# Project 2: Understanding Cache Memories

521021910107, Chenzhi Hu, ether-wind@sjtu.edu.cn

May 28, 2023

## 1 Introduction

In this project, I wrote some simple programs to help me learn about cache memories. The project can be devided into two parts. In part A, I wrote a cache simulator in `csim.c` that takes a valgrind memory trace as input, simulates the hit/miss behavior of a cache memory on this trace, and outputs the total number of hits, misses, and evictions. In part B, I wrote a transpose function in `trans.c` that causes as few cache misses as possible.

## 2 Experiments

### 2.1 Part A

#### 2.1.1 Analysis

In this part, there is no need to simulate the entire behavior of a cache memory. Actually, only the proccess of accessing the cache memory needs to be simulated.

The key is that how to implement the LRU replacement policy. It is very natural to create a new array to record the time since the pages were last visited, but it will cause a waste or time as we have to update the whole array every time a line in the cache is visited.

Another way to implement the LRU replacement policy is use a global variable to record the global time. Every time we visit a line in the cache, we only need to update the timestamp of this line with the global time.

This method is also very convenient in another aspect. If a line in a cache is never visited, its timestamp must be 0. Also, any cache with a timestamp greater than 0 must has been visited. So the valid bit of the lines can be merged with their timestamps. If we want to knwo wether a line is valid, we only need to chech its timestamps.

With the timestamps, the LRU replacement policy can be implement in the folloing way. When we want to replace a line in the cache, what we need to do is checking the timestamps of all lines in the group and finding out the line with the biggest timestamp, which is the line last recently used in the group.

### 2.1.2 Code

The code of `csim.c` is shown in Code Listing 1.

```
#include "cachelab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <strings.h>
#include <unistd.h>
#include <getopt.h>

int displayTrace = 0;
int indexBits;
int setNum;
int associativity;
int offsetBits;
int blockNum;
char *tracefile;
FILE *file;

int hits = 0, misses = 0, evictions = 0;

typedef struct {
    int time;
    uint64_t tag;
} cache_line;

int globalTime = 0;
cache_line **cache;


void usage(char *argv[]);
void init_cache();
void find_data(uint64_t tag, int index, char *result);
void destroy();

/* Print usage information */
void usage(char *argv[]) {
    printf("Usage: %s [-hv] -s <num> -E <num> -b <num> -t <file>\n"
           "Options:\n"
           "  -h         Print this help message.\n"
           "  -v         Optional verbose flag.\n"
           "  -s <num>   Number of set index bits.\n"
           "  -E <num>   Number of lines per set.\n"
           "  -b <num>   Number of block offset bits.\n"
           "  -t <file>  Trace file.\n"
           "\n"
           "Examples:\n"
           "  linux>  %s -s 4 -E 1 -b 4 -t traces/yi.trace\n"
           "  linux>  %s -v -s 8 -E 2 -b 4 -t traces/yi.trace\n", argv
               [0], argv[0], argv[0]);
}

/* Initiate the cache */
void init_cache() {
```

```
53      cache = (cache_line **) malloc(sizeof(cache_line *) * setNum);
54      for (int i = 0; i < setNum; ++i) {
55      cache[i] = (cache_line *) malloc(sizeof(cache_line) * associativity
            );
56      memset(cache[i], 0, sizeof(cache_line) * associativity);
57      }
58  }
59
60
61  /* Destroy the cache */
62  void destroy() {
63      for (int i = 0; i < setNum; ++i) {
64      free(cache[i]);
65      }
66      free(cache);
67  }
68
69  /* Find data in the cache and store hit/miss information into char *
        result */
70  void find_data(uint64_t tag, int index, char *result) {
71      cache_line *group = cache[index];
72      int emptyLine = -1;
73      for (int i = 0; i < associativity; ++i) {
74      if (!group[i].time) {
75          emptyLine = i;
76      }
77      else if (group[i].tag == tag) {
78          group[i].time = globalTime;
79          hits++;
80          strcat(result, " hit");
81          return;
82      }
83      }
84
85      strcat(result, " miss");
86      misses++;
87      // If there is an empty line in the group
88      if (emptyLine >= 0) {
89      group[emptyLine].tag = tag;
90      group[emptyLine].time = globalTime;
91      }
92      // If there is not an empty line in the group, we need to replace
            one of the lines
93      else {
94      strcat(result, " eviction");
95      evictions++;
96      int toReplace = 0;
97      for (int i = 1; i < associativity; ++i) {
98          if (group[i].time < group[toReplace].time)
99          toReplace = i;
100     }
101     group[toReplace].tag = tag;
102     group[toReplace].time = globalTime;
103     }
104 }
105
106 int main(int argc, char *argv[]) {
```

```
107     int opt;
108     opterr = 0;
109     int s_input = 0, E_input = 0, b_input = 0, t_input = 0;
110     while ((opt = getopt(argc, argv, "hvs:E:b:t:")) != -1) {
111     if (opt == 'h') {
112         usage(argv);
113         return 0;
114     }
115     else if (opt == 'v')
116         displayTrace = 1;
117     else if (opt == 's') {
118         indexBits = atoi(optarg);
119         setNum = 1 << indexBits;
120         s_input = 1;
121     }
122     else if (opt == 'E') {
123         associativity = atoi(optarg);
124         E_input = 1;
125     }
126     else if (opt == 'b') {
127         offsetBits = atoi(optarg);
128         blockNum = 1 << offsetBits;
129         b_input = 1;
130     }
131     else if (opt == 't') {
132         tracefile = (char *) malloc((strlen(optarg) + 1) * sizeof(char)
                ;
133         strcpy(tracefile, optarg);
134         t_input = 1;
135     }
136     else if (opt == '?') {
137         printf("%s: Missing required command line argument\n", argv[0])
                ;
138         usage(argv);
139         return 0;
140     }
141     }
142
143     // If one of the parameters is not defined, reprot error
144     if (!(s_input && E_input && b_input && t_input)) {
145     printf("%s: Missing required command line argument\n", argv[0]);
146     usage(argv);
147     return 0;
148     }
149
150     // Initailize the cache
151     init_cache();
152
153     file = fopen(tracefile, "r");
154     if (!file) {
155     printf("Fail to open %s!\n", tracefile);
156     }
157
158     char op[2];
159     uint64_t address;
160     int size;
161
```

```
162     while (fscanf(file, "%s %lx, %d\n", op, &address, &size) != -1) {
163     // Skip I instruction
164     if (op[0] == 'I')
165         continue;
166     int index = (address >> offsetBits) & ~(~0u << indexBits);
167     uint64_t tag = address >> (indexBits + offsetBits);
168
169     ++globalTime;
170     char result[20] = "";
171     find_data(tag, index, result);
172     // M instruction need to visit the cache twice
173     if (op[0] == 'M') find_data(tag, index, result);
174     if (displayTrace)
175         printf("%s %lx,%d%s\n", op, address, size, result);
176     }
177
178     // Destroy the cache
179     destroy();
180
181     // Print summary
182     printSummary(hits, misses, evictions);
183     return 0;
184 }
```

Code Listing 1: csim.c

### 2.1.3 Evaluation

As what is shown in Figure 1, all of the results of my simulator are the same as the ones of the reference simultator. It demonstrates that my simulator is correct.

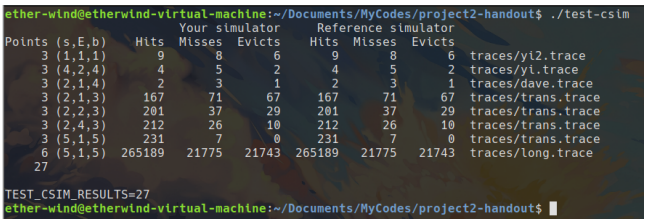


Figure 1: result of part A

## 2.2 Part B

### 2.2.1 Analysis

With twelve available variables, we can use at of them to store the elements in the matrix temperally when copy them into another matrix. This will reduce the number of evictions.

When $M = N = 32$, every eight raws of the matrix use different lines of the cache, and the capcity of the lines is also eight, so it its very natural to

divide the matrix into sixteen $8 \times 8$ blocks and transpose the matrix in blocks. But there is a problem: if the block is on the diagonal, the original method to transpose the matrix will cause extra evictions. To solve this problem, we can store some elements at the same location in B, and then transfer it into the target location when the target location is loaded into the cache.

When $M = N = 64$, it will be more conplicated because in a $8 \times 8$ block, the first four lines use the same lines in the cache with the last four lines. If we still devide the matrix into $8 \times 8$ blocks, there sill be much more evictions. A simple idea is devide the matrix into $4 \times 4$ blocks, but it is also not very efficient. Here is another solution:. First devide the matrix into $8 \times 8$ large blocks, each large block is devided into four $4 \times 4$ small blocks. For each large blodk in A, if we want to transpose it into another large block in B, as what is shown in Figure 2, do the following operations:

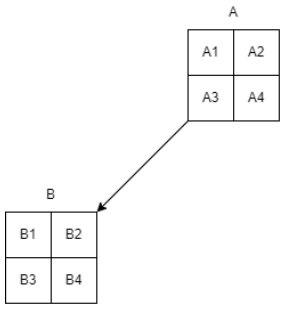


Figure 2: transpose A into B

- $B_1 = A_1^T, B_2 = A_2^T$;
- $B_3 = B_2, B_2 = A_3^T$;
- $B_4 = A_4^T$.

It is important that when copy the elements, we should copy them in lines to avoid evctions.

When $M = 61, N = 67$, it is much simpler than the second situation because there are not so many restrictions. The method in the first situation is OK in this case. The only difference is that there will additional incomplelte blocks to be deal with while it is very easy to realize.

### 2.2.2 Code

The code of `csim.c` is shown in Code Listing 2.

```
1 /*
2 * trans.c - Matrix transpose B = A^T
3 *
```

```
 4 * Each transpose function must have a prototype of the form:
 5 * void trans(int M, int N, int A[N][M], int B[M][N]);
 6 *
 7 * A transpose function is evaluated by counting the number of misses
 8 * on a 1KB direct mapped cache with a block size of 32 bytes.
 9 */
10 #include <stdio.h>
11 #include "cachelab.h"
12
13 int is_transpose(int M, int N, int A[N][M], int B[M][N]);
14
15 /*
16 * transpose_submit - This is the solution transpose function that you
17 *     will be graded on for Part B of the assignment. Do not change
18 *     the description string "Transpose submission", as the driver
19 *     searches for that string to identify the transpose function to
20 *     be graded.
21 */
22 char transpose_submit_desc[] = "Transpose submission";
23 void transpose_submit(int M, int N, int A[N][M], int B[M][N])
24 {
25     if (M == 32) {
26         int i, j, x, y, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8;
27         for (i = 0; i < M; i += 8) {
28             for (j = 0; j < N; j += 8)  {
29
30                 if (i != j) {
31                     for (x = i; x < i + 8; ++x)
32                     for (y = j; y < j + 8; ++y)
33                         B[y][x] = A[x][y];
34                 }
35
36                 else {
37                     tmp1 = A[i][j];
38                     tmp2 = A[i][j + 1];
39                     tmp3 = A[i][j + 2];
40                     tmp4 = A[i][j + 3];
41                     tmp5 = A[i][j + 4];
42                     tmp6 = A[i][j + 5];
43                     tmp7 = A[i][j + 6];
44                     tmp8 = A[i][j + 7];
45
46                     B[i][j] = tmp1;
47                     B[i][j + 1] = tmp2;
48                     B[i][j + 2] = tmp3;
49                     B[i][j + 3] = tmp4;
50                     B[i][j + 4] = tmp5;
51                     B[i][j + 5] = tmp6;
52                     B[i][j + 6] = tmp7;
53                     B[i][j + 7] = tmp8;
54
55                     tmp1 = A[i + 1][j];
56                     tmp2 = A[i + 1][j + 1];
57                     tmp3 = A[i + 1][j + 2];
58                     tmp4 = A[i + 1][j + 3];
59                     tmp5 = A[i + 1][j + 4];
60                     tmp6 = A[i + 1][j + 5];
```

```
61                      tmp7 = A[i + 1][j + 6];
62                      tmp8 = A[i + 1][j + 7];
63
64                      B[i + 1][j] = B[i][j + 1];
65                      B[i][j + 1] = tmp1;
66                      B[i + 1][j + 1] = tmp2;
67                      B[i + 1][j + 2] = tmp3;
68                      B[i + 1][j + 3] = tmp4;
69                      B[i + 1][j + 4] = tmp5;
70                      B[i + 1][j + 5] = tmp6;
71                      B[i + 1][j + 6] = tmp7;
72                      B[i + 1][j + 7] = tmp8;
73
74                      tmp1 = A[i + 2][j];
75                      tmp2 = A[i + 2][j + 1];
76                      tmp3 = A[i + 2][j + 2];
77                      tmp4 = A[i + 2][j + 3];
78                      tmp5 = A[i + 2][j + 4];
79                      tmp6 = A[i + 2][j + 5];
80                      tmp7 = A[i + 2][j + 6];
81                      tmp8 = A[i + 2][j + 7];
82
83                      B[i + 2][j] = B[i][j + 2];
84                      B[i + 2][j + 1] = B[i + 1][j + 2];
85                      B[i][j + 2] = tmp1;
86                      B[i + 1][j + 2] = tmp2;
87                      B[i + 2][j + 2] = tmp3;
88                      B[i + 2][j + 3] = tmp4;
89                      B[i + 2][j + 4] = tmp5;
90                      B[i + 2][j + 5] = tmp6;
91                      B[i + 2][j + 6] = tmp7;
92                      B[i + 2][j + 7] = tmp8;
93
94                      tmp1 = A[i + 3][j];
95                      tmp2 = A[i + 3][j + 1];
96                      tmp3 = A[i + 3][j + 2];
97                      tmp4 = A[i + 3][j + 3];
98                      tmp5 = A[i + 3][j + 4];
99                      tmp6 = A[i + 3][j + 5];
100                     tmp7 = A[i + 3][j + 6];
101                     tmp8 = A[i + 3][j + 7];
102
103                     B[i + 3][j] = B[i][j + 3];
104                     B[i + 3][j + 1] = B[i + 1][j + 3];
105                     B[i + 3][j + 2] = B[i + 2][j + 3];
106                     B[i][j + 3] = tmp1;
107                     B[i + 1][j + 3] = tmp2;
108                     B[i + 2][j + 3] = tmp3;
109                     B[i + 3][j + 3] = tmp4;
110                     B[i + 3][j + 4] = tmp5;
111                     B[i + 3][j + 5] = tmp6;
112                     B[i + 3][j + 6] = tmp7;
113                     B[i + 3][j + 7] = tmp8;
114
115                     tmp1 = A[i + 4][j];
116                     tmp2 = A[i + 4][j + 1];
117                     tmp3 = A[i + 4][j + 2];
```

```
118                     tmp4 = A[i + 4][j + 3];
119                     tmp5 = A[i + 4][j + 4];
120                     tmp6 = A[i + 4][j + 5];
121                     tmp7 = A[i + 4][j + 6];
122                     tmp8 = A[i + 4][j + 7];
123
124                     B[i + 4][j] = B[i][j + 4];
125                     B[i + 4][j + 1] = B[i + 1][j + 4];
126                     B[i + 4][j + 2] = B[i + 2][j + 4];
127                     B[i + 4][j + 3] = B[i + 3][j + 4];
128                     B[i][j + 4] = tmp1;
129                     B[i + 1][j + 4] = tmp2;
130                     B[i + 2][j + 4] = tmp3;
131                     B[i + 3][j + 4] = tmp4;
132                     B[i + 4][j + 4] = tmp5;
133                     B[i + 4][j + 5] = tmp6;
134                     B[i + 4][j + 6] = tmp7;
135                     B[i + 4][j + 7] = tmp8;
136
137                     tmp1 = A[i + 5][j];
138                     tmp2 = A[i + 5][j + 1];
139                     tmp3 = A[i + 5][j + 2];
140                     tmp4 = A[i + 5][j + 3];
141                     tmp5 = A[i + 5][j + 4];
142                     tmp6 = A[i + 5][j + 5];
143                     tmp7 = A[i + 5][j + 6];
144                     tmp8 = A[i + 5][j + 7];
145
146                     B[i + 5][j] = B[i][j + 5];
147                     B[i + 5][j + 1] = B[i + 1][j + 5];
148                     B[i + 5][j + 2] = B[i + 2][j + 5];
149                     B[i + 5][j + 3] = B[i + 3][j + 5];
150                     B[i + 5][j + 4] = B[i + 4][j + 5];
151                     B[i][j + 5] = tmp1;
152                     B[i + 1][j + 5] = tmp2;
153                     B[i + 2][j + 5] = tmp3;
154                     B[i + 3][j + 5] = tmp4;
155                     B[i + 4][j + 5] = tmp5;
156                     B[i + 5][j + 5] = tmp6;
157                     B[i + 5][j + 6] = tmp7;
158                     B[i + 5][j + 7] = tmp8;
159
160                     tmp1 = A[i + 6][j];
161                     tmp2 = A[i + 6][j + 1];
162                     tmp3 = A[i + 6][j + 2];
163                     tmp4 = A[i + 6][j + 3];
164                     tmp5 = A[i + 6][j + 4];
165                     tmp6 = A[i + 6][j + 5];
166                     tmp7 = A[i + 6][j + 6];
167                     tmp8 = A[i + 6][j + 7];
168
169                     B[i + 6][j] = B[i][j + 6];
170                     B[i + 6][j + 1] = B[i + 1][j + 6];
171                     B[i + 6][j + 2] = B[i + 2][j + 6];
172                     B[i + 6][j + 3] = B[i + 3][j + 6];
173                     B[i + 6][j + 4] = B[i + 4][j + 6];
174                     B[i + 6][j + 5] = B[i + 5][j + 6];
```

```
175                     B[i][j + 6] = tmp1;
176                     B[i + 1][j + 6] = tmp2;
177                     B[i + 2][j + 6] = tmp3;
178                     B[i + 3][j + 6] = tmp4;
179                     B[i + 4][j + 6] = tmp5;
180                     B[i + 5][j + 6] = tmp6;
181                     B[i + 6][j + 6] = tmp7;
182                     B[i + 6][j + 7] = tmp8;
183
184                     tmp1 = A[i + 7][j];
185                     tmp2 = A[i + 7][j + 1];
186                     tmp3 = A[i + 7][j + 2];
187                     tmp4 = A[i + 7][j + 3];
188                     tmp5 = A[i + 7][j + 4];
189                     tmp6 = A[i + 7][j + 5];
190                     tmp7 = A[i + 7][j + 6];
191                     tmp8 = A[i + 7][j + 7];
192
193                     B[i + 7][j] = B[i][j + 7];
194                     B[i + 7][j + 1] = B[i + 1][j + 7];
195                     B[i + 7][j + 2] = B[i + 2][j + 7];
196                     B[i + 7][j + 3] = B[i + 3][j + 7];
197                     B[i + 7][j + 4] = B[i + 4][j + 7];
198                     B[i + 7][j + 5] = B[i + 5][j + 7];
199                     B[i + 7][j + 6] = B[i + 6][j + 7];
200                     B[i][j + 7] = tmp1;
201                     B[i + 1][j + 7] = tmp2;
202                     B[i + 2][j + 7] = tmp3;
203                     B[i + 3][j + 7] = tmp4;
204                     B[i + 4][j + 7] = tmp5;
205                     B[i + 5][j + 7] = tmp6;
206                     B[i + 6][j + 7] = tmp7;
207                     B[i + 7][j + 7] = tmp8;
208                 }
209             }
210         }
211     }
212
213     else if (M == 64) {
214         int i, j, x, y, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8;
215         for (i = 0; i < M; i += 8) {
216             for (j = 0; j < N; j += 8) {
217                 for (x = i; x < i + 4; ++x) {
218                     tmp1 = A[x][j];
219                     tmp2 = A[x][j + 1];
220                     tmp3 = A[x][j + 2];
221                     tmp4 = A[x][j + 3];
222                     tmp5 = A[x][j + 4];
223                     tmp6 = A[x][j + 5];
224                     tmp7 = A[x][j + 6];
225                     tmp8 = A[x][j + 7];
226
227                     B[j][x] = tmp1;
228                     B[j + 1][x] = tmp2;
229                     B[j + 2][x] = tmp3;
230                     B[j + 3][x] = tmp4;
231                     B[j][x + 4] = tmp5;
```

```
                    B[j + 1][x + 4] = tmp6;
                    B[j + 2][x + 4] = tmp7;
                    B[j + 3][x + 4] = tmp8;
                }

                for (y = j; y < j + 4; ++y) {
                    tmp1 = A[i + 4][y];
                    tmp2 = A[i + 5][y];
                    tmp3 = A[i + 6][y];
                    tmp4 = A[i + 7][y];
                    tmp5 = B[y][i + 4];
                    tmp6 = B[y][i + 5];
                    tmp7 = B[y][i + 6];
                    tmp8 = B[y][i + 7];

                    B[y][i + 4] = tmp1;
                    B[y][i + 5] = tmp2;
                    B[y][i + 6] = tmp3;
                    B[y][i + 7] = tmp4;
                    B[y + 4][i] = tmp5;
                    B[y + 4][i + 1] = tmp6;
                    B[y + 4][i + 2] = tmp7;
                    B[y + 4][i + 3] = tmp8;
                }

                for (x = i + 4; x < i + 8; ++x) {
                    tmp1 = A[x][j + 4];
                    tmp2 = A[x][j + 5];
                    tmp3 = A[x][j + 6];
                    tmp4 = A[x][j + 7];

                    B[j + 4][x] = tmp1;
                    B[j + 5][x] = tmp2;
                    B[j + 6][x] = tmp3;
                    B[j + 7][x] = tmp4;
                }
            }
        }
    }

    else if (M == 61) {
        int i, j, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8;
        int n = N / 8 * 8;
        int m = M / 8 * 8;
        for (j = 0; j < m; j += 8) {
            for (i = 0; i < n; ++i) {
                tmp1 = A[i][j];
                tmp2 = A[i][j+1];
                tmp3 = A[i][j+2];
                tmp4 = A[i][j+3];
                tmp5 = A[i][j+4];
                tmp6 = A[i][j+5];
                tmp7 = A[i][j+6];
                tmp8 = A[i][j+7];

                B[j][i] = tmp1;
                B[j+1][i] = tmp2;
```

```
289                 B[j+2][i] = tmp3;
290                 B[j+3][i] = tmp4;
291                 B[j+4][i] = tmp5;
292                 B[j+5][i] = tmp6;
293                 B[j+6][i] = tmp7;
294                 B[j+7][i] = tmp8;
295             }
296         }
297         for (i = n; i < N; ++i) {
298             for (j = m; j < M; ++j) {
299                 tmp1 = A[i][j];
300                 B[j][i] = tmp1;
301             }
302         }
303         for (i = 0; i < N; ++i) {
304             for (j = m; j < M; ++j) {
305                 tmp1 = A[i][j];
306                 B[j][i] = tmp1;
307             }
308         }
309         for (i = n; i < N; ++i) {
310             for (j = 0; j < M; ++j) {
311                 tmp1 = A[i][j];
312                 B[j][i] = tmp1;
313             }
314         }
315     }
316 }
317
318 /*
319 * You can define additional transpose functions below. We've defined
320 * a simple one below to help you get started.
321 */
322
323 /*
324 * trans - A simple baseline transpose function, not optimized for the
        cache.
325 */
326 char trans_desc[] = "Simple row-wise scan transpose";
327 void trans(int M, int N, int A[N][M], int B[M][N])
328 {
329     int i, j, tmp;
330
331     for (i = 0; i < N; i++) {
332         for (j = 0; j < M; j++) {
333             tmp = A[i][j];
334             B[j][i] = tmp;
335         }
336     }
337
338 }
339
340 /*
341 * registerFunctions - This function registers your transpose
342 *     functions with the driver.  At runtime, the driver will
343 *     evaluate each of the registered functions and summarize their
344 *     performance. This is a handy way to experiment with different
```

```
345 *     transpose strategies.
346 */
347 void registerFunctions()
348 {
349     /* Register your solution function */
350     registerTransFunction(transpose_submit, transpose_submit_desc);
351 
352     /* Register any additional transpose functions */
353     registerTransFunction(trans, trans_desc);
354 
355 }
356 
357 /*
358 * is_transpose - This helper function checks if B is the transpose of
359 *     A. You can check the correctness of your transpose by calling
360 *     it before returning from the transpose function.
361 */
362 int is_transpose(int M, int N, int A[N][M], int B[M][N])
363 {
364     int i, j;
365 
366     for (i = 0; i < N; i++) {
367         for (j = 0; j < M; ++j) {
368             if (A[i][j] != B[j][i]) {
369             return 0;
370             }
371         }
372     }
373     return 1;
374 }
```

Code Listing 2: trans.c

#### 2.2.3 Evaluation

the detailed results of part B are shown in Figure 3. The brief results are shown in Figure 4, which is implemented by `driver.py`.

As what we can see in the figures, the results of the three tests are all correct and achieve full marks.

# 3 Conclusion

## 3.1 Problems

The biggest problem is that how to reduce evictions in part B, especially when $M = N = 64$. I have tried to use $8 \times 8$ and $4 \times 4$ blocks but the result is not satisfying. And finally I successfully combine the two method, which I have mentioned above.

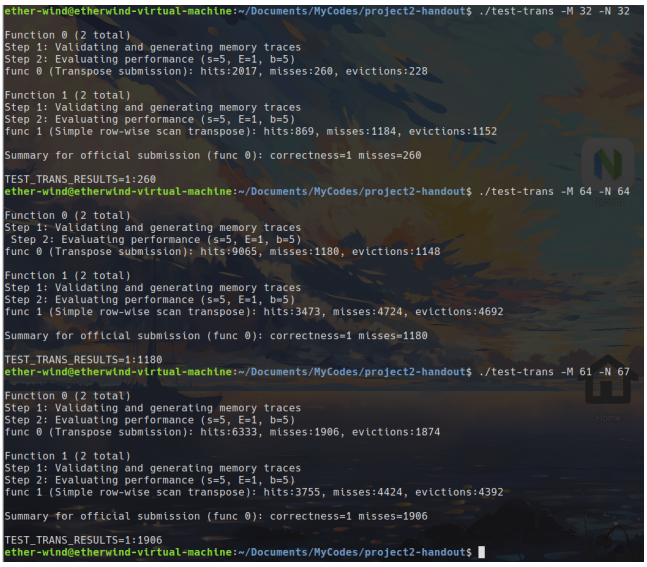


Figure 3: detailed results

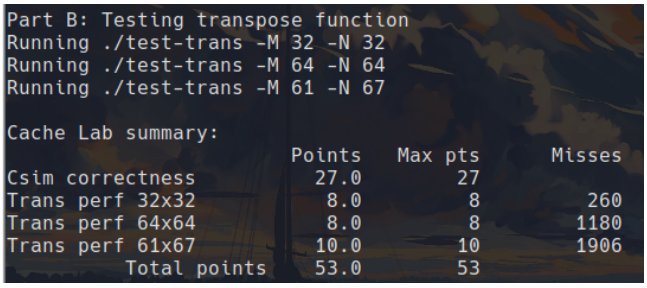


Figure 4: brief results

## 3.2 Achievements

In my sulotion, I take full use of the twelve variables, and use block technique to improvement the performance. I also design different algorithms for different inputs.

To increase the coding readability, I seperate different sections with empty lines, add necessary commentions and keep the code as neat as possible.

In this project, I have a deeper understanding of caching and learn about block technique. I feel that I benefit from this project a lot.