

**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 姓 名： | 樊俊超 |
| 学 院： | 计算机 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS1607 |
| 学 号： | U201614702 |
| 指导教师：ds | 万胜刚 |

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

2019年. 5月. 21日

**目 录**

[1. 第一部分：Cache模拟器 3](#_Toc9332762)

[**1.1.** **实验目的** 3](#_Toc9332763)

[**1.2.** **实验环境** 4](#_Toc9332764)

[**1.3.** **实验思路** 4](#_Toc9332765)

[**1.4.** **实验结果和分析** 5](#_Toc9332766)

[2. 第二部分：矩阵转置优化 5](#_Toc9332767)

[**2.1.** **实验目的** 5](#_Toc9332768)

[**2.2.** **实验环境** 5](#_Toc9332769)

[**2.3.** **实验思路** 6](#_Toc9332770)

[**2.4.** **实验结果和分析** 6](#_Toc9332771)

[3. 总结和体会 7](#_Toc9332772)

[4. 对实验课程的改进建议 7](#_Toc9332773)

# 第一部分：Cache模拟器

* 1. **实验目的**

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。

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

参数信息：

-h: 显示帮助信息（可选）

-v: 显示轨迹信息（可选）

-s <s>: 组索引位数

-E <E>: 关联度（每组包含的缓存行数）

-b <b>: 内存块内地址位数

-t <tracefile>: 内存访问轨迹文件名

实验要求：

* 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);

* 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. **实验环境**

Linu×-ubuntu 16.04

* 1. **实验思路**

1. 首先理清思路，实验一的主要任务就是利用C语言模仿组相联cache的访问过程，该cache采用LRU替换策略，即每次替换时将最久未使用的块替换出去，所以我们需要定义一个变量来记录所有cache使用块的最近访问次数。
2. 实验中输入包括必须部分的组索引位数s，每组行数E，地址位数b，访问文件t，以及可选部分显示帮助信息h，轨迹信息v，所以实验一针对该六项输入一一实现其目标即可。
3. 实验需要确定cache行信息，其结构定义如下：

typedef unsigned long long ULL;

typedef struct block\_info{

int byte\_width; //地址位数

bool valid; //有效

ULL last\_used; //为访问次数计数

ULL tag; //标记

} BI;

1. 定义全局变量IN\_S(int型)，IN\_E(int型)，IN\_B(int型)，IN\_V(bool型)，IN\_H(bool型)分别表示组索引位数s，每组行数E，地址位数b，是否显示轨迹，是否显示帮助信息。定义全局变量hits，misses，evictions分别记录整个访问过程后的命中数，缺失数，淘汰数，最后调用printSummary函数输出即可。

实验设计：

1. 初始化函数：init\_cache()

作用：利用所给参数以及malloc函数初始化cache，初始化时，所有cache行有效位置为false表示该行未被使用。

1. 访问函数：travel(FILE \*fp)

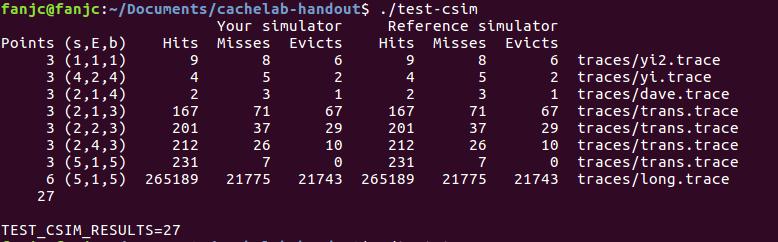
作用：逐一访问每条内存地址，生成对应的标记和索引，根据前面的操作类型决定如何调用访问cache函数，并根据IN\_V决定是否显示帮助信息。

1. 访问cache函数：hit\_cache(ULL Tag,ULL Inde×)

作用：参数分别是需要访问的标记和索引，将所有cache行的LRU计数器加一表示访问过一次，根据标记和索引找到有该标记的cache行，若有效则访问命中；否则根据含有该索引的cache组是否有位置来决定替换一个cache行还是直接调入。

* 1. **实验结果和分析**

实验完成后，生成可执行文件，调用测试程序，结果如下图



# 第二部分：矩阵转置优化

* 1. **实验目的**

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]);

实验要求：

* 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。
* 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。
* 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。、
* 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. **实验环境**

Linu×-ubuntu 16.04

* 1. **实验思路**

实验用到的cache块为32 × 32，每行可以存放8个int变量，也就是说cache一次可以存放矩阵的八行元素，也就是说相差8行整数倍的两个变量会被映射到用一个cache行中。如果按照正常的矩阵转置方法来做，矩阵A和矩阵B的同一行会被映射到cache的同一组中，由于转置是将A[i][j]转到B[j][i]，一定会出现反复冲突不命中的情况。

解决办法：通过cache分块为8 × 8解决，通过设置变量一次读出A矩阵的一行，再存入B的一行。这样，对于未包含对角线元素的12个8 × 8矩阵块，转置的miss数是8×12×2=192次，位于对角线上的分块发生32+64次缺失，总共288次，满足实验要求。

对于64×64的以及61×67的矩阵，都可以考虑使用分块的方法来减少访问cache缺失数，归根结底就是要利用好cache空间和辅助变量。

* 1. **实验结果和分析**

32×32结果如图2.1，64×64结果如图2.2，61×65结果如图2.3。

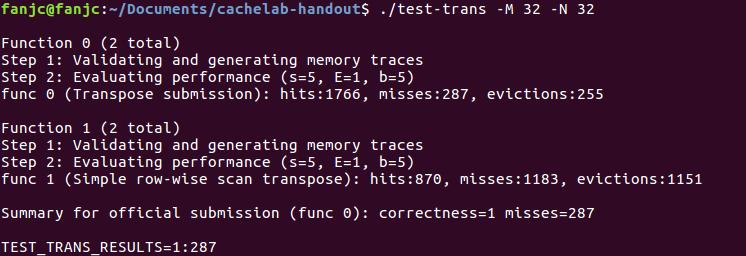


图2.1

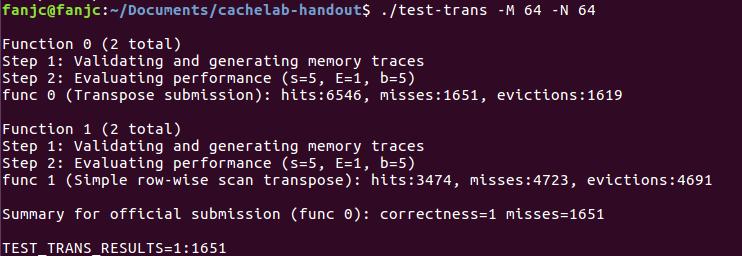


图2.2

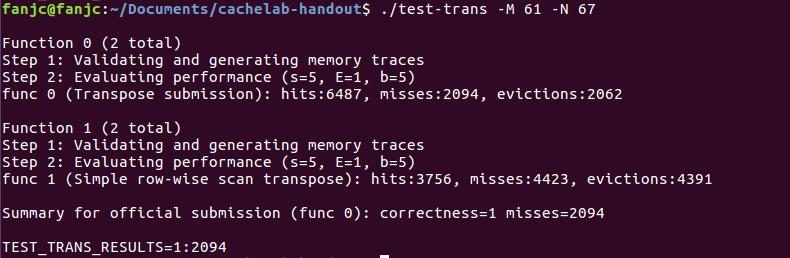


图2.3

# 总结和体会

第一个实验很有意思，在利用logisim实现cache后，我学会了使用编程了解cache的实现过程，对于组相联cache的标记和索引的获得，如何使用以及内存与cache的对应关系，cache的工作方式都有了更深的了解。

第二个实验，思路来源于课本，课本上有讲过做矩阵时cache优化可以考虑分块处理，又联想到辅助变量的使用，所以采用了分块的方式。后续64×64的优化就比较困难了，我知道没有优化到最简，也知道了优化对cache的重要性。

# 对实验课程的改进建议

无。