others. I make a good progress during the time as we optimize our algorithm.

At the same time, I learned a lot of knowledge of cache and make a deep comprehension of cache LRU algorithm. In addition, with the practice in part B, l learned the importance of studying computer architecture. With the knowledge of computer architecture, we can program in good structure to make best use of hardware and get the best performance.

In conclusion, l learned a lot about cache in this lab.