***2018***



**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 题 目： | 计算机系统结构实验报告 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1503 |
| 学 号： | U201514559 |
| 姓 名： | 周铭昊 |
| 电 话： | 15802740273 |
| 邮 件： | 630212894@qq.com |
| 完成日期： | 2018-05-03 周四晚上 |

目 录

[1 实验概述 2](#_Toc513180342)

[1.1 实验目的 2](#_Toc513180343)

[1.2 实验内容 2](#_Toc513180344)

[1.3 实验环境 2](#_Toc513180345)

[1.4 实验要求 2](#_Toc513180346)

[2 实验方案设计 5](#_Toc513180347)

[2.1 编写Cache模拟器 5](#_Toc513180348)

[2.2 优化矩阵转置操作 7](#_Toc513180349)

[3 实验过程与调试 8](#_Toc513180350)

[3.1 实验一测试 8](#_Toc513180351)

[3.2 实验二测试 8](#_Toc513180352)

[4 实验总结与心得 10](#_Toc513180354)

[参考文献 12](#_Toc513180357)

# 实验概述

## 实验目的

理解cache工作原理；

加深Cache缓存组成结构对C程序性能的影响的理解。

## 实验内容

第一部分：编写一个200-300行的C程序来模拟Cache缓存的行为；

第二部分：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。

## 实验环境

Linux 64-bit + valgrind软件包（第二个实验需要），C语言

## 实验要求

实验内容一：

* 任务：在csim.c提供的程序框架中，编写实现一个Cache模拟器：
* 输入：内存访问轨迹
* 操作：模拟缓存相对内存访问轨迹的命中/缺失行为
* 输出：命中、缺失和（缓存行）淘汰/驱逐的总数
* 具体要求：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。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。
* 编程要求：
  + 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。
  + 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）
  + 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
  + 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);

实验内容二：

* 任务：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

char transpose\_submit\_desc[] = "Transpose submission";

void transpose\_submit(int M, int N, int A[N][M], int B[M][N]);

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

# 实验方案设计

## 编写Cache模拟器

修改cism.c文件，首先理解每个变量的含义和每个函数的作用,采用LRU算法，用结构体数组来保存cache数据。

### 变量及结构体定义

表2.1 变量及结构体定义

| 变量名 | 说明 |
| --- | --- |
| Cache\_line | Char valid：有效位，标识这块cache是否有效 |
| mem\_addr\_t tag ：标识位，与内存相对应 |
| unsigned long long int lru：LRU计数位 |
| cache\_set\_t | 一组cache，其中包含多行cache\_line |
| cache\_t | 所有的cache，其中包含多组cache\_set\_t |
| s | 组索引位数 |
| S | 组数，等于2的s次幂 |
| b | 块内偏移位 |
| B | 块大小，等于2的b次幂 |
| E | 关联度，每组cache包含的行数 |
| miss\_count | 未命中计数 |
| hit\_count | 命中计数 |
| eviction\_count | LRU替换计数 |
| eviction\_lru | 被替换的cache的lru计数值 |
| eviction\_line | 被替换的cache行号 |
| isHit | 命中标志位 |
| isFull | Cache已满标志位（一组中所有cache都有效） |
| lru\_counter | Lru计数器，每次访存加一 |

### 算法思想

Cache模拟器的整个运行过程是：先读取命令行的内容，然后设定s，E和b的值，调用initial函数进行初始化。

Initial函数用于cache的初始化。先判断S是否有效，若s<0则cache组数为0，输入无效直接退出。然后初始化一个二维数组cache[S][E]，其中包含S组，每组包含E行cache。每个cache的valid初始化为’n’，表示无效，tag和lru都初始化为0，同时计算组索引掩码，在计算组号时要用到。初始化完成后进入读取函数，读取轨迹。

replayTrace函数用于循环读取轨迹文件。先根据命令行输入的文件名trace\_fn，将其按行读取到buf，每次读取过程判断具体操作是S、L还是M，如果是前两者只需要访存一次，后者需要访存两次，即调用两次accessData函数。循环读取直到文件尾，完成后关闭文件。

AccessData函数用于判断每次访存的cache命中情况。首先计算出被访问的cache组号和内存标识号，然后通过组号选定这一组cache\_set。在判断访存情况之前，先把lru\_counter计数器+1，越大代表越新。随后判断是否命中，如果成功命中，则把这块cache的lru赋值为lru\_counter的值，跳出。如果未命中，则还需要判断这组cache是否还有空位，如果不满，则直接找到某一行没有使用的cache进行载入，并把这行cache的lru赋值为lru\_counter；如果满了，则需要找到lru最小的cache，即最久没有访问的cache进行替换，替换完成后，把这组cache的lru赋值为lru\_counter。所有的访存判断都结束以后，replayTrace函数也结束读文件。

Freecache函数用于释放在初始化时申请的cache空间。

printSummary函数用于打印三个计数器（命中、未命中、替换）的值。

## 优化矩阵转置操作

### 局部变量定义

两个索引变量i，j；用于控制for循环。

八个中间变量tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7, tmp8；用于缓存数组值。

### 32×32矩阵转置的优化策略

由于cache的规格是： s=5, 组索引5位, 共有2^5=32组

E=1, 每组包含1行

b=5, 块内地址5位, 块大小为2^5=32 Bytes

一个cache块 可以包含8个int型变量，由于数组是连续存储的，那么访问A[0][0]时，cache中保存了A[0][0]及其相邻变量的值，其中可能包含A[0][1],A[0][2]---A[0][7]。因此按照顺序访问，可以提高cache命中率。

读取时每次连续读取8个的相邻的int，通过八个中间变量赋值给数组B。

### 64×64矩阵转置的优化策略

试用了32\*32的转置策略，结果并不理想，可能是因为连续访存的8个int型变量分布存储在了不同的cache块中，所以改为每次访存4个int，这样访问到的变量位于同一块cache的概率要大些。

### 61×67矩阵转置的优化策略

与32\*32矩阵的转置策略一样，都采用一次读取八个连续的数据进行转置，但是61\*67的矩阵不能整除8，所以剩下的不能分为8块的部分单独解决，采用最基本的矩阵转置，这部分无优化。

# 实验过程与调试

## 实验一测试

编译完成后，用 test-csim测试，结果如图3.1所示，程序正确运行。

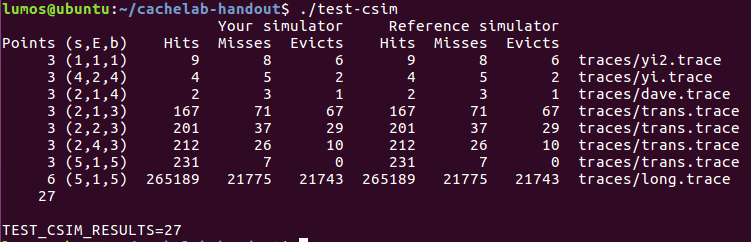


图3.1 实验一测试结果

## 实验二测试

### 32\*32矩阵测试

测试结果如图3.2所示，miss计数为287，得8分。

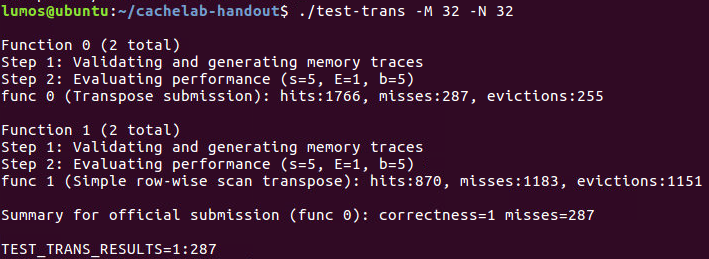


图3.2 32\*32矩阵测试

### 64\*64矩阵测试

测试结果如图3.3所示，miss计数为1651，得4分。

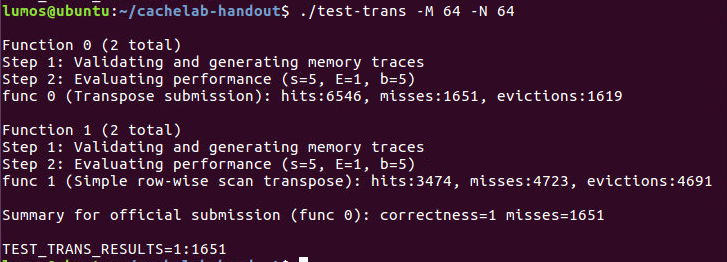


图3.3 64\*64矩阵测试

### 61\*67矩阵测试

测试结果如图3.4所示，miss计数为2192，得8分。

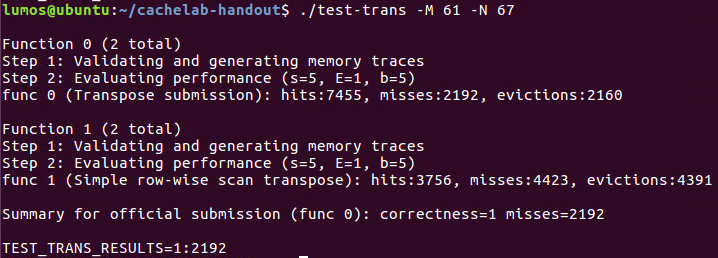


图3.4 61\*67矩阵测试

# 实验总结与心得

本次实验主要设计实现了cache的模拟器以及三个矩阵转置的优化。经过本次课设，我对cache的理解更加深刻，对cache工作方式的了解更进了一步。Cache对于程序的运行速度有着重大的影响，于是做完实验马上去了解了一下自己电脑的cache规格，只知道是8m，采用了英特尔智能高速缓存技术，现阶段还不明白怎样的编程方式最适合这种cache结构。但在以后编程序时，都会考虑到尽量提高cache命中率，这是做完本次实验之后才的思想，也是本次实验最大的收获。

# 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 |
| 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:** |