***2018***



**计算机系统结构 课程实验报告**

|  |  |
| --- | --- |
| 题 目： | Cache实验 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1508 |
| 学 号： | U201514699 |
| 姓 名： | 张鑫 |
| 电 话： | 15927282852 |
| 邮 件： | zhangxinhust@qq.com |
| 完成日期： | 2018-04-23 周一下午 |

目 录

[1 课程实验概述 1](#_Toc512162180)

[1.1 实验目的 1](#_Toc512162181)

[1.2 设计任务 1](#_Toc512162182)

[1.3 设计要求 1](#_Toc512162183)

[1.4 技术指标 2](#_Toc512162184)

[2 总体方案设计 4](#_Toc512162185)

[2.1 Cache模拟器 4](#_Toc512162186)

[2.2 矩阵转置 4](#_Toc512162187)

[3 详细设计与实现 5](#_Toc512162188)

[3.1 Cache模拟器 5](#_Toc512162189)

[3.2 矩阵转置 6](#_Toc512162190)

[4 实验过程与调试 7](#_Toc512162191)

[4.1 功能测试和性能分析 7](#_Toc512162192)

[4.2 主要故障与调试 7](#_Toc512162193)

[5 实验总结与心得 10](#_Toc512162194)

[5.1 实验总结 10](#_Toc512162195)

[5.2 实验心得 10](#_Toc512162196)

[参考文献 11](#_Toc512162197)

# 课程实验概述

## 实验目的

计算机系统结构是计算机专业的核心基础课。课程实验综合利用所学的理论知识，模拟一个简单的cache系统，理解cache工作原理，并加深Cache缓存组成结构对C程序性能的影响的理解。

## 设计任务

### cache模拟器

在csim.c提供的程序框架中，编写一个200-300行的C程序来模拟Cache缓存的行为：

1. 输入：内存访问轨迹；
2. 操作：模拟缓存相对内存访问轨迹的命中/缺失行为；
3. 输出：命中、缺失和（缓存行）淘汰/驱逐的总数。

### 矩阵转置

1. 在参考Cache实现的基础上，优化一个矩阵转置函数，以最小化缓存不命中（cache miss）的数量。
2. 在trans.c中编写实现一个矩阵转置函数transpose\_submit，要求其在参考Cache模拟器csim-ref上运行时对不同大小的矩阵均能最小化缓存缺失的数量

## 设计要求

### cache模拟器

1. 完成的csim.c文件应能接受与参考缓存模拟器csim-ref相同的命令行参数并产生一致的输出结果。

命令行格式：csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>

1. 模拟器必须在输入参数s、E、b设置为任意值时均能正确工作——即需要使用malloc函数（而不是代码中固定大小的值）来为模拟器中数据结构分配存储空间。
2. 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（即轨迹中“I”起始的行）
3. 假设内存访问的地址总是正确对齐的，即一次内存访问从不跨越块的边界——因此可忽略访问轨迹中给出的访问请求大小
4. main函数最后必须调用printSummary函数输出结果，并如下传之以命中hit、缺失miss和淘汰/驱逐eviction的总数作为参数：printSummary(hit\_count, miss\_count, eviction\_count)。

### 矩阵转置

1. 限制对栈的引用——在转置函数中最多定义和使用12个int类型的局部变量，同时不能使用任何long类型的变量或其他位模式数据以在一个变量中存储多个值。

原因：实验测试代码不能/不应计数栈的引用访问，而应将注意力集中在对源和目的矩阵的访问模式上

1. 不允许使用递归。如果定义和调用辅助函数，在任意时刻，从转置函数的栈帧到辅助函数的栈帧之间最多可以同时存在12个局部变量。

例如，如果转置函数定义了8个局部变量，其中调用了一个使用4个局部变量的函数，而其进一步调用了一个使用2个局部变量的函数，则栈上总共将有14个变量，则违反了本规则。

1. 转置函数不允许改变矩阵A，但可以任意操作矩阵B。
2. 不允许在代码中定义任何矩阵或使用malloc及其变种。

## 技术指标

### cache模拟器

1. 每一数据装载(L)或存储(S)操作可引发最多1次缓存缺失(miss)

数据修改操作(M)可认为是同一地址上1次装载后跟1次存储，因此可引发2次缓存命中(hit) ，或1次缺失+1次命中外加可能1次淘汰/驱逐(evict)

1. 对每一测试，test-csim从缓存的Hits（命中）/Misses（缺失）/Evicts（淘汰/驱逐）数量三个指标比较了所实现csim模拟器和参考Cache模拟器csim-ref的性能，
2. 计算csim实现获得的分数：每个用例的每一指标1分（最后一个用例2分）——与参考csim-ref模拟器输出指标相同则判为正确，如图 1.1所示。

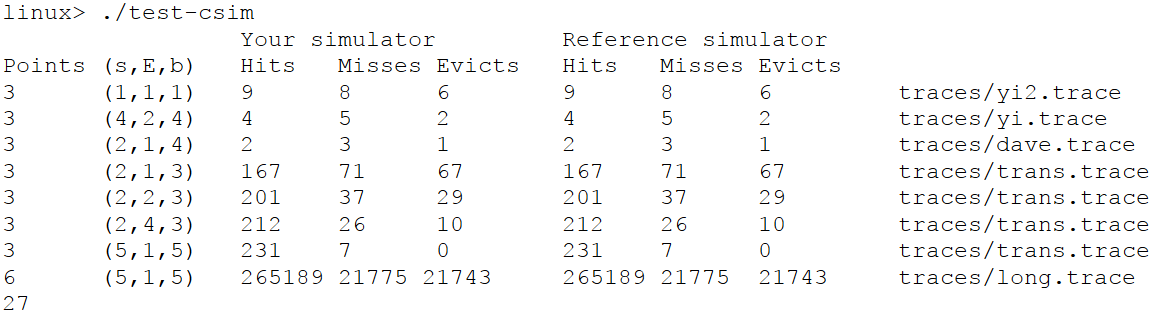


图 1.1 实验一测试结果

### 矩阵转置

1. test-trans程序在三个不同大小的矩阵上测试转置函数的正确性和性能:

32 × 32 (M = 32, N = 32)

64 × 64 (M = 64, N = 64)

61 × 67 (M = 61, N = 67)

1. 针对每一矩阵大小，性能分数线性依赖于发生的Cache缺失总数m：

32×32：如果m<300得8分，如果m>600得0分，对其他m得(600-m)\*8/300分。

64×64：如果m<1300得8分，如果m>2000得0分，对其他m得(2000-m)\*8/700分。

61×67：如果m<2000得10分，如果m>3000得0分，对其他m得(3000-m)\*10/1000分。

# 总体方案设计

## Cache模拟器

csim.c中已经提供了程序框架代码，只需要实现cache的数据访问函数accessData，所以我们也只需要设计这一部分。

在实际的硬件cache中，要判断数据是否命中，使用的是比较器，将多个比较结果用多路选择器选择，得到最终输出。我们要用软件来实现这一过程，并不能像硬件那样并发比较，只能循环比较，用“==”判断符代替比较器，得到我们想要的结果。所以程序的总体结构是多个for循环，对应硬件cache的多个多路选择器。

这里采用三种相联方式中的组相联方式，替换算法采用lru算法，每次命中时lru值清零，每访问一次则lru值加一。将地址右移一定位数得到tag标志，将tag与所有cache行的tag进行比较，如果相等则表示命中，将该行的lru值置零，命中次数加一。若没有命中，则缺失次数加一，此时需要将缺失的tag写入到cache的某一行中。写入也有两种情况，首先是cache行未被全部占用，也就是存在cache行的valid有效位为0时，直接将新的tag写入该行；若所有的valid都为1，则需要进行替换，替换的原则是，选出所有行中lru值最大的一行。替换完成后，替换次数也要加一。

## 矩阵转置

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。

# 详细设计与实现

## Cache模拟器

先用一个for循环遍历cache的所有行，判断tag值是否与cache行的tag值相等，如果相等则命中次数加一，将标志变量flag置为1。值得注意的是，cache行未被使用时，该行的tag也有初值，该初值刚好等于tag的情况也可能存在，所以要在判断相等的同时还保证有效位为1。在每次循环时都将所在行的lru值加一，这是每次访问都要做的，可以在任意一个循环中实现。循环结束后判断flag变量是否为1，若不是，则表示缺失，还需要进行下一步的替换工作。

替换也分两步，先要循环所有的cache行，判断是否仍有未使用的行，直接将新的tag写入，若没有空的行，则进行lru算法的替换，循环找出lru值最大的行，进行替换。这两个循环可以合二为一。

void accessData(mem\_addr\_t addr)

{

int i;

mem\_addr\_t set\_index = (addr >> b) & set\_index\_mask;

mem\_addr\_t tag = addr >> (s+b);

cache\_set\_t cache\_set = cache[set\_index];

int flag1 = 0;

for(i=0; i<E;++i)

{

cache\_set[i].lru++;

if(flag1) continue;

if(cache\_set[i].tag == tag && cache\_set[i].valid == 1)

{

cache\_set[i].lru = 0;

hit\_count++;

flag1 = 1;

}

}

if(flag1==0)

{

miss\_count++;

int max\_index=0,max\_num = cache\_set[0].lru;

for(i=0;i<E;++i)

{

if(cache\_set[i].valid == 0)

{

cache\_set[i].tag = tag;

cache\_set[i].lru = 0;

cache\_set[i].valid = 1;

break;

}

if(max\_num < cache\_set[i].lru)

{

max\_num = cache\_set[i].lru;

max\_index = i;

}

}

if(i>=E)

{

cache\_set[max\_index].tag = tag;

cache\_set[max\_index].lru = 0;

eviction\_count++;

}

}

}

## 矩阵转置

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;

}

}

}

}

# 实验过程与调试

## 功能测试和性能分析

用driver.py测试程序一次性测试两个实验程序，结果如图 4.1所示，实验一得分27，实验二得分26，总分53。

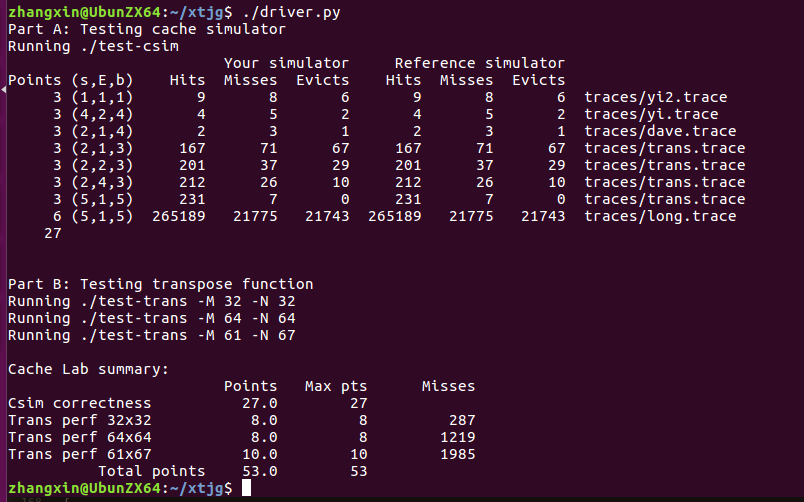


图 4.1 测试结果

## 主要故障与调试

### cache模拟器故障

**故障现象：**编译并测试csim程序，测试结果如图 4.1所示，通过了6个测试用例，但有两个测试用例没有通过。

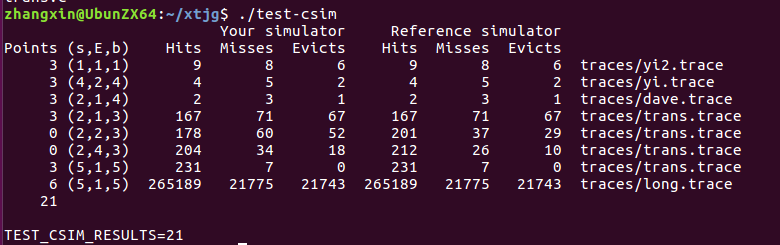


图 4.2 cache模拟器测试结果

**原因分析：**程序的逻辑有一些问题，大部分的测试用例都没有涵盖这一情况，所以能够通过多数用例，需要单独运行没有通过的两个测试文件，打印出每次命中或者缺失的行坐标，部分结果如图 4.2所示，将每一步的结果在纸上记录下，当执行到需要替换处，发现替换的行数不对，这就是问题所在，所以可以肯定是lru算法出了问题。

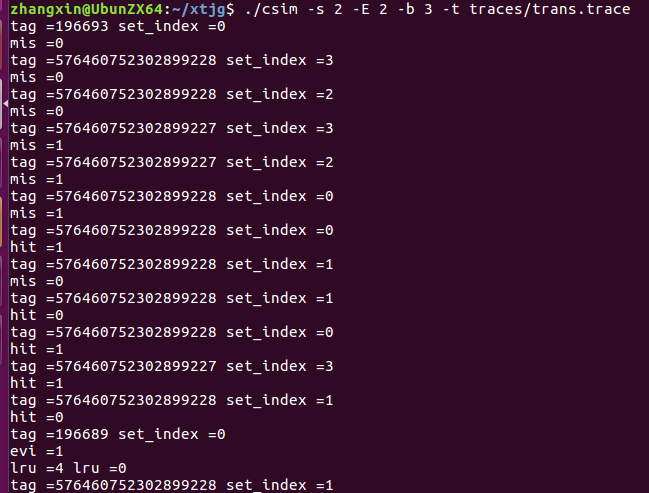


图 4.3 cache模拟器调试

**解决方案：**观察lru替换算法的代码，如图 4.3所示，比较cache行的lru值时，误将lru写成了tag，所以最终淘汰的是tag最大的行，而非lru值最大的行，所以发生了错误，将代码修改过来即可。

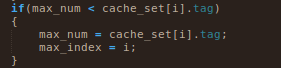


图 4.4 lru算法故障

### 矩阵转置故障

**故障现象：**对矩阵转置程序进行测试，32\*32和61\*67矩阵均可得到满分，而64\*64矩阵的缺失次数超出很多，使用8\*8矩阵进行分割时，缺失次数达到4635，改用其他参数逐一测试，取4\*4时缺失最少，但是仍有1763，如图 4.5所示，这与实验要求的1300以下还有差距。

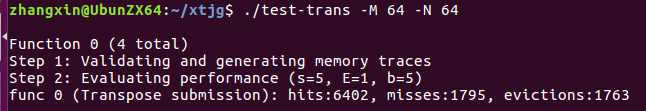


图 4.5 矩阵转置故障

**原因分析：**由于4\*4矩阵已经是最优解，所以这一思路不能继续进行优化，必须用其他方法。

**解决方案：**查阅资料得知，64\*64矩阵要进行进一步的划分，不能直接分成8\*8或者4\*4，要先分成8\*8，再分成4个4\*4：

1. 先考虑把A的上半部分存入到B，但是为了考虑Cache不冲突，所以把右上角的4×4的区域也存在B的右上角。对于在对角线上的块，A的miss率是1/8，B的左上角部分miss率是1/2。对于不在对角线上的块，A的miss率还是1/8，B左上角部分的miss率为1/4.
2. 接下来这步是减少miss率的关键，把A左下角的一列4个数据读出，B右上角的一行4个数据读出，都用int变量暂存，然后把前四个填入B右上角行中，后四个填入B的左下角行中。因为从B右上角读取的时候，把块放入了Cache，然后从A往B中填的时候，就不会出现miss操作。

# 实验总结与心得

## 实验总结

本次实验主要作了如下几点工作：

1. 实现了一个简单的cache模拟器，在已学理论知识的基础上，进一步加深了对于cache工作原理的理解。
2. 对矩阵转置函数进行了优化，懂得了如何巧妙地利用cache来对代码性能进行提升。

## 实验心得

1. 软件cache的效率是很低的，每访问一次都需要用到多个for循环，不能并发执行，由于实验只是要模拟这一过程，所以不考虑效率，但在实际使用过程中应该关注效率问题，用最快的方法判断是否命中。
2. 矩阵转置函数花费了较多的时间进行优化，其中64\*64矩阵的优化，如果完全不查资料，自己的确想不到优化的办法。优化以后的执行效率的确得到了很大的提升。不过值得思考的是，在今后的编程中，我们值不值得花费很多时间去考虑怎样利用cache性能，对代码进行优化，因为这样会使得代码变得复杂，可读性也下降，而且在不同的计算机上，cache大小不同，优化的策略也要不同，很难找到普适的方法。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
5. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| C:\Users\ASUS\Documents\Tencent Files\2452663915\FileRecv\MobileFile\IMG_20180405_230149.jpg本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |