日志 2.7

【10%】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 分块处理过程记录和记录 miss 结果。注意，在前一次 trans.c 基础上添加代码。

```
for(int x = i; x < i+4; x++)
{
tmp0 = A[x][j];      tmp1 = A[x][j+1];
tmp2 = A[x][j+2];    tmp3 = A[x][j+3];
tmp4 = A[x][j+4];    tmp5 = A[x][j+5];
tmp6 = A[x][j+6];    tmp7 = A[x][j+7];
B[j][x] = tmp0;      B[j+1][x] = tmp1;
B[j+2][x] = tmp2;    B[j+3][x] = tmp3;
B[j+4][x] = tmp4;    B[j+5][x] = tmp5;
B[j+6][x] = tmp6;    B[j+7][x] = tmp7;
}
```

A 数组上方的一半，将其按行进行转置，但是写入数组 B 的时 候，右上角 4*4 的一块先放在 B 数组的右上角一块，这样没有写入数组 B 的后四行，没有造成冲突

```
for(int y = j;y < j+4; y++)
{
tmp0 = A[i+4][y];      tmp1 = A[i+5][y];
tmp2 = A[i+6][y];      tmp3 = A[i+7][y];
tmp4 = B[y][i+4];      tmp5 = B[y][i+5];
tmp6 = B[y][i+6];      tmp7 = B[y][i+7];
B[y][i+4] = tmp0;      B[y][i+5] = tmp1;
B[y][i+6] = tmp2;      B[y][i+7] = tmp3;
B[y+4][i] = tmp4;      B[y+4][i+1] = tmp5;
B[y+4][i+2] = tmp6;    B[y+4][i+3] = tmp7;
}
```

A 数组左下角的 4*4 的一块，将其按列进行转置。因为我们有 8 个临时变量，可以同时存储 A 数组将要转置的一列 4 个元素，已经 B 数组占 了位置需要转移的 4 个元素。由此减小冲突

```
for(int y = j+4;y < j+8; y=y+2)
{
tmp0 = A[i+4][y];      tmp1 = A[i+5][y];
tmp2 = A[i+6][y];      tmp3 = A[i+7][y];
tmp4 = A[i+4][y+1];    tmp5 = A[i+5][y+1];
tmp6 = A[i+6][y+1];    tmp7 = A[i+7][y+1];
B[y][i+4] = tmp0;      B[y][i+5] = tmp1;
B[y][i+6] = tmp2;      B[y][i+7] = tmp3;
B[y+1][i+4] = tmp4;    B[y+1][i+5] = tmp5;
B[y+1][i+6] = tmp6;    B[y+1][i+7] = tmp7;
}
```

A 数组右下角的 4*4 的一块。将其按列进行转置。这个就正常转 置即可。不会冲突。

Miss 结果：

```
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
```

Miss 数 1183 达到要求。

【20%】追踪分析 M64N64 的组索引，分析 8 分块优化处理过程。

修改 csim 代码使得其可以输出组索引和标记位,输入指令./csim -v -s 5 -E 1 b 5 -t trace.f0 > f0.txt。得到文件后进行分析：

1. 第一部分转置时，A 的每一行读取第一个会冷不命中，其他的都命中，B 写入 时，只有第一次冷不命中，其余都命中

2. 第二部分转置时，A 的列读第一次冷不命中，其他都命中，B 写入时，由于先写入的地址还在块中，因此命中，开新地址时，第一次冷不命中，其余命中

3. 第三部分转置时，全部命中。

【20%】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，选取其中较好的方案。注意：在前一次 trans.c 基础上添加代码。

17 分块的代码如下：

```
for(i=0;i<N;i=i+17)
{
  for(j=0;j<M;j=j+17)
   {
    for(x=i;((x<N)&&(x<i+17));x++)
      {
        for(y=j;((y<M)&&(y<j+17));y++)
        {
          B[y][x]=A[x][y];
        }
    }
   }
}
```

17 分块的结果如下：

```
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:6197, misses:1985, evictions:1953
```

实验报告 2.2

【20%】整理实验信息。回顾个人日志 2.4~2.7，描述实验目标、资源、实验步骤(含关键命令行及说明)。

实验目标：了解 cache 对 C 语言程序产生的影响。具体目标为两个模块：第一个模块是完成一个缓存模拟器，了解计算机中缓 存的是如何命中，执行如何策略进行驱逐等操作；第二个模块是编写一个矩阵转 置函数，针对 cache 性能进行优化，了解了如何编写出 cache 友好的程序。

实验资源：实验配套的实验指导书与 ppt 指引了整个实验。

模块 1 中提供了 csim-ref 示例文件，作为标准指引了缓存模拟器的制作。 同时通过了老师的课程引导完成了函数的构建。 模块 2 中提供了 test-trans 文件，评估了自己编写的程序是否正确，以及打印了 miss 数目。同时将 C 语言代码转换为缓存模拟器可以识别的文件，方便逐步分析

实验步骤：

（1） 安装了 valgrind，搭建实验环境；打开 cachelab，找到可运行的文件并输入命令，尝试不同参数；利用 valgrind 输入，打印出最终的 hit（命中）、miss（不命中）、evictions（驱逐数）。运行缓存模型-v -s 4 -E 1 -b 4：编写 parta 中 csim 文件的部分函数，使自己的模拟器可呈现部分结果，初始化及帮助信息。

（2） 编写模拟器计算部分的函数，实现更新 lru 计数值，更新 cache；指令如下：

./csim - v - s 1 - E 2 - b 4 - t traces/yi.trace

观察每一行的有效值、标记位、lru，查看更新。

编写加载数据 l，存储数据 s，修改数据 m 命令的函数，并添加函数，输出组索引和标记位，检查 lru 主函数代码并与参考的 csim-ref 的结果对比，结果相同则正确。

（3） 通过 csim-ref 跟踪文件 trace.f 文件（设置 M32N32）,命令行./test-trans -M 32 -N 32，生成跟踪文件 trace.f1；使用 csim 跟踪 trace.f1，命令行./csim -v -s 5 -E 1 -b 5 -t trace.f1，观察结果。分别以 4*4，32*32 阵为例分析参数变化原因。并对 4 分块，8 分块的代码进行对角线优化，最终提高命中率。

（4） 以 64*64,61*67 矩阵为例，分析结果，并找到最佳优化分块方式。

【15%】实验结果。检验(driver. py) PartA 和 PartB 两部分参照评分标准的结果。解释实验现象截图:如何看出实验正确?

```
                        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
```

设置 s，E，b 为不同参数，对文件进行跟踪，对比我的模拟器数据与参考模拟器的数据，判断 hits，misses，evicts 结果是否正确。

```
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        1243
Trans perf 61x67          10.0        10        1985
          Total points    53.0        53
```

测试 32*32,64*64，61*67 的评分情况，均为最高分，结果显示都有优化效果。

【15%】实验总结。在实验室做实验过程中，你获取的课堂外的新知识、学习方法、技能、人文生活等方面。

（1） 获取的课堂外的新知识

在这次实验中，对 cache 有了更加深入的了解，对于 cache 的存储方式有了清晰的认识，在自己编写模拟器时，对于 cache 的运行方式有所了解，对相关学习和接下来的实验内容有很大的帮助，测试了不同的参数后，验证实验正确就可以证明模拟器的基本功能满足实验要求。

（2） 学习方法

提高了网上查阅资料的能力

（3） 技能

对时间效率的提升很有帮助，之后在自己写代码的时候可以很好的避免超时痛苦

（4） 人文生活

和同学们的交流更加密切，一起讨论，一起解决问题还是很快乐的，思维的碰撞更能提高学习激情。