实验日志 2.7

任务1. 实验 test-trans 以 M64N64 矩阵为例，编写代码优化trans.c 中转置函数，采用8分块处理过程和记录 miss 结果。

由运行结果可以知道直接 8*8 分块，miss 数改变很小。原因可能为数组规模扩充为64*64，cache 块大小是 32 字节，组数为 32，则 cache 存满需要 32*32/4=256 个整型，所以 256/64=4 行就会存满 cache 说明每四行数组中的数据就会重复映射到已经映射过的组，8*8 分块仍然出现 y 方向写 cache 时的驱逐，命中率为 0。

所以考虑将内部分化为 4*4 的小块。如果直接内部 4*4 分块，无法最大化提高命中率， 因为一次读取 A[i][j]~A[i][j+7]，但是只使用 A[i][j]~A[i][j+3]，后四个数据将会在 B 访问时候被驱逐，为了避免此种情况，可以在处理 A 的第一个分块时，将 A 的第二个 4*4 分块转置后暂存在 B 的第二块上，然后在处理 A 的第二个块的时候将 B 的第二块平移到 B 的第三块，从而避免冲突。

优化代码如下：

```
void transpose_submit7(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, k;
    int a0, a1, a2, a3, a4, a5, a6, a7;
    for (i = 0; i < N; i += 8)
        for (j = 0; j < M; j += 8)
        {
            for (k = i; k < i + 4; k++)
            { //转置第一个块，将A第二个块转置到B的第二个块
                a0 = A[k][j];a1 = A[k][j + 1];a2 = A[k][j + 2];a3 = A[k][j + 3];
                a4 = A[k][j + 4];a5 = A[k][j + 5];a6 = A[k][j + 6];a7 = A[k][j + 7];
                B[j][k] = a0;B[j + 1][k] = a1;B[j + 2][k] = a2;B[j + 3][k] = a3;
                B[j][k + 4] = a4;B[j + 1][k + 4] = a5;B[j + 2][k + 4] = a6;B[j + 3][k + 4] = a7;
            }
            for (k = j; k < j + 4; k++)
            { //将第三个块转置到B的第二个块，B的第二个块平移到第三块
                a0 = A[i + 4][k];a1 = A[i + 5][k];a2 = A[i + 6][k];a3 = A[i + 7][k];
                a4 = B[k][i + 4];a5 = B[k][i + 5];a6 = B[k][i + 6];a7 = B[k][i + 7];
                B[k][i + 4] = a0;B[k][i + 5] = a1;B[k][i + 6] = a2;B[k][i + 7] = a3;
                B[k + 4][i] = a4;B[k + 4][i + 1] = a5;B[k + 4][i + 2] = a6;B[k + 4][i + 3] = a7;
            }
            for (k = i + 4; k < i + 8; k++)
            { //处理最后一块，将A的第四块转置到B的第四块
                a0 = A[k][j + 4];a1 = A[k][j + 5];a2 = A[k][j + 6];a3 = A[k][j + 7];
                B[j + 4][k] = a0;B[j + 5][k] = a1;B[j + 6][k] = a2;B[j + 7][k] = a3;
            }
        }
}
```

运行结果：miss 次数降到了 1179 次

```
lh@lh-VirtualBox:~/cachelab-handout$ ./test-trans -M 64 -N 64

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:9066, misses:1179, evictions:1147

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 tranpose): hits:3474, misses:4723, evictions:4691
```

任务2. 追踪分析 M64N64 的组索引，分析8分块优化处理过程

```
lh@lh-VirtualBox:~/cachelab-handout$ ./csim -v -s 5 -E 1 -b 5 -t trace.f1>test.txt
```

将 8*8 分为 4*4 的块的过程中具体分为 3 步：

1） 将 A 左上方 4*4 分块转置到 B 的相同位置。读取 A 时，第一列将会冷不命中。 将 A 右上方 4*4 分块转置到 B 的相同位置。处理左上方时已经缓存在 cache 中，都命中。

2） 将 A 左下方 4*4 分块转置到 B 的右上方。此时 A 的第一列将会发生冷不命中，B 的该 4*4 分块会被驱逐，但除了第一列，数据都会缓存，因此该行后续的访问，将会命中。在处理完 A 左下方的某一列的同时，将 B 的右上方的块平移到 B 的左下方，这时访问 cache 会因为缓存右上方的第一行被占据，所以处理 B 的左下方的第一列都会发生驱逐不命中，其余部分则会命中。

3） 将 A 右下方 4*4 分块转置到 B 的相同位置。访问 A 时，在前一次处理中已经缓存在 cache 中，命中；写 B

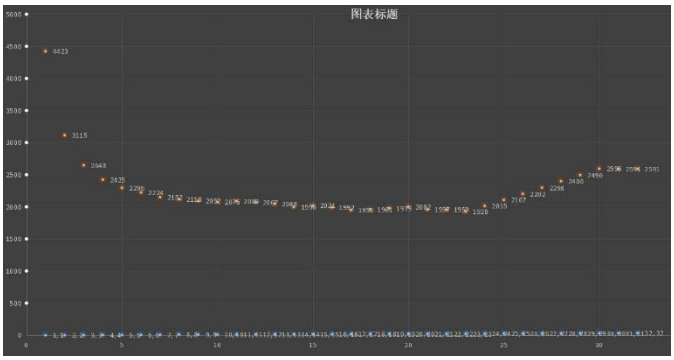
时，在平移右上方到左下方时已经缓存，命中。

```
set:12 ,tag:1076L 10d180,4 miss eviction
set:12 ,tag:1076L 10d184,4 hit
set:12 ,tag:1076L 10d188,4 hit
set:12 ,tag:1076L 10d18c,4 hit
set:12 ,tag:1076L 10d190,4 hit
set:12 ,tag:1076L 10d194,4 hit
set:12 ,tag:1076L 10d198,4 hit
set:12 ,tag:1076L 10d19c,4 hit
5209 set:16 ,tag:1332L 14d210,4 hit
5210 set:16 ,tag:1332L 14d214,4 hit set:8 ,tag:1336S 14e100,4 miss eviction
5211 set:16 ,tag:1332L 14d218,4 hit set:16 ,tag:1336S 14e200,4 miss eviction
5212 set:16 ,tag:1332L 14d21c,4 hit set:24 ,tag:1336S 14e300,4 miss eviction
5213 set:16 ,tag:1332S 14d210,4 hit set:0 ,tag:1337S 14e400,4 miss eviction
5214 set:16 ,tag:1332S 14d214,4 hit set:8 ,tag:1336S 14e110,4 hit
5215 set:16 ,tag:1332S 14d218,4 hit set:16 ,tag:1336S 14e210,4 hit
5216 set:16 ,tag:1332S 14d21c,4 hit set:24 ,tag:1336S 14e310,4 hit
                                   set:0 ,tag:1337S 14e410,4 hit
```

结论：按照 8*8 分块内部 4*4 分块并且利用暂存的方式处理，只会在读取 A 和写 B 第一次取这一块的数据发生不命中，和在对角线上的元素发生不命中。

任务3. 编写代码优化 M61N67 的trans.c，尝试更多的分块并记录 miss 数目减少结果，选取其中较好方案

由上图可以得出，最佳分块点是 N=23，其分块处理代码如右图。




```
char test23des[] = "test23";
void test23(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, k, r;
    for (i = 0; i < N; i += 23)
        for (j = 0; j < M; j += 23)
            for (k = i; k < N && k < i + 23; k++)
                for (r = j; r < M && r < j + 23; r++)
                    B[r][k] = A[k][r];
}
```

实验报告 2.2

一、整理实验信息

1、**实验目标：**

学会cache的存储原理，利用Part A编写cache模拟器进一步学习miss，hit与eviction的原理，利用part B理解C语言程序对于cache存储的影响与实际的cache优化方法，根据不同规模，合理利用分块技术，减少不命中率，寻找最佳的分块方法，理解cache运行过程。

2、**实验资源：**windows 10专业版，ubuntu 64 位，VMware

3、**实验步骤：**

1）安装实验环境：sudo apt-get install valgrind 安装 valgrind。

2)PartA：自制模拟器

编写一个cache模拟器csim.c，该模拟器可以模拟数据读（L）写（S）中cache的命中、不命中与牺牲行的情况，需要牺牲行时，用LRU算法进行替换并实现相关LRU，set，tag信息显示及cache运行过程。

3)PartB

写4x4,8x8分块以及对角线优化实现32x32，64x64矩阵转置的函数使函数调用过程中对cache的不命中数miss尽可能少。包括对于61x67矩阵分块的分析。

关键命令：

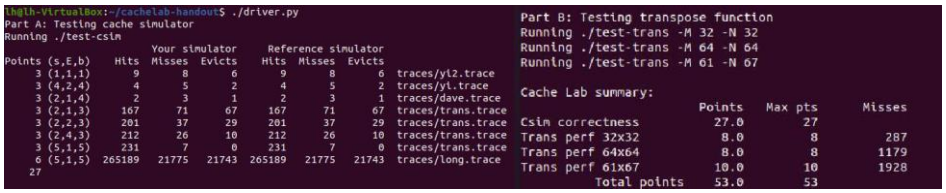
①./csim-ref -v/h -s(组索引位数) -E (每组行数)-b (偏移宽度)-t +trace (被追踪文件名):统计 miss、hit、evction 的具体情况

②./test-csim：展示参考模拟器与自己编写的模拟器运行对比情况

③./test-trans -M t -N t:生成 t *t 大小的矩阵作为测试样例（参数-s 5 -E 1 -s 5），测试 trans.c 文件中自己写的转置优化函数的 hit 、miss、eviction 情况

④ ./csim -v -s 5 -E 1 -b 5 -t trace.f1>tracef1.txt:让模拟器追踪 PartB 生成的 trac 文件

二、实验结果



```
lh@lh-VirtualBox:~/cachelab-handout$ ./driver.py
Part A: Testing cache simulator
Running ./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

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         287
Trans perf 64x64           8.0         8        1179
Trans perf 61x67          10.0        10        1928
                Total points      53.0        53
```

Part A 是./test-csim 的结果，模拟器需解析 trace.c 文件中的指令并且正确执行，得到相应 8 组hit，miss，eviction 数据，这 5 组数据会和参考模拟器运行结果的数据对比，如果数据相同则表示 5 组 trace 样例测试通过。由上述结果可以看出结果正确。

Part B 是 3 组矩阵规模 32*32，64*64，61*67 的测试结果，生成相应的 3 组 miss 数据， 只有在 miss 数据小于基准值时才可能的到满分。由运行结果截图可看出 partB 正确。

最终得分是 53，实验正确。

三、实验总结

1、学习的新知识：

由于实验比课堂早接触cache，相当于为课堂学习打下了基础。cache模拟器的设计加深了我们对cache运行情况的理解。通过实验中对缓存操作的模拟，了解了cache的读取、miss、hit、eviction与LRU算法。这将对以后的理论课有很大的帮助。part B的实现让我了解到如何利用cache来提高C语言代码运行速度，这对于以后的编程也有很大的帮助，使程序cache亲和性更好，更适应现代处理器特色。我对计算机的底层运行状态有了更好的了解，也让我明白了作为一个程序员了解计算机底层运行状态的重要性。

2、学习方法：

本次实验需要学习大量的新知识，因此，这个实验需要我们仔细阅读，广泛查阅。除了自己查资料，和同学讨论也是很重要的。实验中参考了许多资料学习cache知识，因为一开始课堂上并没有讲过相应知识。

3、技能：

本次实验获得的主要技能是实现对于同一个目的代码的优化，这避免了很多看起来很好但实际上不起作用的优化。逐步想办法更进一步减少miss次数。同时，根据课程提示实现了cache模拟器，所以其实很多测试程序、模拟程序都可以作为辅助工具帮助理解或者是debug主要部分程序，这些工具都可以自行设计。

4、人文生活：

当遇到困难时，我们不应该退缩。同时应加强同学间的交流，对cache理解不同，所得到的结果不同，但是经过深入讨论后我和同学还是得到了统一的结论，这对于实验的顺利进行起到了莫大的帮助。