一、 test-trans 以及 trace.f

生成了 trace.f1 文件，并用 csim-ref 观察结果，重定向输出到 trace_f1.txt 文件中

```
ubuntu@ubuntu:~/cachelab-handout$ ./test-trans -M 32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Validation error at function 0! Run ./tracegen -M 32 -N 32 -F 0 for details.
Skipping performance evaluation for this function.

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=0 misses=2147483647

TEST_TRANS_RESULTS=0:2147483647
```

```
 5 L 30a080,4 miss eviction
 6 S 34a080,4 miss eviction
 7 L 30a084,4 miss eviction
 8 S 34a100,4 miss
 9 L 30a088,4 hit
10 S 34a180,4 miss
11 L 30a08c,4 hit
12 S 34a200,4 miss
13 L 30a090,4 hit
14 S 34a280,4 miss
15 L 30a094,4 hit
16 S 34a300,4 miss
17 L 30a098,4 hit
18 S 34a380,4 miss
```

二、 处理本条命令行同时显示标记位和组号

修改显示组号和标记位后:

```
while(fscanf(tracefile,"%s %x,%d",ope,&addr,&size)!=EOF) {
    if (ope[0]=='I')
        continue;
    int setBits,tagBits;
    setBits=getSet(addr,s,b);
    tagBits=getTag(addr,s,b);
    printf("setBits=%d\ttagBits=%d\t",setBits,tagBits);
    if (v==1)
        printf("%s %x,%d ",ope,addr,size);
    if (ope[0]=='L')
        loadData(&cache,setBits,tagBits,v);
    else if (ope[0]=='S')
        storeData(&cache,setBits,tagBits,v);
    else if (ope[0]=='M')
        modifyData(&cache,setBits,tagBits,v);

    //putSets(sim_cache);
    if (v==1)
        printf("\n");
}
```

使用./csim -v 并且重定向输出到 trace_f1_csim.txt 中，结果如下

```
 5 setBits=4   tagBits=3112    L 30a080,4 miss eviction
 6 setBits=4   tagBits=3368    S 34a080,4 miss eviction
 7 setBits=4   tagBits=3112    L 30a084,4 miss eviction
 8 setBits=8   tagBits=3368    S 34a100,4 miss
 9 setBits=4   tagBits=3112    L 30a088,4 hit
10 setBits=12  tagBits=3368    S 34a180,4 miss
11 setBits=4   tagBits=3112    L 30a08c,4 hit
12 setBits=16  tagBits=3368    S 34a200,4 miss
13 setBits=4   tagBits=3112    L 30a090,4 hit
14 setBits=20  tagBits=3368    S 34a280,4 miss
15 setBits=4   tagBits=3112    L 30a094,4 hit
16 setBits=24  tagBits=3368    S 34a300,4 miss
17 setBits=4   tagBits=3112    L 30a098,4 hit
18 setBits=28  tagBits=3368    S 34a380,4 miss
19 setBits=4   tagBits=3112    L 30a09c,4 hit
20 setBits=0   tagBits=3369    S 34a400,4 miss
```

三、 以 M4N4 为例，分析 miss 过多的情况

```
 5 setBits=4   tagBits=3112    L 30a080,4 miss eviction
 6 setBits=4   tagBits=3368    S 34a080,4 miss eviction
 7 setBits=4   tagBits=3112    L 30a084,4 miss eviction
 8 setBits=4   tagBits=3368    S 34a090,4 miss eviction
 9 setBits=4   tagBits=3112    L 30a088,4 miss eviction
10 setBits=5   tagBits=3368    S 34a0a0,4 miss
11 setBits=4   tagBits=3112    L 30a08c,4 hit
12 setBits=5   tagBits=3368    S 34a0b0,4 hit
13 setBits=4   tagBits=3112    L 30a090,4 hit
14 setBits=4   tagBits=3368    S 34a084,4 miss eviction
15 setBits=4   tagBits=3112    L 30a094,4 miss eviction
16 setBits=4   tagBits=3368    S 34a094,4 miss eviction
17 setBits=4   tagBits=3112    L 30a098,4 miss eviction
18 setBits=5   tagBits=3368    S 34a0a4,4 hit
19 setBits=4   tagBits=3112    L 30a09c,4 hit
20 setBits=5   tagBits=3368    S 34a0b4,4 hit
21 setBits=5   tagBits=3112    L 30a0a0,4 miss eviction
22 setBits=4   tagBits=3368    S 34a088,4 miss eviction
23 setBits=5   tagBits=3112    L 30a0a4,4 hit
24 setBits=4   tagBits=3368    S 34a098,4 hit
25 setBits=5   tagBits=3112    L 30a0a8,4 hit
26 setBits=5   tagBits=3368    S 34a0a8,4 miss eviction
27 setBits=5   tagBits=3112    L 30a0ac,4 miss eviction
28 setBits=5   tagBits=3368    S 34a0b8,4 miss eviction
29 setBits=5   tagBits=3112    L 30a0b0,4 miss eviction
30 setBits=4   tagBits=3368    S 34a08c,4 hit
31 setBits=5   tagBits=3112    L 30a0b4,4 hit
32 setBits=4   tagBits=3368    S 34a09c,4 hit
33 setBits=5   tagBits=3112    L 30a0b8,4 hit
34 setBits=5   tagBits=3368    S 34a0ac,4 miss eviction
35 setBits=5   tagBits=3112    L 30a0bc,4 miss eviction
36 setBits=5   tagBits=3368    S 34a0bc,4 miss eviction
```

对应的 16 个 LS 操作，b=5 所以 cache 有 32 字节，可以存放 8 个 int 型，所有下标[0][0]到[1][3]的映射到一个组，其余的映射到另一组。

从第 5 行开始:

对 A[0][0]进行 L 操作，由于之前有初始化，标记位不匹配，发生 miss eviction
对 B[0][0]进行 S 操作，由于之前的 L 对 4 组进行操作，标记位不匹配，发生 miss eviction
对 A[0][1]进行 L 操作，由于之前 S 也对 4 组操作，标记位不匹配，发生 miss eviction
对 B[1][0]进行 S 操作，由于之前的 L 对 4 组进行操作，标记位不匹配，发生 miss eviction
对 A[0][2]进行 L 操作，由于之前 S 也对 4 组操作，标记位不匹配，发生 miss eviction
对 B[2][0]进行 S 操作，由于此时的组号为 5，发生一次 cold miss
对 A[0][3]进行 L 操作，组号为 4，由于标记位匹配，hit
对 B[3][0]进行 S 操作，由于此时的组号为 5，标记位匹配，发生 hit
对 A[1][0]进行 L 操作，组号为 4，由于标记位匹配，hit
对 B[0][1]进行 S 操作，由于上条 L 对 4 组进行操作，标记位不匹配，发生 miss eviction
对 A[1][1]进行 L 操作，由于之前 S 也对 4 组操作，标记位不匹配，发生 miss eviction
对 B[1][1]进行 S 操作，由于上条 L 对 4 组进行操作，标记位不匹配，发生 miss eviction
对 A[1][2]进行 L 操作，由于之前 S 也对 4 组操作，标记位不匹配，发生 miss eviction
对 B[2][1]进行 S 操作，由于此时的组号为 5，标记位匹配，发生 hit
对 A[1][3]进行 L 操作，组号为 4，由于标记位匹配，hit
对 B[3][1]进行 S 操作，由于此时的组号为 5，标记位匹配，发生 hit
对 A[2][0]进行 L 操作，由于之前 S 也对 5 组操作，标记位不匹配，发生 miss eviction
其余部分与上述相似。由于对 A 数组是顺序遍历，而 B 则先列后行，所以每次操作中有四次会映射到同一组，发生 conflict miss，而之后则映射到不同的组，发生 hit

四、 以 M32N32 为例分析 miss 过多的原因

	0	1	2	3	4	5	6	7	8
0	8	8	8	8	8	8	8	8	9
1	12	12	12	12	12	12	12	12	13
2	16							16	17
3	20							20	21
4	24							24	25
5	28							28	29
6	0							0	1
7	4	4	4	4	4	4	4	4	5
8	8							8	9

```
 5 setBits=4   tagBits=3112    L 30a080,4 miss eviction
 6 setBits=4   tagBits=3368    S 34a080,4 miss eviction
 7 setBits=4   tagBits=3112    L 30a084,4 miss eviction
 8 setBits=8   tagBits=3368    S 34a100,4 miss
 9 setBits=4   tagBits=3112    L 30a088,4 hit
10 setBits=12  tagBits=3368    S 34a180,4 miss
11 setBits=4   tagBits=3112    L 30a08c,4 hit
12 setBits=16  tagBits=3368    S 34a200,4 miss
13 setBits=4   tagBits=3112    L 30a090,4 hit
14 setBits=20  tagBits=3368    S 34a280,4 miss
15 setBits=4   tagBits=3112    L 30a094,4 hit
16 setBits=24  tagBits=3368    S 34a300,4 miss
17 setBits=4   tagBits=3112    L 30a098,4 hit
18 setBits=28  tagBits=3368    S 34a380,4 miss
19 setBits=4   tagBits=3112    L 30a09c,4 hit
20 setBits=0   tagBits=3369    S 34a400,4 miss
21 setBits=5   tagBits=3112    L 30a0a0,4 miss
22 setBits=4   tagBits=3369    S 34a480,4 miss eviction
```

由于 A 是先行后列而 B 先列后行，则在 A[0][0]和 B[0][0]时会发生 miss eviction，接下来对 A 来说，到 A[0][9]由于都对应于 4 组，而 B 则对应于 4,8,12,16,20,24,28,0 的这样一个循环，所以每次都会发生一次 cold miss。接下来 A 则对应 5 组，发生一次 cold miss，而 B 重复上述的循环由于与之前的组存放的是 A，由于标记位不同，又会发生 miss eviction，导致 miss 过多。所以 A 按行产生 miss 是由于 cold miss 且只有一次，而 B 按列则在第一轮 cold miss 后标记位不同，所以后续产生 miss eviction。

五、 编写 transpose_submit()

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i,j,i1,j1;
    for (i=0;i<N;i+=4) {
        for (j=0;j<M;j+=4){
            int im=i+4;
            for (i1=i;i1<im;i1++) {
                int jm=j+4;
                for (j1=j;j1<jm;j1++){
                    int temp=A[i1][j1];
                    B[j1][i1]=temp;
                }
            }
        }
    }
}
```

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    int i,j,i1,j1;
    for (i=0;i<N;i+=8) {
        for (j=0;j<M;j+=8){
            int im=i+8;
            for (i1=i;i1<im;i1++) {
                int jm=j+8;
                for (j1=j;j1<jm;j1++){
                    int temp=A[i1][j1];
                    B[j1][i1]=temp;
                }
            }
        }
    }
}
```

```
Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1566, misses:487, evictions:455
```

按 4 分块的结果

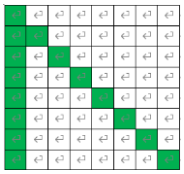
```
Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1710, misses:343, evictions:311
```

按 8 分块的结果

```
Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255
```

进行对角线优化后的结果

数组 B 第一个 8*8 块的情况，绿色为 miss，白色为 hit



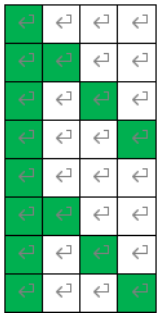
六、 分析采用分块后 miss 改善原因。

按照 8*8 进行分块后，按照不同位置的元素对应的组号来看，相对于对角线对称的两个 8*8 的块，其对应的组号均不同，这就保证了在转置的过程中，将 A 的对应位置元素写入 B 对应位置时，降低了 miss 数，得到了改善。而进行对角线的优化后，将 A 中连续的 8 个元素赋值给临时变量，并且一并写入 B，这样保证了不会 miss。

4x4 分块的结果：

```
Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1614, misses:439, evictions:407
```

A 数组 8*4 命中情况：（绿色为 miss，白色为 hit，下同）（下左图）



B 数组 4*8 命中情况（上右图）：

由于分块后，A 右侧一块和 B 下侧一块对应的组号均不相同，且 B 在第一列过后均位于 cache 内，所有 hit 数增加。