Project 2: Understanding Cache Memories

516030910259, Xinpeng Liu, lxp0907@163.com

May 28, 2018

1 Introduction

In this lab, I am asked to do some work about the cache.
In the Part A, I implement a cache simulator that can simulate the hit/miss behavior of an LRU cache memory with arbitrary size and associativity in C.
In the part B, I optimize the performance of the C program $trans.c$ running on the cache we simulate in Part A by eliminating the misses it generates.

2 Experiments

2.1 Part A

2.1.1 Analysis

This part is actually easy. I will focus on the implementation of the cache.
First, I define a data structure $cacheLine$, composed of three elements: an int variant $valid$ indicating whether this line is valid (in this lab, empty), an $long$ variant tag storing the tag of the line, and another int variant t storing the latest time that this line was accessed.
Then, for every data access, there are three situations:

1. The data is in the cache. For ‘S‘ and ‘L‘, it's simply hit. For ‘M‘, it's $hit\ hit$.

2. The data is not in the cache, but there exists an empty line in cache to contain it. For ‘S‘ and ‘L‘ instructions, this is $miss$. For ‘M‘ instructions this is $miss\ hit$.

3. The data is not in the cache and there are no room for it. Then we need to find the least recently used line, and replace that line with the data line we want. For ‘S‘ and ‘L‘, this is $miss\ eviction$. For ‘M‘, this is $miss\ eviction\ hit$.

So the job turns out to be clear. For every access that wants line A, executing the following algorithm:

1. If A is in cache, save the address as dst, go to 4.

2. If there is an empty line in cache that can contain A, save the address as dst, go to 4.

3. Find the least recently used line that can contain A, save the address as dst,

4. Update dst's $valid$, tag and t. Update hit, $miss$ and $eviction$ counts according to different situations.

2.1.2 Code

```
//516030910259 Xinpeng Liu
#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 tag;
  int t;
} cacheLine;

cacheLine *cache;// Cache
int s,e,b;// S, E, B
int verboseFlag=0;// Whether verbose mode is used.
int hit=0,miss=0,eviction=0
int nowt=0;// Time for the access.
char temp[25];
char* filePath=NULL;// Path of the trace file.

void printHelp();
void accessCache(char type,
                 long unsigned int addr,
                 int size);

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

```c
    case 'b':
      b=(int) pow(2,atoi(optarg));break;
    case 't':
      filePath=optarg;break;
    case 'v':
      verboseFlag=1;break;
    default:
      printHelp();return 1;break;
      //Error type 1: invalid arguments and -h.
      //Output Help doc.
  }
}//Handling arguments

if (in==0 && tmp==-1){
  printHelp();
  return 1;
}
//Error type 1: invalid arguments. Output Help doc.

FILE *file=fopen(filePath,"r");
if (file==NULL){
  printf("File not found.");
  return 2;
}
//Error type 2: File not found.
cache=(cacheLine*)malloc(s*e*sizeof(cacheLine));
if (cache==NULL){
  printf("Fail to allocate cache.");
  return 3;
}
//Error type 3: Fail to allocate cache.
cacheLine* ptr;
for (int i=0;i<s*e;i++) {
  ptr=(cache+i);
  ptr->valid=0;
}
//Initialize cache.
char type;
int size;
long unsigned int addr;
while (!feof(file)){
  int tr=fscanf(file, " %c %lx,%x",&type, &addr, &size);
  if (tr!=3) continue;
  if (type=='I') continue;
  //Invalid insructions.
  accessCache(type,addr,size);
```

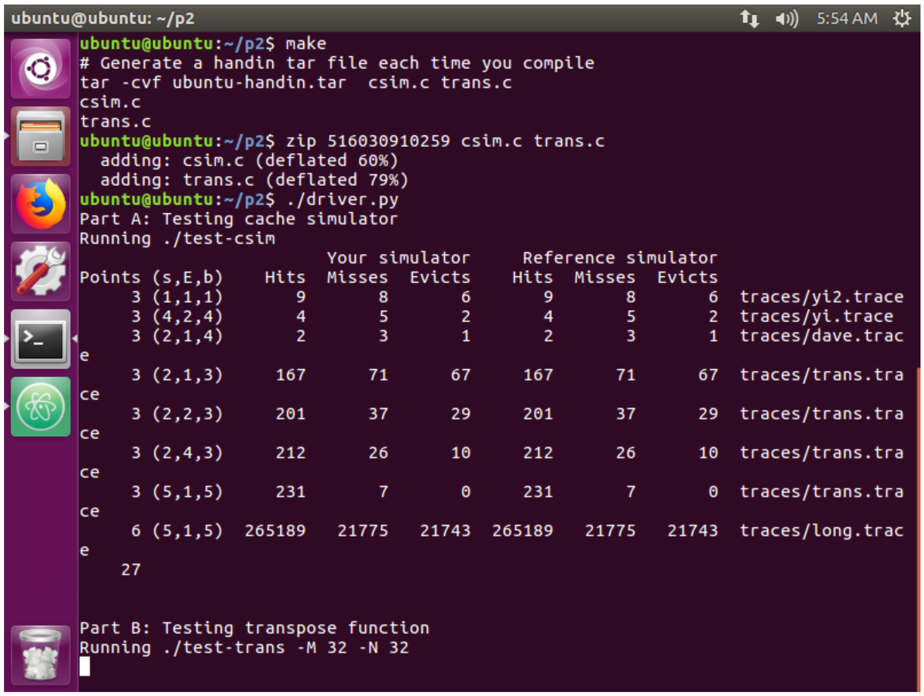
```c
    nowt++;
  }
  free(cache);
  cache=NULL;
  printSummary(hit, miss, eviction);
  return 0;
}

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)
    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 accessCache(char type, long unsigned int addr, int size){
  int sIndex=(int) ((addr/b)%s);
  int tag=(int) (addr/(b*s));
  int insert=-1;// Address used by situation 2.
  int replace=-1;// Address used by situation 3.
  int find=-1;// Address used by situation 1.
  int i,mint=nowt;
  cacheLine* ptr;
  for (i=sIndex*e;i<(sIndex+1)*e;i++){
    ptr=(cache+i);
    if (ptr->tag==tag && ptr->valid) {
      find=i;
      break;
      // Situation 1
    }
    else if (!ptr->valid) {insert=i;}
    // Situation 2
    else if (ptr->t<mint){
      replace=i;mint=ptr->t;
      // Situation 3
    }
  }
  if (verboseFlag) printf(" %c %lx,%x ",type,addr,size);
  // Verbose mode.
  if (find!=-1){
    ptr=cache+find;
    ptr->t=nowt;
    hit++;
```

```
    if (verboseFlag) printf("hit");
    // Situation 1
  }
  else if (insert!=-1){
    ptr=cache+insert;
    ptr->valid=1;
    ptr->tag=tag;
    ptr->t=nowt;
    miss++;
    if (verboseFlag) printf("miss");
    // Situation 2
  }
  else {
    ptr=cache+replace;
    ptr->tag=tag;
    ptr->t=nowt;
    miss++;eviction++;
    if (verboseFlag) printf("miss eviction");
    // Situation 3
  }
  if (type=='M') {
    hit++;
    if (verboseFlag) printf(" hit");
    // M instructions.
  }
  if (verboseFlag) printf("\n");
}
```

2.1.3 Evaluation



The evaluation results show the correctness of my simulator.

2.2 Part B

2.2.1 Analysis

We know each cache line contains 8 int elements. And I'll try to exploit it to optimize the performance.
For 32x32 matrix, we have the following table, each square represents 8 elements, and the number indicates which cache line it will be put into.

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
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
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
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

In the picture above, we can see that if we handle at most an 8x8 matrix at the same time without conflict misses. 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 64x64 matrix, the situation is a little different.

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	4	5	6	7
…………							

Now we can handle at most a 4x4 matrix at the same time without conflict misses. And I also use the diagonal optimization. However, if I want to achieve higher marks, I'll have to use another optimization.

This optimization will use 8x8 matrix as unit to make full use of each cache line. The idea can be described by the following pictures:

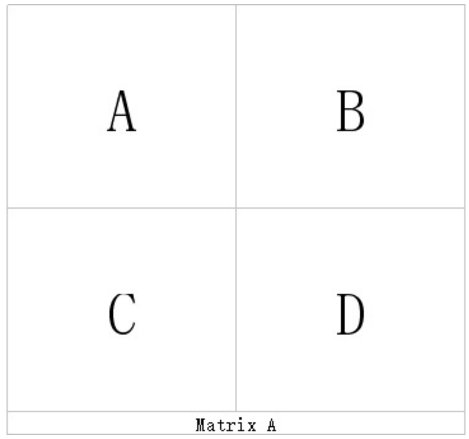


Figure 1: Divide A into 4 4x4 matrices: A, B, C, D

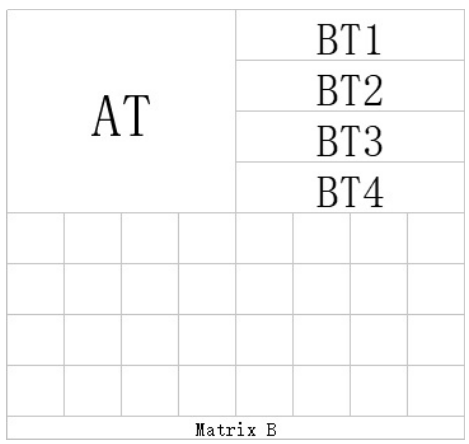


Figure 2: Put A^T and B^T into matrix B

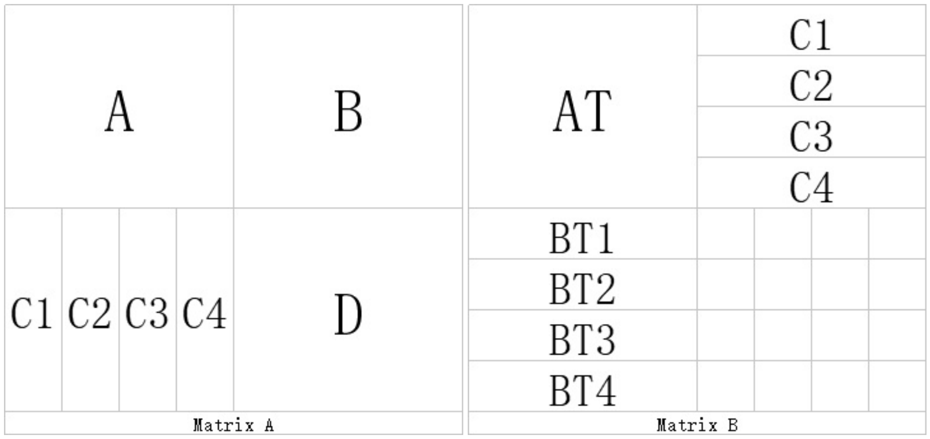


Figure 3: Put B^T and C^T into right places by line

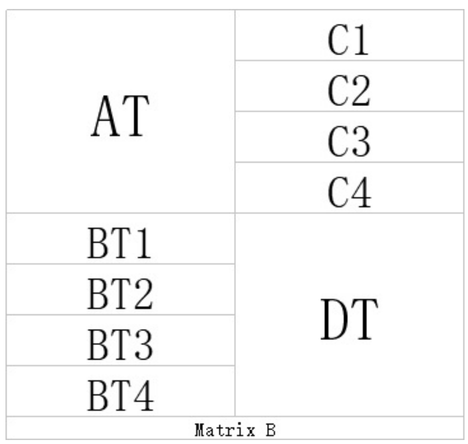


Figure 4: Put D^T into Matrix B

And for 8x8 matrices lying on the diagonal, I handle them before the others. And when handling them, I used the empty space in B as temporary space to help reduce conflict misses.
Finally, for 61x67 matrix, it's hard to tell the "unit" size of a matrix we can handle without conflict misses. Through testing, I finally find that use 16x16 matrix as a unit achieve max marks.

2.2.2 Code

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N]){
    if (N==32){
```

```
    int i, j, p, q, tmp, k;
    for (i = 0; i < N/8; i++)
      for (j = 0; j < M/8; j++)
        for (p=0;p<8;p++){
          for (q=0;q<8;q++){
            if (j*8+q==i*8+p){
              tmp=A[i*8+p][j*8+q];
              k=j*8+q;
              continue;
              // for elements A[k][k], postpone to avoid miss
            }
            B[j*8+q][i*8+p]=A[i*8+p][j*8+q];
          }
          if (i==j) B[k][k]=tmp;
          // for elements A[k][k], postpone to avoid miss
        }
  }// divide matrix by 8x8
  else if (N==64){
    int i, j, p, q;
    int t[4];
    for (i=0;i<N;i+=8){
      for (p=0;p<4;p++){
        for (q=0;q<8;q++)
          B[p][q+8]=A[i+p][i+q];
        for (q=0;q<8;q++)
          B[p][q+16]=A[i+p+4][i+q];
      }
      for (p=0;p<8;p++){
        for (q=0;q<4;q++)
          B[i+p][i+q]=B[q][p+8];
        for (q=4;q<8;q++)
          B[i+p][i+q]=B[q-4][p+16];
      }
    }
    //for diagonal matrixes, use other matrixes as temporary space
    for (i = 0; i < N/8; i++)
      for (j = 0; j < M/8; j++){
        if (i==j) continue;
        for (p=0;p<4;p++){
          for (q=0;q<4;q++)
            B[j*8+q][i*8+p]=A[i*8+p][j*8+q];
          for (q=4;q<8;q++)
            B[j*8+q-4][i*8+p+4]=A[i*8+p][j*8+q];
        }//use the top right 4x4 matrix of B as temporary space
        for (p=0;p<4;p++){
          for (q=4;q<8;q++)
```

```
                t[q-4]=B[j*8+p][i*8+q];
                //store the pth line of the top right matrix in t
              for (q=4;q<8;q++)
                B[j*8+p][i*8+q]=A[i*8+q][j*8+p];
                //fill the pth line of the top right matrix with right values
              for (q=4;q<8;q++)
                B[j*8+p+4][i*8+q-4]=t[q-4];
                //fill the bottom left matrix with t
            }
            for (p=4;p<8;p++)
              for (q=4;q<8;q++)
                B[j*8+q][i*8+p]=A[i*8+p][j*8+q];
                //fill the bottom right matrix with right values
          }
    }//divide matrix by 8x8, and change the order of handling 4x4 matrixes
    else {
      int i, j, p, q, tmp, k;
      for (i = 0; i <= N/16; i++)
        for (j = 0; j <= M/16; j++)
          for (p = 0; p < 16 && i*16+p<N; p++){
            for (q = 0; q < 16 && j*16+q<M; q++){
              if (j*16+q == i*16+p){
                tmp=A[i*16+p][j*16+q];
                k=j*16+q;
                continue;
                // for elements A[k][k], postpone to avoid miss
              }
              B[j*16+q][i*16+p] = A[i*16+p][j*16+q];
            }
            if (i==j) B[k][k]=tmp;
            // for elements A[k][k], postpone to avoid miss
          }
    }//divide matrix by 16x16
}
```

2.2.3 Evaluation

```
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         289
Trans perf 64x64           8.0         8        1205
Trans perf 61x67          10.0        10        1987
          Total points    53.0        53
ubuntu@ubuntu:~/p2$ 
```

The result shows the correctness and performance of my optimization as I expressed above.

3 Conclusion

3.1 Problems

The main problem I met in this lab is in part B. The optimization implemented on 32x32 and 61x67 can't work well on the 64x64 matrix. However, by analyzing, I come up with the idea finally makes it to achieve max marks. And it works well.

3.2 Achievements

In the part A of this lab, I implemented a simple cache simulator. This part helps me to develop a deep and practical comprehension of cache.
In the part B of this lab, to achieve higher marks, I began to know about blocking. And the implementation of blocking optimization makes me know how a program's structure change can have an impact on its performance, and the importance to know about computer architecture even when programming with higher level programming language. In addition, I find that in this part, I'm asked to use a direct-mapped cache, and this results in the most of the misses. So it also reminds me of that higher associativy helps reducing misses.
In conclusion, this lab expands the content of the class, and helps me with the comprehension of how cache works.