

**实验报告**

**实 验（六）**

题 目 Cachelab

高速缓冲器模拟

专 业 计算机类

学　　 号 1190202405

班　　 级 1903005

学 生 林逸灏

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 2021.05.26

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc72915484)

[1.1 实验目的 - 3 -](#_Toc72915485)

[1.2 实验环境与工具 - 3 -](#_Toc72915486)

[1.2.1 硬件环境 - 3 -](#_Toc72915487)

[1.2.2 软件环境 - 3 -](#_Toc72915488)

[1.2.3 开发工具 - 3 -](#_Toc72915489)

[1.3 实验预习 - 3 -](#_Toc72915490)

[第2章 实验预习 - 4 -](#_Toc72915491)

[2.1 画出存储器层级结构，标识容量价格速度等指标变化（5分） - 4 -](#_Toc72915492)

[2.2用CPUZ等查看你的计算机Cache各参数，写出各级Cache的C S E B s e b（5分） - 4 -](#_Toc72915493)

[2.3写出各类Cache的读策略与写策略（5分） - 4 -](#_Toc72915494)

[2.4 写出用gprof进行性能分析的方法（5分） - 4 -](#_Toc72915495)

[2.5写出用Valgrind进行性能分析的方法（5分） - 4 -](#_Toc72915496)

[第3章 Cache模拟与测试 - 5 -](#_Toc72915497)

[3.1 Cache模拟器设计 - 5 -](#_Toc72915498)

[3.2 矩阵转置设计 - 5 -](#_Toc72915499)

[第4章 总结 - 6 -](#_Toc72915500)

[4.1 请总结本次实验的收获 - 6 -](#_Toc72915501)

[4.2 请给出对本次实验内容的建议 - 6 -](#_Toc72915502)

[参考文献 - 7 -](#_Toc72915503)

# 第1章 实验基本信息

## 1.1 实验目的

* + 填写……理解现代计算机系统存储器层级结构
  + 掌握Cache的功能结构与访问控制策略
  + 培养Linux下的性能测试方法与技巧
  + 深入理解Cache组成结构对C程序性能的影响

## 1.2 实验环境与工具

### 1.2.1 硬件环境

* + X64 CPU；2GHz；2G RAM；256GHD Disk 以上

### 1.2.2 软件环境

### Windows7 64位以上；VirtualBox/Vmware 11以上；Ubuntu 16.04 LTS 64位/优麒麟 64位；

### 1.2.3 开发工具

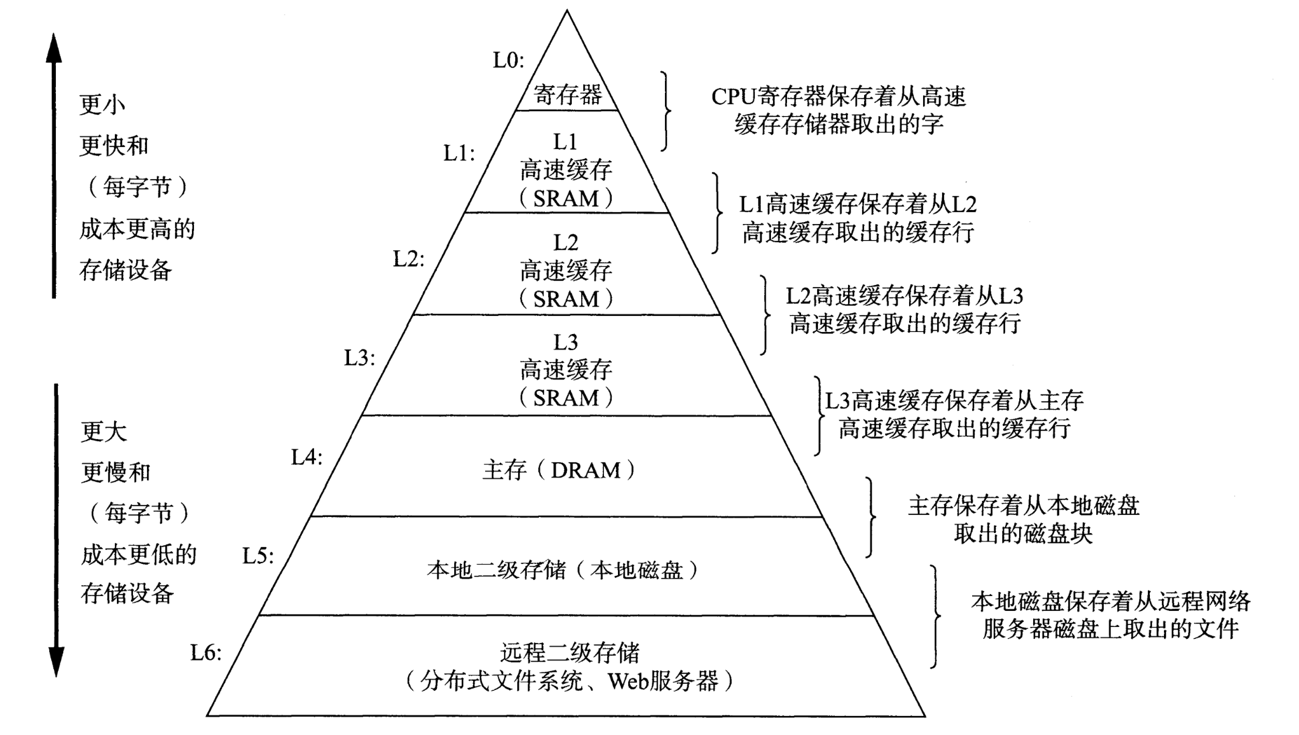
* + Visual Studio 2010 64位以上；TestStudio；Gprof;Valgrind等

## 1.3 实验预习

填写

# 第2章 实验预习

## 2.1 画出存储器层级结构，标识容量价格速度等指标变化（5分）



## 2.2用CPUZ等查看你的计算机Cache各参数，写出各级Cache的C S E B s e b（5分）



一级数据Cache大小为32Kb，采用八路组相连，每一路块大小为64B，故E=8，B=64，C=32\*1024B=S\*8\*64，S=64，故s=6，e=3，b=6.

一级指令cache分析同上。

二级Cache大小为256KB，采用四路组相联，故E=4，每一路块大小为64B，C=256\*1024\*B=S\*4\*64，S=1024，s=10，e=2，b=6

三级Cache大小为12MB，采用16路组相联，E=16，每一路块大小为64B，C=12\*220B=S\*16\*64，S=3\*212，s=14，e=4，b=6

## 2.3写出各类Cache的读策略与写策略（5分）

Cache读策略

1：命中，则从cache中读相应数据到CPU或上一级cache中。

2：失败，则从主存或下一级cache中读取数据，并替换出一行数据，通常采用LRU算法。

Cache写策略

1：命中，又分两种策略

（1）写回法：只写本级cache，暂时不写数据到主存或下一级cache，等到该行被替换出去时，才将数据写回到主存或下一级cache。

（2）写直达：写本级cache，同时写数据到主存或下一级cache，等到该行被替换出去时，就不用写回数据了。

2：失败，又分两种策略

（1）按写分配，又分两种：[1]先写数据到主存或下一级cache，并从主存或下一级cache读取刚才修改过的数据，即：先写数据，再为所写数据分配cache line；[2]先分配cache line给所写数据，即：从主存中读取一行数据到cache，然后直接对cache进行修改，并不把数据到写到主存或下一级cache，一直等到该行被替换出去，才写数据到主存或下一级cache。

（2）写不分配：直接写数据到主存或下一级cache，并且不从主存或下一级cache中读取被改写的数据，即：不分配cache line给被修改的数据

## 2.4 写出用gprof进行性能分析的方法（5分）

gprof是GNU profile工具，可以运行于linux、AIX、Sun等操作系统进行C、C++、Pascal、Fortran程序的性能分析，用于程序的性能优化以及程序瓶颈问题的查找和解决。通过分析应用程序运行时产生的“flat profile”，可以得到每个函数的调用次数，每个函数消耗的处理器时间，也可以得到函数的“调用关系图”，包括函数调用的层次关系，每个函数调用花费了多少时间。使用步骤如下：

（1）用gcc、g++、xlC编译程序时，使用-pg参数，如：g++ -pg -o test.exe test.cpp编译器会自动在目标代码中插入用于性能测试的代码片断，这些代码在程序运行时采集并记录函数的调用关系和调用次数，并记录函数自身执行时间和被调用函数的执行时间。

（2）执行编译后的可执行程序，如：./test.exe。该步骤运行程序的时间会稍慢于正常编译的可执行程序的运行时间。程序运行结束后，会在程序所在路径下生成一个缺省文件名为gmon.out的文件，这个文件就是记录程序运行的性能、调用关系、调用次数等信息的数据文件。

（3）使用gprof命令来分析记录程序运行信息的gmon.out文件，如：gprof test.exe gmon.out则可以在显示器上看到函数调用相关的统计、分析信息。上述信息也可以采用gprof test.exe gmon.out> gprofresult.txt重定向到文本文件以便于后续分析。

注意事项：

程序如果不是从main return或exit()退出，则可能不生成gmon.out。

程序如果崩溃，可能不生成gmon.out。

测试发现在虚拟机上运行，可能不生成gmon.out。

一定不能捕获、忽略SIGPROF信号。man手册对SIGPROF的解释是：profiling timer expired. 如果忽略这个信号，gprof的输出则是：Each sample counts as 0.01 seconds. no time accumulated.

如果程序运行时间非常短，则gprof可能无效。因为受到启动、初始化、退出等函数运行时间的影响。

程序忽略SIGPROF信号！

## 2.5写出用Valgrind进行性能分析的方法（5分）

Valgrind是运行在Linux上一套基于仿真技术的程序调试和分析工具，它包含一个内核──一个软件合成的CPU，和一系列的小工具，每个工具都可以完成一项任务──调试，分析，或测试等。Valgrind可以检测内存泄漏和内存违例，还可以分析cache的使用等。Valgrind包含以下工具：Memcheck（用来检测程序中出现的内存问题，所有对内存的读写都会被检测到，一切对malloc()/free()/new/delete的调用都会被捕获）、Callgrind（收集程序运行时的一些数据，建立函数调用关系图，还可以有选择地进行cache模拟。在运行结束时，它会把分析数据写入一个文件，callgrind\_annotate可以把这个文件的内容转化成可读的形式）、Cachegrind（模拟CPU中的一级缓存I1，Dl和二级缓存，能够精确地指出程序中cache的丢失和命中。如果需要，它还能够为我们提供cache丢失次数，内存引用次数，以及每行代码，每个函数，每个模块，整个程序产生的指令数）、Helgrind（用来检查多线程程序中出现的竞争问题）、Massif（堆栈分析器，能测量程序在堆栈中使用了多少内存，告诉我们堆块，堆管理块和栈的大小）。Valgrind的使用非常简单，valgrind命令的格式如下：valgrind [valgrind-options] your-prog [your-prog options] 。一些常用的选项如下：

Valgrind是运行在Linux上一套基于仿真技术的程序调试和分析工具，它包含一个内核──一个软件合成的CPU，和一系列的小工具，每个工具都可以完成一项任务──调试，分析，或测试等。

一些常用的选项如下：   
选项   
作用   
-h --help   
显示帮助信息。   
--version   
显示valgrind内核的版本，每个工具都有各自的版本。   
-q --quiet   
安静地运行，只打印错误信息。   
-v --verbose   
打印更详细的信息。   
--tool= [default: memcheck]   
最常用的选项。运行valgrind中名为toolname的工具。如果省略工具名，默认运行memcheck。   
--db-attach= [default: no]

# 第3章 Cache模拟与测试

## 3.1 Cache模拟器设计

提交csim.c

程序设计思想：

1.编写一个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(基于LRU算法)的总数

该模拟器能处理如下指令：

./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>

其中的各个参数为：

* + -h: 显**csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>**示帮助（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>: 缓存行的内存块内地址位数
  + -t <tracefile>: 内存访问轨迹文件名

2.分析其中的各个函数：

1) void initCache()函数：为cache分配内存，有效位为0，并标记和LRU，同时计算set\_index\_mask：

对每一块与每一组分配内存：

cache = malloc(S \* sizeof(cache\_set\_t))；

cache[i] = malloc(E \* sizeof(cache\_line\_t)；

将cache中的每一位全部置为0，设为有效：

2) void freeCache()函数：需要把在initCache()里申请的空间释放，我们可以一组一组的释放Cache中的空间。

3) void accessData(mem\_addr\_t addr)函数：访问内存地址addr中的数据，若该数据已经在缓存中，就执行hit\_count++;若它不在缓存中，就执行miss\_count++;若该线被驱逐，就执行evict\_count++。以此来记录命中，不命中，驱逐的访存记录。该函数的设计，对内存中数据的访问，需要分以下几种情况讨论：

(1)若命中，且有效位为1，执行hit\_count++；

(2)若不命中，执行miss\_count++;

(3)不命中后需要考虑它是否被驱逐。遍历每一组，若该组中有一个有效位为0，则说明未满，则将有效位不是1的那一行更新有效位为1，再更新lru计数与标记为；若有效位全为1，则需要驱逐，eviction\_count++;

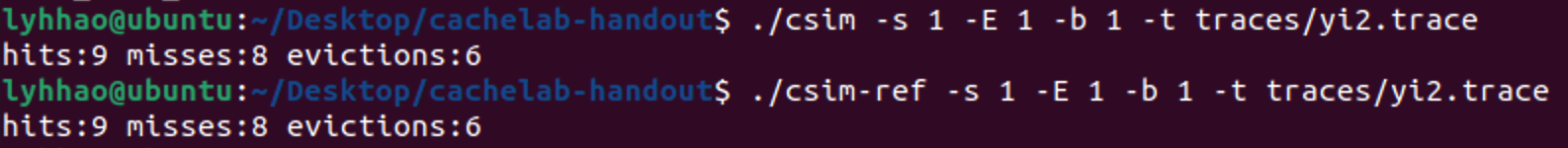
3. 计算S、E、B的值

S = 1<<s;

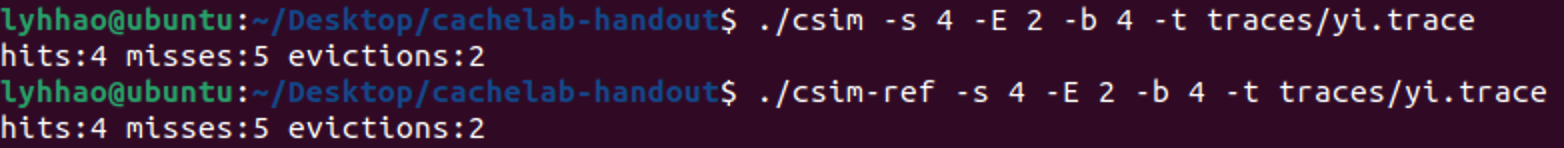
B = 1<<b;

E=E;

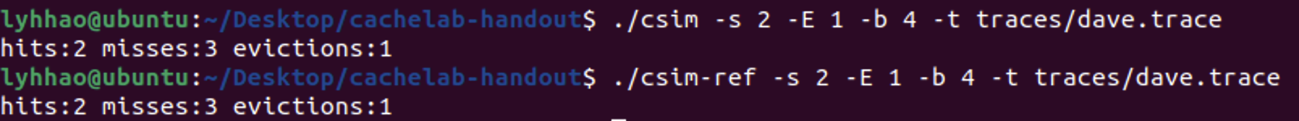
测试用例1的输出截图（5分）：



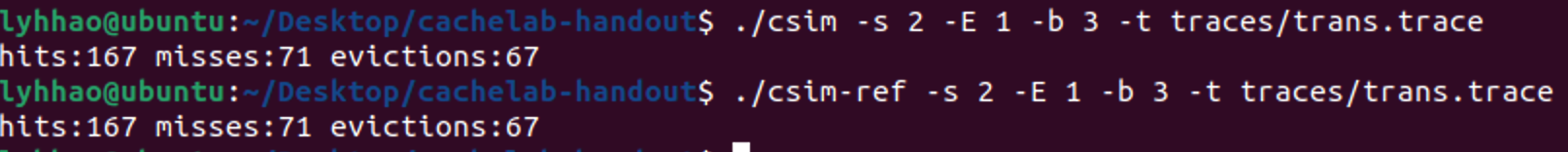
测试用例2的输出截图（5分）：



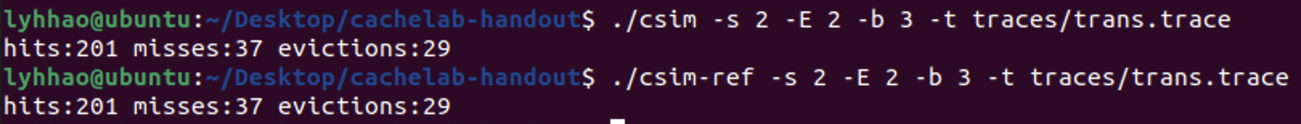
测试用例3的输出截图（5分）：



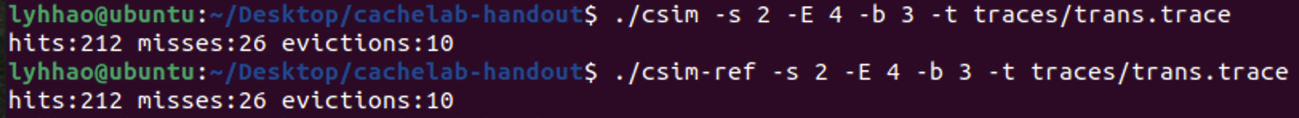
测试用例4的输出截图（5分）：



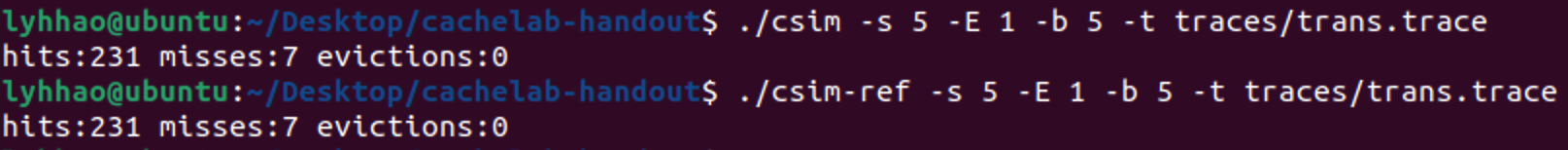
测试用例5的输出截图（5分）：



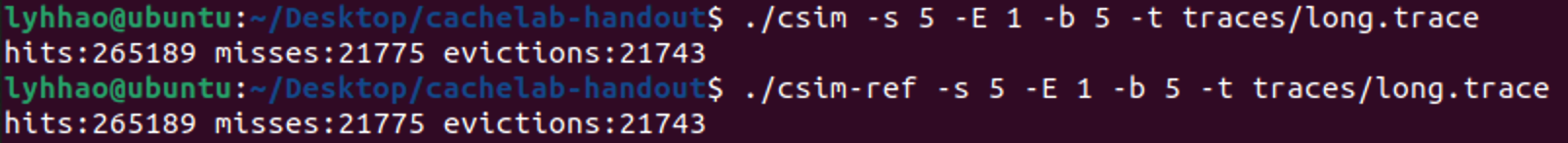
测试用例6的输出截图（5分）：



测试用例7的输出截图（5分）：

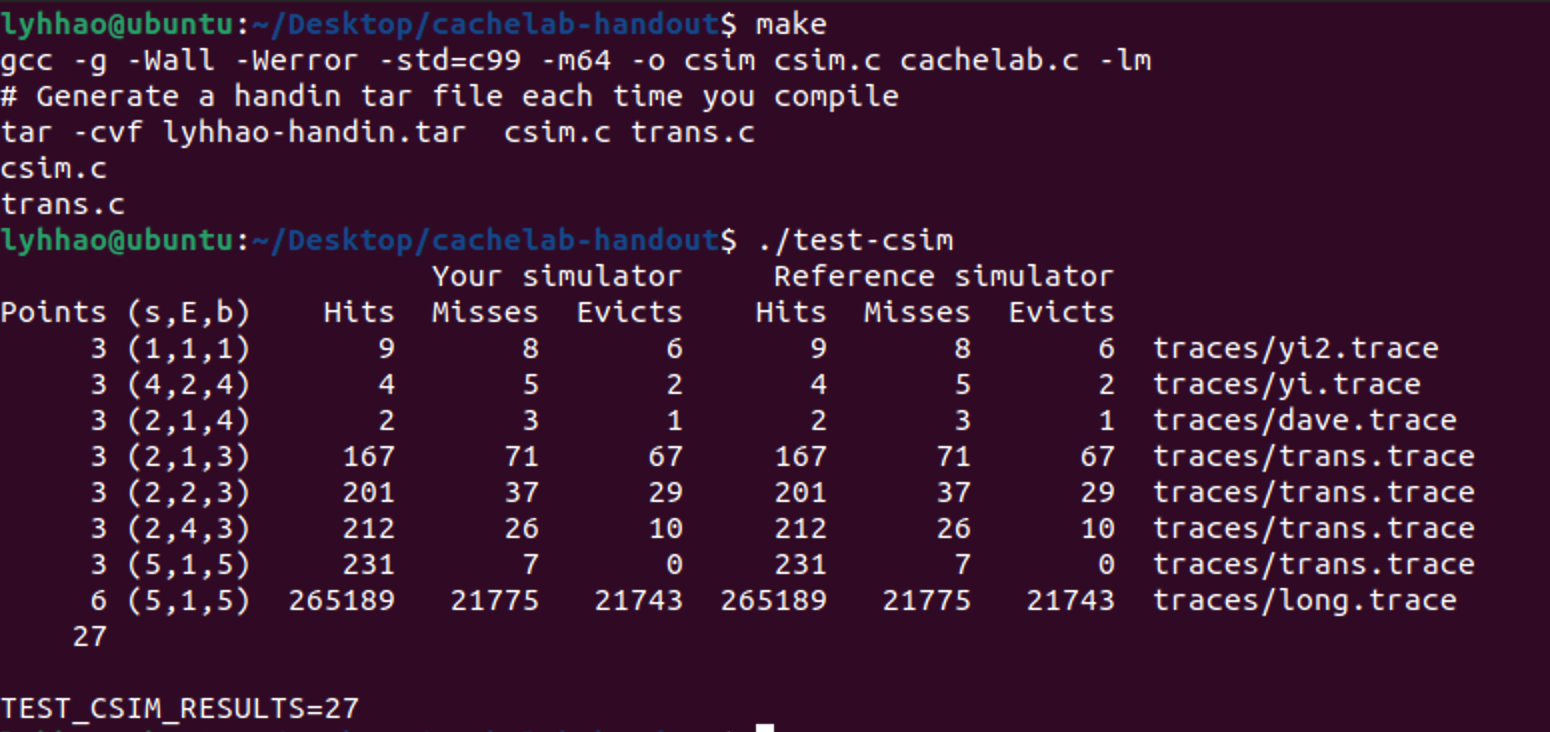


测试用例8的输出截图（10分）：



**注：每个用例的每一指标5分（最后一个用例10）**——与参考csim-ref模拟器输出指标相同则判为正确

执行test-csim后的结果：



## 3.2 矩阵转置设计

提交trans.c

程序设计思想：

实现矩阵转置，对于给定矩阵A[M][N]，得到对应的转置矩阵B[N][M]。使函数调用时cache的不命中率尽可能小。满足：

32×32：如果m<300得10分，如果m>600得0分，对其他m得(600-m)\*10/300分。

64×64：如果m<1300得10分，如果m>2000得0分，对其他m得(2000-m)\*10/700分。

61×67：如果m<2000得20分，如果m>3000得0分，对其他m得(3000-m)\*20/1000分。

矩阵转置实现要求：

1.限制对栈的使用，最多定义和使用12个int类型的局部变量。

2.不允许使用递归。

3.不允许改变矩阵A

4.不允许使用malloc

5.在测试中，cache的参数为s = 5，E=1，b=5。所以该缓存共32组，每组中块的大小为32字节。Cache只能装下32\*8=256个int类型的变量，待转换矩阵大小远大于cache的大小。

对于不同大小的矩阵数组的转置：

1. 32 \* 32

因为cache能存储的数据的数量远小于该矩阵的大小，每一行的cache能存8个数据，所以考虑将32\*32的矩阵划分为8\*8的块，每次对这个8\*8的块做转置的操作 。

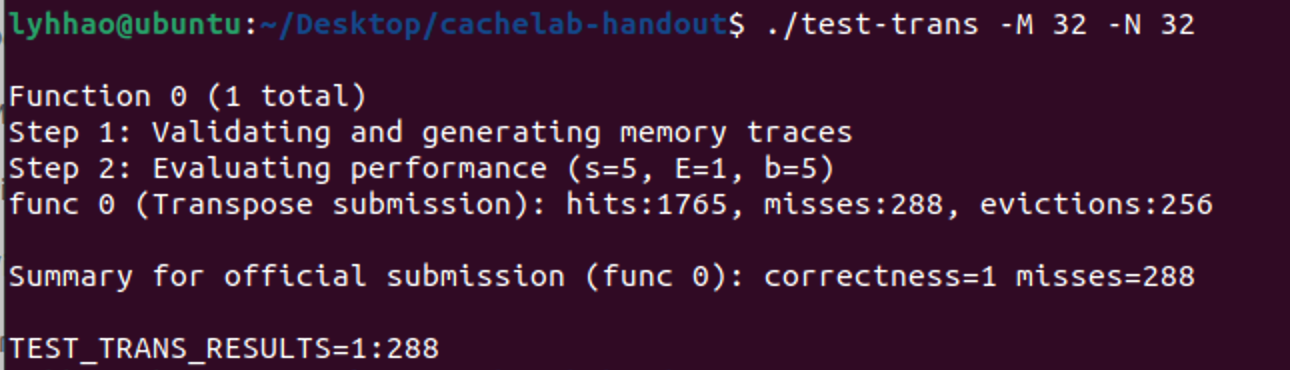
1. 64 \* 64

Cache每次能存入4行矩阵里的元素，64 \* 4 = 256。对cache中的每个块进行考虑，一个块存储8个int值，64为8的倍数，因此同一个块中的元素都是位于同一行的。先考虑像上一题用8\*8的矩阵来存储数据，但是报错，原因是访问完之前的数据后，再访问后面的数据会冲突不命中，使得原来的块被驱逐，无法运行。也按照将该矩阵进行划分的思想，每次读取同一行的八个元素，先将前四个存到转置对应的B矩阵的位置上，剩下的四个元素暂时存放到B矩阵的最右下的一列中保存（因为这一列是最后才进行数据的赋值与加载的），再将这些元素放到B矩阵对应的位置上，这八个元素即完成了转置操作。对矩阵其他的元素也进行如上的操作，即可完成转置。

1. 61 \* 67

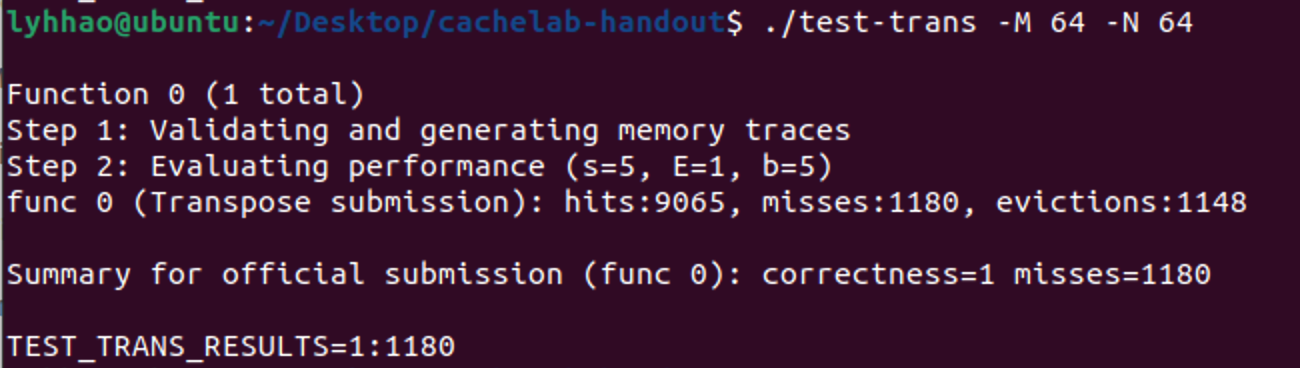
这个矩阵比较特殊，不像上面的矩阵一样，每一行元素能占据整数个块，我们可以不断尝试不同的分块方式，发现23\*23的分块方式的效果较好，可行。

**32×32（10分）：运行结果截图**



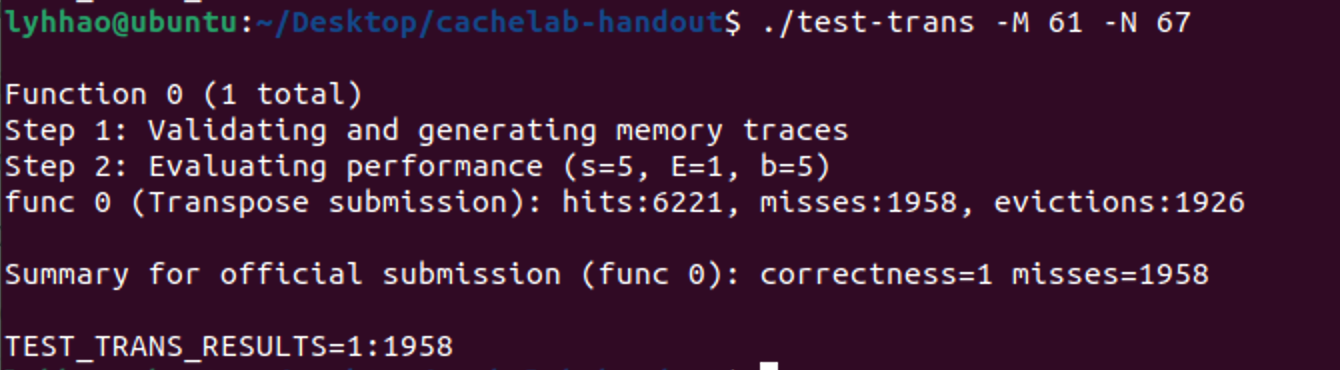
**m=288**

**64×64（10分）：运行结果截图**



**m=1180**

**61×67（20分）：运行结果截图**



**m=1958**

# 第4章 总结

## 4.1 请总结本次实验的收获

更加深入理解了cache缓存的相关知识，了解了缓存命中的原理，

## 4.2 请给出对本次实验内容的建议

无

注：本章为酌情加分项。

# 参考文献

**为完成本次实验你翻阅的书籍与网站等**

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/ collection/anatmorp.