Project 2: Understanding Cache Memories

520021910431,Yaoxuan Cheng,
2212582443@sjtu.edu.cn

May 16, 2022

1 Introduction

The project: **Understanding Cache Memories** is the second project of CS2305: Computer Architecture. By completing this experiment, we can gain a deeper understanding of how memory interacts with the cache, and how to optimize the code to make the cache hit rate higher. This project consists of two parts:

- **Part A: Writing a Cache Simulator**
  In this part, we need to finish a *C* program with the file name *csim.c*. This program take the *valgrind* memory trace as the input to simulate the cache's behaviours and count the number of **cache hits, cache misses, evictions.** We can better observe the actions of the cache and get a deeper understanding of it in this part.
- **Part B: Optimizing Matrix Transpose**
  In this part, we need to optimize the *transpose_submit* function in a *C* program with the file name *trans.c*, which aims to store the **transpose of matrix** A into matrix B. To optimize the function, we need to make the cache misses as few as possible. Based on the understanding of cache, this part focuses us to have a full comprehension in the cache misses and think about methods of optimizing a program by attaching special attention on the cache.

2 Experiments

The project: **Understanding Cache Memories**

2.1 Part A

2.1.1 Analysis

Generic Cache Memory Organization: In a computer system that each memory address has m bits, a cache is organized as an array of $S = 2^s$ cache sets. Each set consists of E cache lines. Each line consists of a data block of $B = 2^b$ bytes, a *valid bit* that indicates whether or not the line contains meaningful information, and $t = m-(b+s)$ *tag bits* (a subset of the bits from the current block's memory address) that uniquely identify the block stored in the cache line. As illustrated in Figure 1.

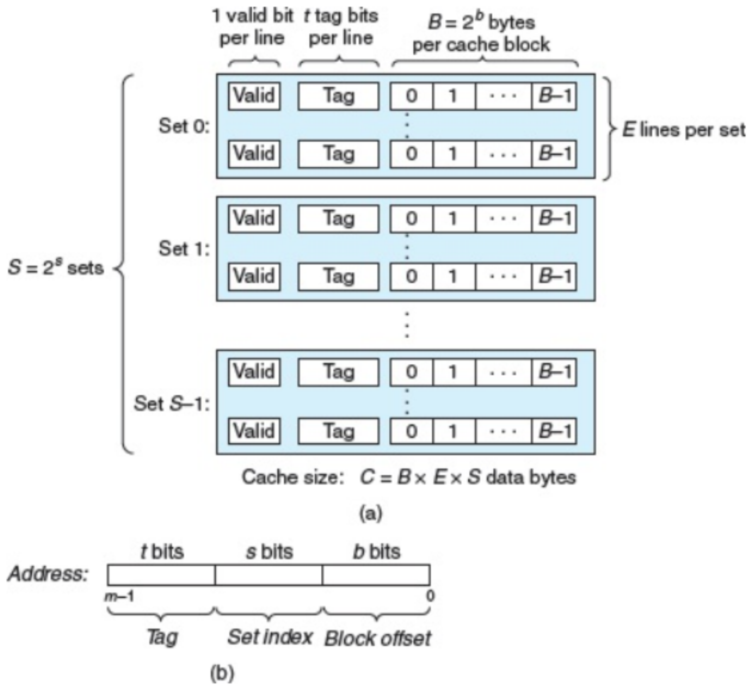


Figure 1: Generic Cache Memory Organization

In our class, we have learned about three block placement way: **fully associative, direct mapped, set associative,** which are very similar essentially. By showing Figure.1 we know that they are actually all set associative, different in the set size.

Instruction: In this part, we will meet 4 types of instructions in **trace files :**

- **I(instruction load):**Fetch instructions from instruction memory without accessing the data cache
- **L(data load):** Load *size* bytes from *address*
- **M(data modify):** Load *size* bytes from *address*, modify them, and store them back into *address*.(load + store)
- **S(data store):** Store *size* bytes into *address*

The form of each line in **trace files** is : [space]**operation address,size**
There is never a space before each "I". Thus, when we are parsing each operation, we should ignore all instruction cache accesses (lines starting with "I") and consider the "M" operation may lead to hit/miss twice.

LRU(Least Recently Used): LRU replacement associates with each page the time of that page's last use.When a page must be replaced, LRU chooses the page that has not been used for the longest period of time. We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.

2.1.2 Code

In this section, we show the specific implementation of our simulator.The structure of the **main()** is as follows:

main()

```
int main(int argc, char*argv[])
{
    verbose=0,s=-1,S=-1,E=-1,b=-1,B=-1;
    char trace_name[50];
    int ch;
    opterr=0;
    while ((ch = getopt(argc,argv,"hvs:E:b:t:"))!= -1)
    {
        switch (ch)
        {
        case 'h':
            printf("Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>\n"
                        "-h: Optional help flag that prints usage info\n"
                        "-s <s>: Number of set index bits (S = 2^s is the number of sets)\n"
                        "-E <E>: Associativity (number of lines per set)\n"
                        "-b <b>: Number of block bits (B = 2^b is the block size)\n"
                        "-t <tracefile>: Name of the valgrindtrace to replay\n");
            break;
        case 'v':
            verbose=1;
            break;

        case 's':
            s=atoi(optarg);
            S=1<<s;
            break;

        case 'E':
            E=atoi(optarg);
            break;

        case 'b':
            b=atoi(optarg);
            B=1<<b;
            break;

        case 't':
            strcpy(trace_name, optarg);
            break;

```

```
            default:
                printf("Please input the correct command format.\n");
                printf("Usage: ./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>\n"
                       "-h: Optional help flag that prints usage info\n"
                       "-s <s>: Number of set index bits (S = 2^s is the number of sets)\n"
                       "-E <E>: Associativity (number of lines per set)\n"
                       "-b <b>: Number of block bits (B = 2^b is the block size)\n"
                       "-t <tracefile>: Name of the valgrindtrace to replay\n");
                return -1;
        }
    }
    if(S<0||E<0||B<0) return -1;
    input=fopen(trace_name,"r");
    if(input==NULL)
    {
      printf("Can't find the corresponding trace file.\n");
      return-1;
    }

    init();

   char operation;
   unsigned int address;
   int size;

   while(fscanf(input," %c %x,%d",&operation,&address,&size)!=EOF) memory_access(operation,address,size);

   printSummary(hits, misses, evictions);

   for(int i=0; i<S; ++i)  free(cache[i]);
   free(cache);
   fclose(input);

   return 0;
}
```

In the **main()**, we first get cache-related information from the console through the while loop, including s, E, b and so on. And we will judge and report some simple error conditions.

And then we will call the **init()** to initialize the cache and program, the code is as follows:

init()

```
void init()
{
  hits=0;
  misses=0;
  evictions=0;
  time=0;

  cache = (cache_line**)malloc(sizeof(cache_line*) * S);
  for(int i=0; i<S; i++)
  {
    cache[i] = (cache_line*)malloc(sizeof(cache_line) * E);
    for(int j=0; j<E; j++)
    {
      cache[i][j].valid = 0;
      cache[i][j].tag = -1;
      cache[i][j].RU = -1;
    }
  }
}
```

We can see that in this program, *cache* is a two-dimensional dynamic array composed of *cache_line*.The definition of *cache_line* is as follows:

cache_line

```
typedef struct CACHE_LINE
{
    int valid;
    int tag;
    int RU;
}cache_line;
cache_line **cache;
```

After initialization, the program will read instructions from the *input* file. Through the while loop, it obtain the three elements of the instruction(*operation*, *address*, *size*) and use **memory_access()** to execute instructions until the file is all read. The code is as follows:

memory_access()

```
void memory_access(char operation,unsigned int address,int size)
{
   if(operation!='L'&&operation!='S'&&operation!='M') return;
   int tag = (address>>(b+s));
   int set = (address>>b) & ((0xFFFFFFFF)>>(64-s));
```

```
    int if_hit=0;
    int if_eviction=1;

    for(int i=0;i<E;i++)
    {
      if(cache[set][i].tag==tag&&cache[set][i].valid)
      {
        if_hit=1;
        hits++;
        cache[set][i].RU=time;
        break;
      }
    }

    if(!if_hit)
    {
      misses++;
      for(int i=0;i<E;i++)
      {
        if(!cache[set][i].valid)
        {
          cache[set][i].valid=1;
          cache[set][i].tag=tag;
          cache[set][i].RU=time;
          if_eviction=0;
          break;
        }
      }

      if(if_eviction)
      {
        evictions++;
        int earliest_time=time;
        int earliest_lines;
        for(int i=0;i<E;i++)
        {
          if(cache[set][i].RU<earliest_time)
          {
            earliest_time=cache[set][i].RU;
            earliest_lines=i;
          }
        }
        cache[set][earliest_lines].valid=1;
        cache[set][earliest_lines].tag=tag;
        cache[set][earliest_lines].RU=time;

      }
```

```

    }

    if(operation=='M') hits++;

    time++;

    if(verbose)
    {
      switch(operation)
      {
      case 'L':
          printf("%c %x,%d ",operation,address,size);

          if(if_hit)
          {
            printf("hit\n");
            break;
          }else printf("miss");

          if(if_eviction) printf(" eviction\n");
          else printf("\n");
          break;

      case 'S':
          printf("%c %x,%d ",operation,address,size);

          if(if_hit)
          {
            printf("hit\n");
            break;
          }else printf("miss");

          if(if_eviction) printf(" eviction\n");
          else printf("\n");
          break;

      case 'M':
          printf("%c %x,%d ",operation,address,size);

          if(if_hit)
          {
            printf("hit\n");
            break;
          }else printf("miss");

          if(if_eviction) printf(" eviction hit\n");
          else printf(" hit\n");
```

```
            break;

        default:break;

        }
    }
}
```

In **memory_access()**, the execution process of the code can be divided into 5 steps:

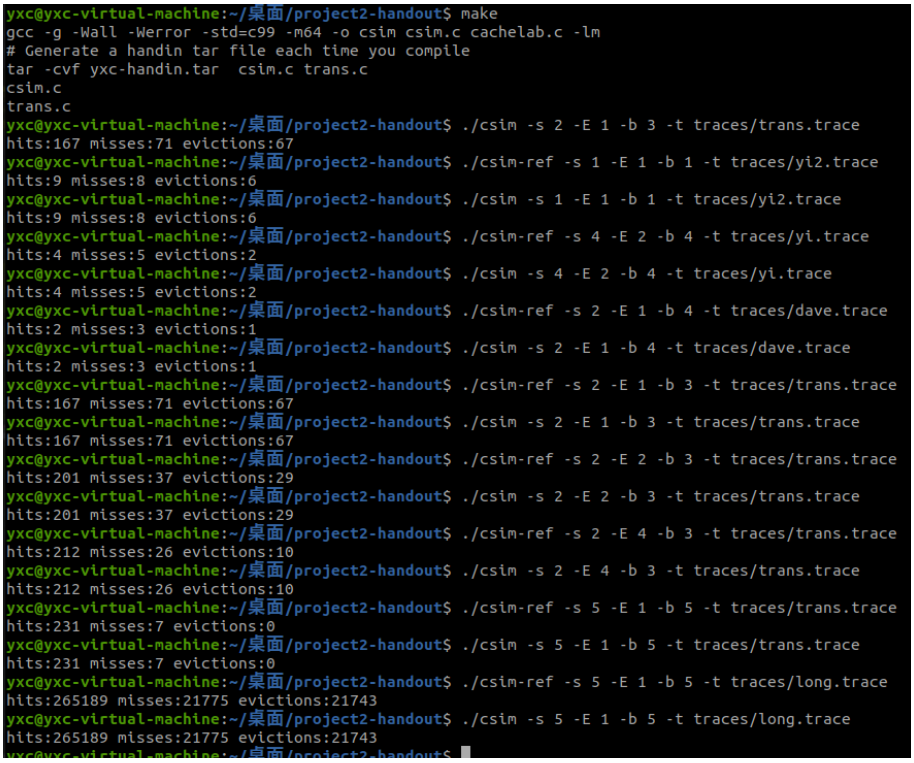
1. Parse the *address* into *tag* and *set*
2. Search in the set for the existence
   (a) If hit, update *RU*
   (b) If not hit, go to next step
3. Search in the set for the empty space
   (a) If found, update *valid, tag and RU*
   (b) If not found, go to next step
4. Use **LRU** to replace the block. Update *valid, tag and RU*
5. Determine whether to print memory access details based on the *verbose* variable

It should be emphasized again that the **"M"** operation may lead to hit/miss twice.

At the end of **main()**, we will call the **printSummary()** to print the *hits, misses and evictions* recorded during the memory access and do some finishing touches.

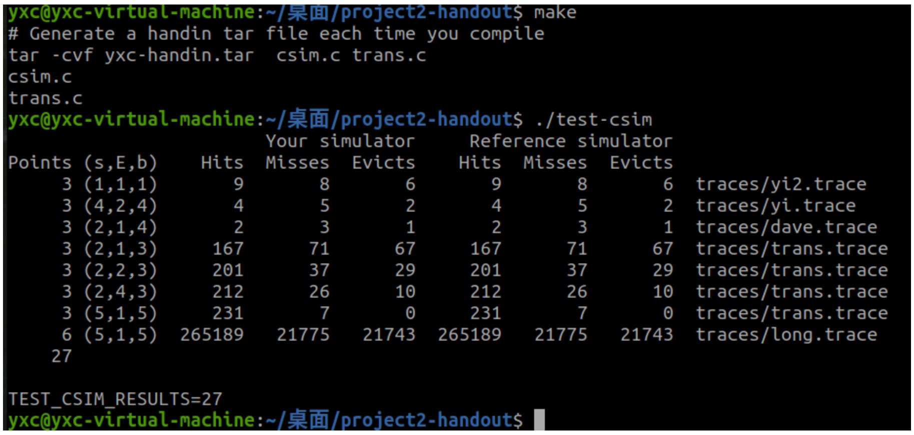
2.1.3 Evaluation

In Figure.2 and Figure.3, we show the results of our program, auto graded by **test-csim**. Thus,we successfully build our Cache Simulator.



```
yxc@yxc-virtual-machine:~/桌面/project2-handout$ make
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
# Generate a handin tar file each time you compile
tar -cvf yxc-handin.tar  csim.c trans.c
csim.c
trans.c
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim-ref -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
yxc@yxc-virtual-machine:~/桌面/project2-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
yxc@yxc-virtual-machine:~/桌面/project2-handout$
```

Figure 2: Evaluation Results



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

Figure 3: Score of Part A

2.2 Part B

2.2.1 Analysis

There are several tricks we could apply to optimize matrix transpose, but only a few of them could meet our requirements. Besides, different size of matrix have to use different ways to get the best results. Next will directly state the best algorithm I got for each case.

32 × 32 : Due to the experimental limit, we can use up to 12 temporary variables. After removing the loop variable i, j , we can use up to 10. In **Part B**, we use the cache with $s = 5$, $E = 1$, $b = 5$. Because the **int** variable occupies 4 bytes, one cache line could contain only $2^b \div 4 = 2^5 \div 4 = 8$ **int** variable. And the address of the row elements of the two-dimensional array is continuous in memory, so reading 8 elements on the same row at one time can make full use of a cache hit. We divide the whole matrix into smaller size($\mathbf{8 \times 8}$) of blocks and loop inside block with temporary variable k. In order to avoid diagonal conflict misses, we use 8 variables to undertake between the row of matrix A and the column of matrix B. See the detailed code in the next section.

$\mathbf{64 \times 64}$: Similar to $\mathbf{32 \times 32}$, we still read 8 numbers at once. The difference is that since $2^{s+b} \div (4 \times 64) = 2^{10} \div 2^{8} = 4$, so the elements in row i and row $i+4$ have the same s in their addresses,which means once we access row $i+4$, the cache line corresponding to row i will be overwritten. So, whether in matrix A or B, we try our best to only access elements in 4 rows at a time. So we will try to divide the $\mathbf{8 \times 8}$ blocks into smaller size $\mathbf{4 \times 4}$ to get a better algorithm.See the detailed code in the next section.

$\mathbf{61 \times 67}$: The transpose of a $\mathbf{61 \times 67}$ matrix is no different from the previous one, except that the number of rows and columns is not divisible by 8. So we use the simplest way(read 8 numbers at once) to transpose elements in the matrix $\mathbf{56 \times 64}$ and solve the remainder with two double loops.See the detailed code in the next section.

2.2.2 Code

The detailed code in the *trans.c* is as follows:

32 × 32

```
int i,j,k,t0,t1,t2,t3,t4,t5,t6,t7;//11
    for(i=0; i<N; i+=8)
        for(j=0; j<M; j+=8)
            for(k=i; k<i+8; k++)
            {
                t0 = A[k][j];
                t1 = A[k][j+1];
                t2 = A[k][j+2];
                t3 = A[k][j+3];
                t4 = A[k][j+4];
                t5 = A[k][j+5];
                t6 = A[k][j+6];
                t7 = A[k][j+7];

                B[j][k] = t0;
                B[j+1][k] = t1;
                B[j+2][k] = t2;
                B[j+3][k] = t3;
                B[j+4][k] = t4;
                B[j+5][k] = t5;
                B[j+6][k] = t6;
                B[j+7][k] = t7;
            }
```

64×64

```
int i,j,k,t0,t1,t2,t3,t4,t5,t6,t7;//11
    for(i=0; i<N; i+=8)
        for(j=0; j<M; j+=8)
        {
          for(k=i;k<i+4;k++)
          {
            t0 = A[k][j];
            t1 = A[k][j+1];
            t2 = A[k][j+2];
            t3 = A[k][j+3];
            t4 = A[k][j+4];
            t5 = A[k][j+5];
            t6 = A[k][j+6];
            t7 = A[k][j+7];

            B[j][k] = t0;
            B[j+1][k] = t1;
            B[j+2][k] = t2;
            B[j+3][k] = t3;

            B[j][k+4] = t4;
            B[j+1][k+4] = t5;
            B[j+2][k+4] = t6;
            B[j+3][k+4] = t7;
          }

          for(k=j;k<j+4;k++)
          {
            t0 = A[i+4][k];
            t1 = A[i+5][k];
            t2 = A[i+6][k];
            t3 = A[i+7][k];

                t4 = B[k][i+4];
                t5 = B[k][i+5];
                t6 = B[k][i+6];
                t7 = B[k][i+7];

                B[k][i+4] = t0;
                B[k][i+5] = t1;
                B[k][i+6] = t2;
                B[k][i+7] = t3;

                B[k+4][i] = t4;
                B[k+4][i+1] = t5;
                B[k+4][i+2] = t6;
```

```
                    B[k+4][i+3] = t7;
              }

        for (k=i+4;k<i+8;k++)
              {
                t0 = A[k][j+4];
                t1 = A[k][j+5];
                t2 = A[k][j+6];
                t3 = A[k][j+7];

                B[j+4][k] = t0;
                B[j+5][k] = t1;
                B[j+6][k] = t2;
                B[j+7][k] = t3;
              }
          }
      }
```

61 × 67

```
int i,j,k,t0,t1,t2,t3,t4,t5,t6,t7;//11
    for(j=0; j<M-M%8; j+=8){
        for(i=0; i<N-N%8; i++)
        {
                t0 = A[i][j];
                t1 = A[i][j+1];
                t2 = A[i][j+2];
                t3 = A[i][j+3];
                t4 = A[i][j+4];
                t5 = A[i][j+5];
                t6 = A[i][j+6];
                t7 = A[i][j+7];

                B[j][i] = t0;
                B[j+1][i] = t1;
                B[j+2][i] = t2;
                B[j+3][i] = t3;
                B[j+4][i] = t4;
                B[j+5][i] = t5;
                B[j+6][i] = t6;
                B[j+7][i] = t7;

        }
        }
        for(i=0;i<N;i++)
           for(j=M-M%8;j<M;j++)
           {
             k=A[i][j];
```

```
                B[j][i]=k;
            }
        for(i=N-N%8;i<N;i++)
            for(j=0;j<M;j++)
            {
                k=A[i][j];
                B[j][i]=k;
            }
```

2.2.3 Evaluation

After several attempts, we finally got the optimal result as follows:

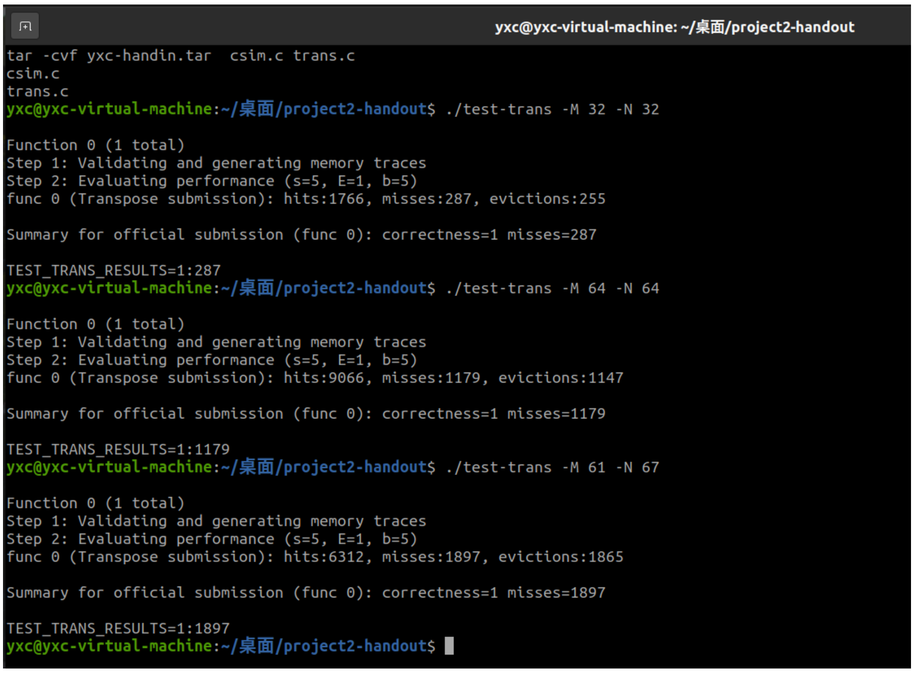


Figure 4: The Result of Part B

3 Conclusion

3.1 Problems

In Project 2 **Understanding Cache Memories**, we encounter many difficulties and problems and solve them in the end, and we list the problems below:

- **Part A:**
  1. Without the guidance of experimental documentation, it is difficult for us to use the *getopt* function to parse my command line arguments.
  2. It takes some effort to further understand the cache and construct the code structure.
- **Part B:**

1. It is difficult to notice that reading 8 numbers at once is the most efficient way to use the cache.
2. It is hard to understand why we could only access elements in 4 rows at a time in **64** $\times$ **64** matrix.
3. In the process of matrix transposition, the conflict misses about the diagonal is difficult to find and notice.

3.2 Achievements

The total evaluation of two parts is shown below, using **driver.py**, which shows our simulator and the optimized function both work well.

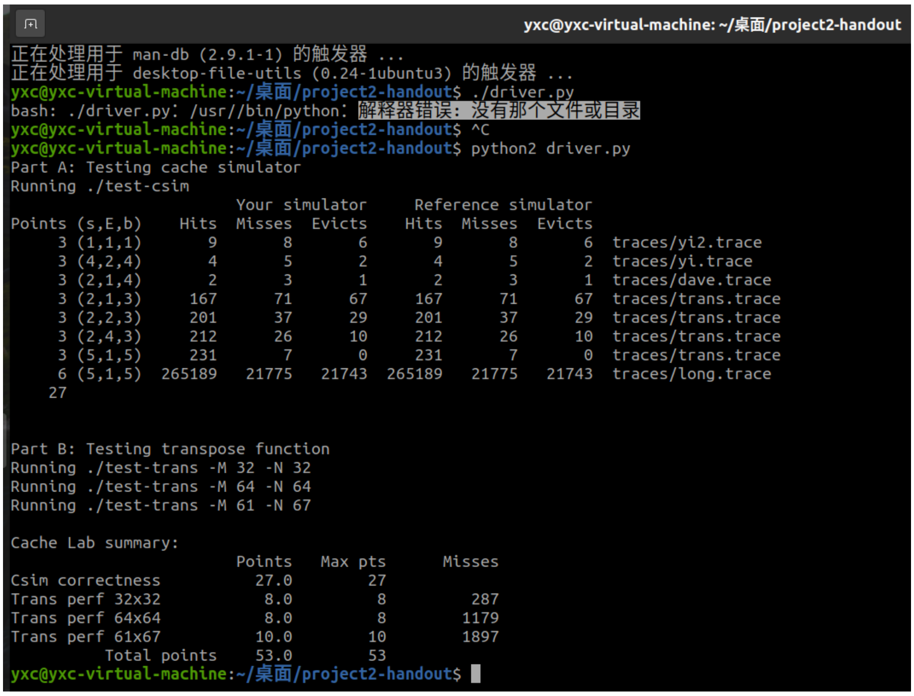


Figure 5: The Evaluation for Whole Project

4 Acknowledgements

- Thanks for the teacher of CS2305, Prof.Yanyan Shen to teach us the basic knowledge of the cache and give us the chance to do this interesting project.
- Thanks for Bryant and other writers of CS:APP, who provide the material about Blocking and the basic design of the challenging lab.
- Thanks to the previous learners of CS:APP for the discussion and analysis of this project, which guided me and inspired me to move forward step by step in this project.