

**计算机系统结构实验报告**

|  |
| --- |
| 姓 名：龙际全 |
| 学 院：计算机科学与技术 |
| 专 业：计算机科学与技术 |
| 班 级：CS1603 |
| 学 号：U201614577 |
| 指导教师：陈俭喜 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2019 年. 04月. 25日

**目 录**

[1. 第一部分：Cache模拟器 3](#_Toc1520169817)

[1.1. 实验目的 3](#_Toc2104302945)

[1.2. 实验环境 3](#_Toc1730388283)

[1.3. 实验思路 3](#_Toc1230209455)

[1.4. 实验结果和分析 5](#_Toc293452155)

[2. 第二部分：矩阵转置优化 6](#_Toc1520347594)

[2.1. 实验目的 6](#_Toc1365761196)

[2.2. 实验环境 6](#_Toc140807328)

[2.3. 实验思路 6](#_Toc569702920)

[2.4. 实验结果和分析 7](#_Toc886648394)

[3. 总结和体会 9](#_Toc482958952)

[4. 对实验课程的改进建议 9](#_Toc1825414192)

# 第一部分：Cache模拟器

* 1. **实验目的**

1. 理解cache工作原理；
2. 加深Cache缓存组成结构对C程序性能的影响的理解。
   1. **实验环境**
3. 操作系统：Manjaro 64 bit；
4. 编译器：GCC 8.2.1；
5. IDE：Visual Studio Code。
   1. **实验思路**
6. 内存地址采用组相联映射，一个内存地址具有如下图 1-1所示形式，set\_index用于确定内存地址映射到的组，tag用于在组里用全相联的方式寻找所在的cache行，block\_offset为行内偏移地址。

深度截图_选择区域_20190426004240

图 1-1

1. 该实验只是模拟cache工作原理中的一小部分(Hit/Miss/Eviction)，因此我们可以看到实验框架对cacheline进行了非常简单的抽象，如下图 1-2所示：

深度截图_选择区域_20190426005533

图 1-2

1. Cache替换采用lru算法，lru即最近最不经常使用(least recently used)。检查cache某一组内所有cacheline的tag是否与输入的tag相等，如果相等且cacheline的valid位为1则表示cache访问命中，此时lru计数器清零，相反，对于不命中的cacheline，lru计数器加1；如果没有命中的cacheline，即cache不命中，则需要把主存块调入cache set(cache组)中，先去找cache空闲即valid == 0的那些cacheline；如果没有空闲的cacheline，则此时需要对cache set进行替换，替换策略是将lru计数器最大的cacheline替换掉。
2. 具体到本实验，cache为二维结构如下图 1-3所示，先通过address和set\_index定位到特定的cacheset，然后用循环遍历cacheset里所有的cacheline即可判断是否命中并对lru计数器作相应处理，对于Miss的两种情况：1，有空闲，直接将tag写入空闲cacheline并置valid位为1；2，没有空闲，将tag写入lru最大的cacheline并置lru为0即可。

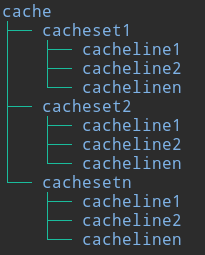


图 1-3

1. 此外，还需要编写replayTrace对输入指令进行相应的处理，对于Load/Store指令，只需要访问主存一次，因此只需要调用一次accessData；由于实验仅关心数据Cache的性能，因此模拟器应忽略所有指令cache访问；Modify指令需要访问两次主存，因此需要调用两次accessData。
   1. **实验结果和分析**
2. 单例./csim(-ref) -v -s 4 -E 1 -b 4 -t traces/yi.trace运行结果如下图 1-4：

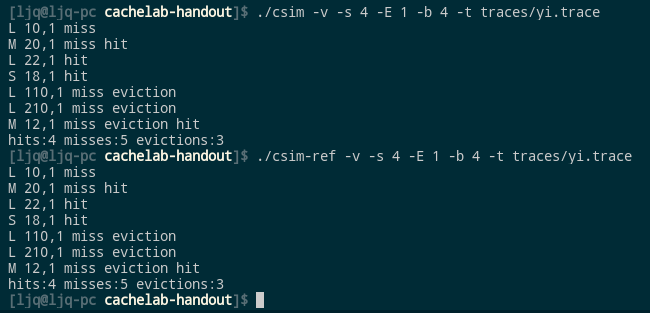


图 1-4

1. 全部样例./test-csim运行结果如下图 1-5：

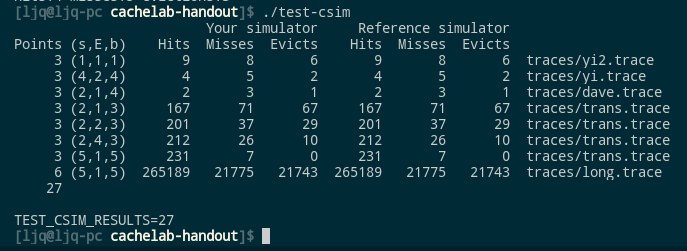


图 1-5

1. 命中数Hits、不命中数Misses、淘汰数Evicts均与标答吻合。

# 第二部分：矩阵转置优化

* 1. **实验目的**

1. 加深对cache工作环境的理解；
2. 能针对系统cache结构对矩阵转置程序进行优化，使得程序运行过程中cache不命中数较少以此提高程序性能。
   1. **实验环境**
3. 操作系统：Manjaro 64 bit；
4. 编译器：GCC 8.2.1；
5. IDE：Visual Studio Code;
6. 软件包：valgrind。
   1. **实验思路**
7. 矩阵转置的常规思路是用两层for循环，将一行直接转为一列，这种方法实现简单，代码量小，在不考虑性能的情况下是最优的。不过，要考虑cache命中率的话，这种方法的性能就非常差了，因为行遍历能够很好地利用cache，但是列访问不能，所以要采取一种折中的方法，使得行和列的大小都不太大，从而减少cache的缺失。很容易想到对矩阵进行分割。具体分成多大一块要根据矩阵的大小来决定，不妨先设为8\*8，因为一个cache行的大小为8个字节。不过，cache总共只有8行，行访问要占用1行，如果8个列访问各占一行，总共会有9行，必有一行被淘汰，为了防止淘汰，要对对角线上的元素另行处理，用变量保存。
8. 用两个for循环，将大矩阵分割成8\*8的小矩阵，对于每个小矩阵，再用一个for循环，将A矩阵的行转到B矩阵的列。每次循环都要判断是否为对角线上的元素，若是则用局部变量存起来，循环结束后再写入B。部分代码如下：

for(int k = 0;k < M ;k+=8) {

for(int l =0 ;l < N; l+=8) {

for(int i=k ; i<k+8;i++) {

for(int j=l;j<l+8;j++) {

if(i!=j) {

B[j][i] = A[i][j];

} else {

tmp = A[i][j];

index = i;

}

}

if(l == k) {

B[index][index] = tmp;

}

}

}

}

* 1. **实验结果和分析**

1. 运行./test-trans -M 8 -N 8结果如下图 2-1所示：

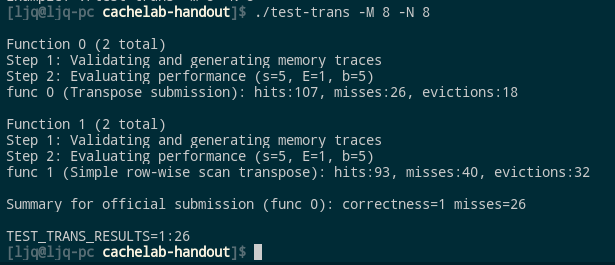


图 2-1

1. 运行./driver.py结果如下图 2-2所示：

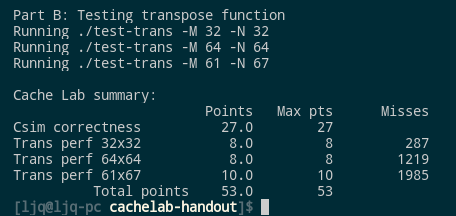


图 2-2

1. 优化后的程序满足实验要求。

# 总结和体会

1. 这次实验进一步加深了我对cache的理解，我之前在组成原理课程中分别用verilog和logisim实现过cache，但这次实验仍然有小小的翻车，因为具体电路比较是否命中是全部并行比较tag的，而在代码中为串行循环比较，我在cache命中后break跳出了循环导致后面没有命中的cacheline的lru计数器没有发生改变，导致最终淘汰算法出错。
2. 这次实验让我了解到底层原理和实现对上层应用软件和算法的重要性，一个好的算法能很大程度提升软件性能。

# 对实验课程的改进建议

1. 注意到实验选自CSAPP(深入理解计算机系统)第三次cache实验，希望可以给出完整的实验文档和实验链接。
2. 实验2矩阵转置优化可以计算cache理论上的hit和miss数，可以将这个作为任务之一。