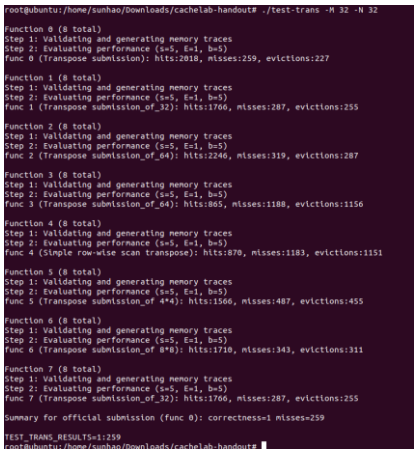
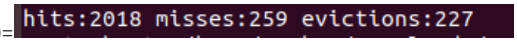
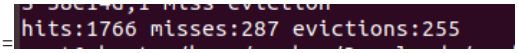
【10%】实验test-trans，以M32N32矩阵为例，并通过csim-ref详细选项（-v）在缓存跟踪trace.f文件中观察结果。

以 m=32,n=32 矩阵为例子：

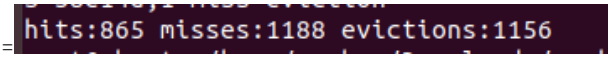


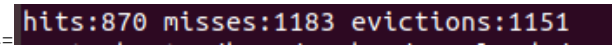
通过 csim-ref（-v）跟踪 trace.f 文件中结果：

F0= 

F1= 

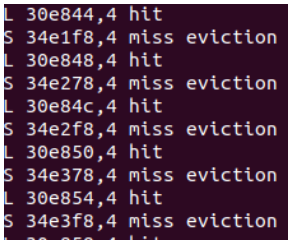
F2= 

F3= 

F4= 

：【10%】编写csim.c处理本条命令行同时输出显示标记位和组号。

编写 csim.c 结果实例：



从图中看到输出了组号和标志位

【20%】实验test-trans以M4N4矩阵为例，并分析示例函数miss过多原因。利用csim模拟器详细模式输出结果；利用L和S地址来定位16个L和S操作；记录各命令行对应的数组A和B元素标号（如A[0][1]）；结合组号、标记位和缓存操作的结果(hit或miss eviction)分析conflict miss产生原因。注意：数组B和A相同的下标元素是映射到缓存中同一个组。

实例函数结果：

```
Function 4 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 4 (Simple row-wise scan transpose): hits:15, misses:22, evictions:19
```

实例函数中 miss 较多是因为：在访问 A 数组（按行访问），访问 B 数组（按列访问）的过程中存在很多 conflict miss

```
S 0038e14c,1
L 0038e1e0,8
L 0038e144,4
L 0038e140,4
L 0030e140,4
S 0034e140,4
L 0030e144,4
S 0034e150,4
L 0030e148,4
S 0034e160,4
L 0030e14c,4
S 0034e170,4
L 0030e150,4
S 0034e144,4
L 0030e154,4
S 0034e154,4
L 0030e158,4
S 0034e164,4
L 0030e15c,4
S 0034e174,4
L 0030e160,4
S 0034e148,4
L 0030e164,4
S 0034e158,4
L 0030e168,4
S 0034e168,4
L 0030e16c,4
S 0034e178,4
L 0030e170,4
S 0034e14c,4
L 0030e174,4
S 0034e15c,4
L 0030e178,4
S 0034e16c,4
L 0030e17c,4
S 0034e17c,4
S 0038e14d,1
```

L,S 地址定位以及对应数组 A 和 B 元素的标号

利用 csim 模拟器更详细的输出结果：

```
L 30e844,4 hit
S 34e1f8,4 miss eviction
L 30e848,4 hit
S 34e278,4 miss eviction
L 30e84c,4 hit
S 34e2f8,4 miss eviction
L 30e850,4 hit
S 34e378,4 miss eviction
L 30e854,4 hit
S 34e3f8,4 miss eviction
```

图

产生 conflict miss 原因：

1. A 数组访问 A[0][0],不命中，将块 11 载入 cache
2. B 数组访问 B[0][0],虽然该映射块 11 在 cache 中，但标记位不同，不[illegible] B 对应的块 11 载入 cache 中
3. A 数组访问 A[0][1]，虽然该映射块 11 在 cache 中，但标记位不同，不命中，重新将 A 对应的块 11 载入 cache 中
4. B 数组访问 B[1][0]，虽然该映射块 11 在 cache 中，但标记位不同，不命中，重新将 A 对应的块 11 载入 cache 中
5. A 数组访问 A[0][2]，虽然该映射块 11 在 cache 中，但标记位不同，不命中，重新将 A 对应的块 11 载入 cache 中
6. B 数组访问 B[2][0],B[2][0]所映射的块 12 不在 cache 中，不命中，将对应的块 12 载入 cache
7. A 数组访问 A[0][3],A[0][3]所映射的块 11 在 cache 中，标记位相同，命中
8. B 数组访问 B[3][0],B[3][0]所映射的块 12 在 cache 中，标记位相同，命中
9. A 数组访问 A[1][0],A[1][0]所映射的块 11 在 cache 中，标记位相同，命中
10. B 数组访问 B[0][1],虽然 B[0][1]所映射的块 11 在 cache 中，但标记位不同，不命中，将块 11 载入 cache
11. 载入 cache

    可以以此类推剩下的过程

    综上可以看出两个数组存在太多的冲突不命中，造成此现象的原因是因为数组 A，B 中下标相同的会映射到同一个 cache 块，这样会不断造成冲突不命中。

【20%】实验test-trans以M32N32矩阵为例，并分析示例函数miss过多原因。跟踪结果输出在文件上（>****.txt选项）；结合trace中一部分命令行（如A[0][0]B[0][0]~ A[0][9]B[9][0]）组号、标记位和缓存操作的结果（hit或miss eviction）分析数组A按行访问miss产生原因和数组B按列访问miss产生原因。

实例函数：

```
Function 4 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 4 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151
```

Miss 过多的原因，由于对角线的元素转置后仍在同一个区域，造成大量冲突不命中

输出跟踪结果：

```
S 38e14c,1 miss
L 38e160,8 miss
L 38e144,4 hit
L 38e140,4 hit
L 30e140,4 miss eviction
L 30e144,4 hit
L 30e148,4 hit
```

对于数组 A 来说每个块中只有一个元素不被命中

图

对于数组 B 来说每个块中有一个元素不被命中，同时还有对角线的元素也不会被命中。

图

【20%】分别按4和8分块编写transpose_submit()代码，记录test-trans以 M32N32矩阵下miss数目结果；进一步编写代码重新处理相同下标的对角线上元素来再次优化，列表记录数组B第一个8*8块写操作的命中情况。

编写 transpose_submit()

```
char transpose_submit_1_desc[]="Transpose submission_of 4*4";
void transpose_submit_1(int M,int N,int A[N][M],int B[M][N]){
        int i,j,x,y;
//      int x1=0,x2=0,x3=0,x4=0,x5=0,x6=0,x7=0,x8=0;
        for(i=0;i<N;i+=4){
        for(j=0;j<M;j+=4){
                        for(x=i;x<N&&x<i+4;x++){
                                for(y=j;y<M&&y<j+4;y++){
                                        B[y][x]=A[x][y];
                                }
                        }
        }
    }
}
```

```
char transpose_submit_2_desc[]="Transpose submission_of 8*8";
void transpose_submit_2(int M,int N,int A[N][M],int B[M][N]){
        int i,j,x,y;
//      int x1=0,x2=0,x3=0,x4=0,x5=0,x6=0,x7=0,x8=0;
        for(i=0;i<N;i+=8){
        for(j=0;j<M;j+=8){
                        for(x=i;x<N&&x<i+8;x++){
                                for(y=j;y<M&&y<j+8;y++){
                                        B[y][x]=A[x][y];
                                }
                        }
        }
    }
}
```

M=32，N=32 Miss 结果

```
Function 5 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 5 (Transpose submission_of 4*4): hits:1566, misses:487, evictions:455

Function 6 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 6 (Transpose submission_of 8*8): hits:1710, misses:343, evictions:311
```

优化处理同一下标下对角线上的元素

优化结果

```
Function 7 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 7 (Transpose submission_of_32): hits:1766, misses:287, evictions:255
```

【20%】分析采用分块技术后miss改善原因。按4分块编写transpose_submit()代码，记录test-trans以 M32N32矩阵下miss数目结果；以两个4*4为例分析hit增多原因。注意：加载8*4 的矩阵A，存储到4*8的矩阵B。

4 块 test_trans 在 m=32,n=32 矩阵 miss 数目

```
Function 5 (8 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 5 (Transpose submission_of 4*4): hits:1566, misses:487, evictions:455
```

以两个 4x4 为例：每块有 8 个整型，每八行充满缓存，所以八块可以将不需要的替换降到最少，冲突也会减少 。4 块也可以减少。不分块时，A 数组按行访问，B 按列。当缓存满时，就会进行替换。导致不命中。当分为两个四块时访问 B【1】【2】在之前就已经存入缓存，所以会命中，同理会减少 miss.