

**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 1190200523

班　　 级 1903002

学 生 石翔宇

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.5.28

**计算机科学与技术学院**

**目 录**

[第1章 实验基本信息 - 3 -](#_Toc73190990)

[1.1 实验目的 - 3 -](#_Toc73190991)

[1.2 实验环境与工具 - 3 -](#_Toc73190992)

[1.2.1 硬件环境 - 3 -](#_Toc73190993)

[1.2.2 软件环境 - 3 -](#_Toc73190994)

[1.2.3 开发工具 - 3 -](#_Toc73190995)

[1.3 实验预习 - 3 -](#_Toc73190996)

[第2章 实验预习 - 5 -](#_Toc73190997)

[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） - 5 -](#_Toc73190998)

[2.2计算机Cache的参数查看与分析（5分） - 5 -](#_Toc73190999)

[2.3写出各类Cache的读策略与写策略（5分） - 5 -](#_Toc73191000)

[2.4 写出用gprof进行性能分析的方法（5分） - 6 -](#_Toc73191001)

[2.5写出用Valgrind进行性能分析的方法（5分） - 6 -](#_Toc73191002)

[第3章 Cache模拟与测试 - 7 -](#_Toc73191003)

[3.1 Cache模拟器设计 - 7 -](#_Toc73191004)

[3.2 矩阵转置设计 - 8 -](#_Toc73191005)

[第4章 总结 - 11 -](#_Toc73191006)

[4.1 请总结本次实验的收获 - 11 -](#_Toc73191007)

[4.2 请给出对本次实验内容的建议 - 11 -](#_Toc73191008)

[参考文献 - 12 -](#_Toc73191009)

# 第1章 实验基本信息

## 1.1 实验目的

* 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

* Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz
* 16GB RAM
* 1TB HDD + 512G SSD

### 1.2.2 软件环境

* Windows 10 21H1
* Ubuntu 20.04 LTS

### 1.2.3 开发工具

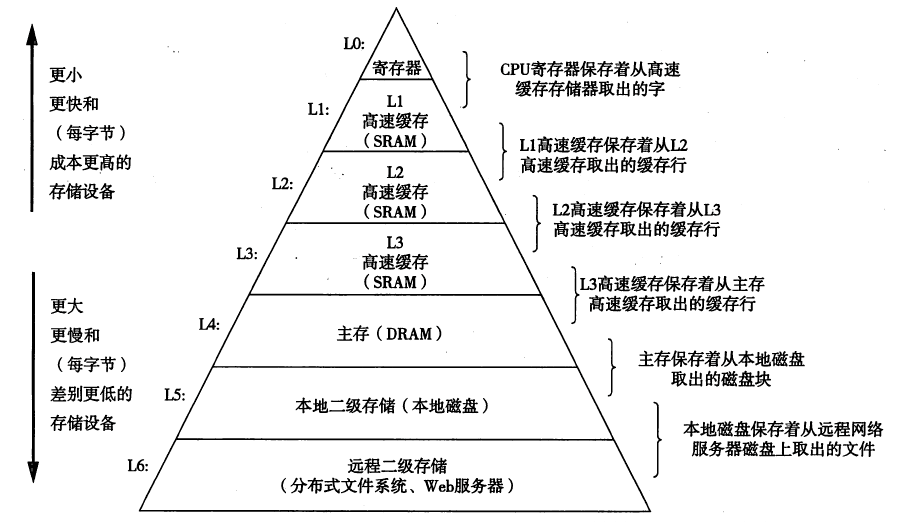
* VSCode，CodeBlocks，gcc+gdb

## 1.3 实验预习

* 上实验课前，必须认真预习实验指导书（PPT或PDF）
* 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。
* 画出存储器的层级结构，标识其容量价格速度等指标变化
* 用CPUZ等查看你的计算机Cache各参数，写出Cache的基本结构与参数：缓存大小C、分组数量 S、关联度/组内行数 E、块大小 B，及对应的编码位数 ：组索引位数s、 e 、块内偏移位数b
* 写出Cache的各种读策略与写策略
* 掌握Valgrind、gprof的使用方法

# 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计算机Cache的参数查看与分析（5分）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Name | C(大小) | S(组数) | E(路数) | B(块大小) | s（组编码位数） | b（块内偏移地址位数） |
| L1 Data cache | 32 KBytes | 64 | 8-way set associative | 64-byte line size | 6 | 6 |
| L1 Instruction cache | 32 KBytes | 64 | 8-way set associative | 64-byte line size | 6 | 6 |
| L2 cache | 256 KBytes | 1024 | 4-way set associative | 64-byte line size | 10 | 6 |
| L3 cache | 12 MBytes, | 2048 | 16-way set associative | 64-byte line size | 11 | 6 |

## 2.3写出各类Cache的读策略与写策略（5分）

1. 读策略：
   1. 命中：将cache中的数据存储到CPU或者上一级cache中。
   2. 未命中：从下一级cache中读取数据，存储在当前层级的cache中。
2. 写策略：
   1. 命中：
      1. Write-through: 命中后更新缓存，同时写入到内存中。
      2. Write-through: 命中后更新缓存，同时写入到内存中。
   2. 未命中：
      1. Write-allocate: 载入到缓存中，并更新缓存。
      2. No-write-allocate: 直接写入到内存中，不载入到缓存。

## 2.4 写出用gprof进行性能分析的方法（5分）

Profiling 可以使我们看到程序运行时程序的调用关系、函数的消耗时长等。这些信息可以使我们了解程序中那块代码耗时高于预期。

使用 Profiling 主要包括如下三步：

1. 编译链接程序时要添加-pg选项使能 profiling。
2. 执行编译处的可执行文件，产生 profile 数据文件gmon.out，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 分析 profile 数据。

根据产生的 profile ，可以产生各种不同实行的分析输出。如 The Flat Profile、The Call Graph、The Annotated Source Listing。

## 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，灵活轻巧而又强大。

在linux下，使用命令valgrind --tool=callgrind execname会在当前目录下生成文件，文件名字为“callgrind.out.进程号"

生成的文件有两种处理方式：

1. 生成dot文件

gprof2dot.py -f callgrind -n10 -s callgrind.out.31113 > valgrind.dot

1. 将dot文件转换成图片

dot -Tpng valgrind.dot -o valgrind.png

# 第3章 Cache模拟与测试

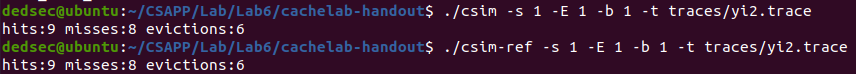
## 3.1 Cache模拟器设计

提交csim.c

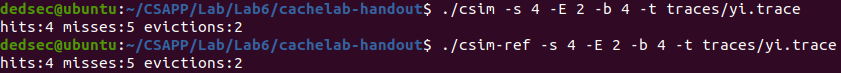
程序设计思想：

1. 维护全局变量lru\_counter， 每次操作完成后加一。
2. 操作时：
   1. 先找出组索引和标记。
   2. 扫描整个块的数组，若匹配到了数据（valid=1且tag相同），则hit，注意此时要更新当前行的lru，将当前的lru\_counter赋值给它即可，操作完成直接结束函数。
   3. 扫描整个块的数组，找到lru最小的行，若扫描完之后还没有进行b步的操作，则 miss。若lru最小的行的valid=1，则eviction。更新当前行的数据。

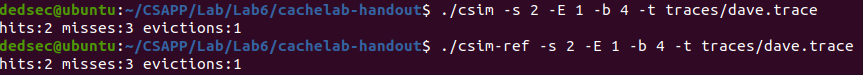
测试用例1的输出截图（5分）：



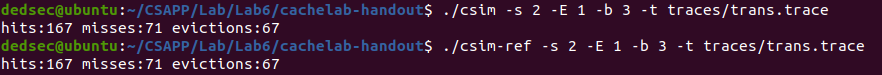
测试用例2的输出截图（5分）：



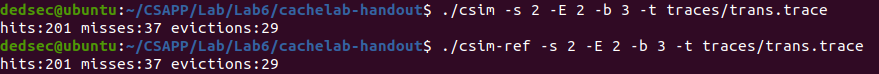
测试用例3的输出截图（5分）：



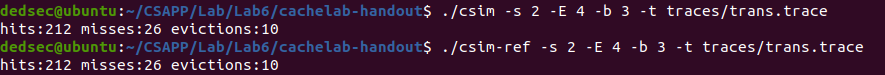
测试用例4的输出截图（5分）：



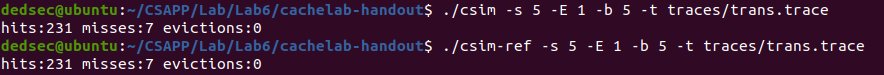
测试用例5的输出截图（5分）：



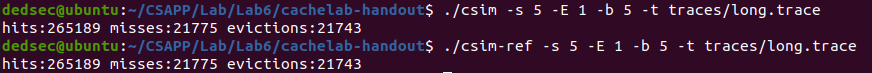
测试用例6的输出截图（5分）：



测试用例7的输出截图（5分）：

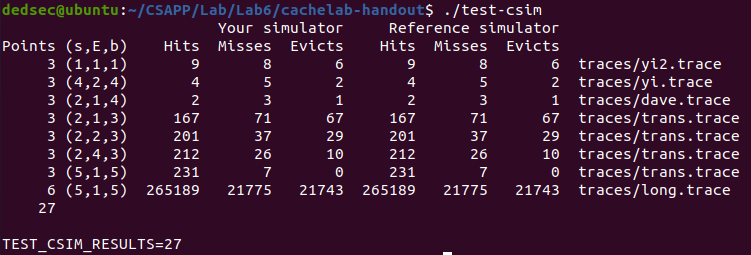


测试用例8的输出截图（10分）：



**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 后结果图：



## 3.2 矩阵转置设计

提交trans.c

程序设计思想：

缓存一共有 32组，每个组有一个cache line，每个块的大小为 32 字节，也就是说一个cache line里面能装入 8 个 int。缓存总共能装进256个int。

1. 32×32:

鉴于同一个矩阵中的缓存冲突问题，我们考虑分块，由于一个cache line中能装进8个int，且缓存可装进8行的数据，则选择8\*8的块。对每一块操作时，我们先依次访问当前行的8个int，将这8个数据放入8个局部变量中，再将这8个放到转置后的矩阵当中。

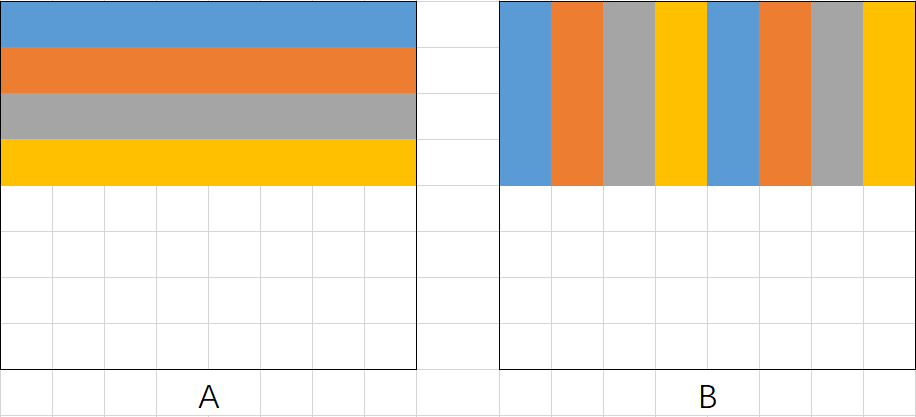
1. 64×64:

每一行的数据个数增加到了64，这时若还使用上述的8\*8分块，则块中第0行和第4行的数据会发生冲突，这是因为64×64的矩阵每个行需要8个缓存块，每四行缓存index会重复一次。

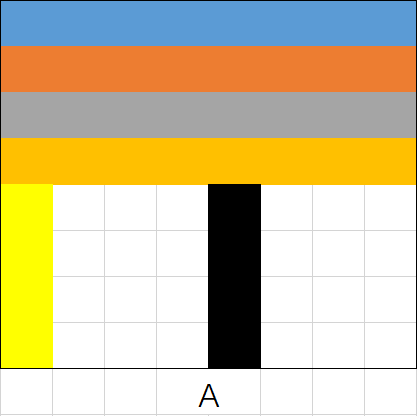
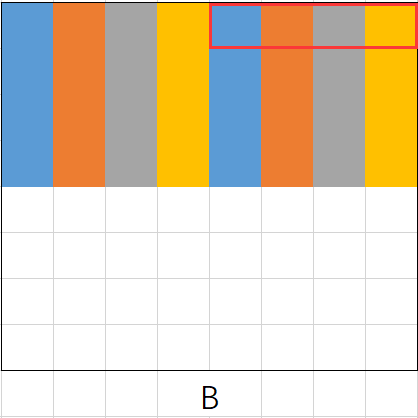
若使用4\*4分块，则没有充分利用到每次加载到缓存中的数据。

考虑新的方法，将8\*8和4\*4分块结合起来。

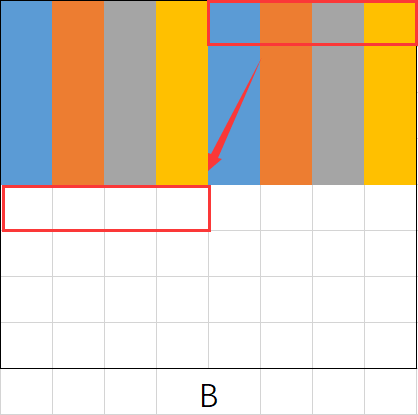
* + - 1. 8\*8分块。
      2. 遍历A的前4行，将每行的前4个数字放在B中相应的位置。将每行的后4个数字放在前4个数字位置列数+4的位置，如图所示。放置时同样使用上述的策略。



* + - 1. 遍历k从0到3。k表示列数。（**图示均为k=0时的情况**）
         1. 将8\*8块中的左下角4\*4的小块的第k列数据（图中黄色部分）复制到局部变量tmp[0~3]中，将右下角4\*4的小块的第k列数据（图中黑色部分）复制到局部变量tmp[4~7]中。

* + - * 1. 则tmp[1~3]中的数据应该放到上图中红色框所示部分。将tmp[1~3]分别与上图红色框中数据交换。

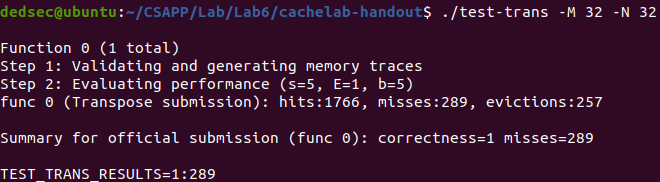


* + - * 1. 将tmp[1~3]中数据放到左下角4\*4小块的第k行，如上图所示。
        2. 将tmp[4~7]中数据放到右下角4\*4小块的第k行。
        3. 重复上述步骤。

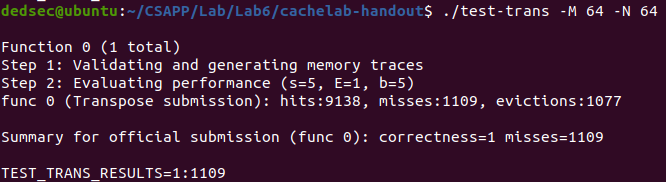
1. 61×67:

使用与64×64相同的方法。注意分块后会有块中元素个数不满的情况。

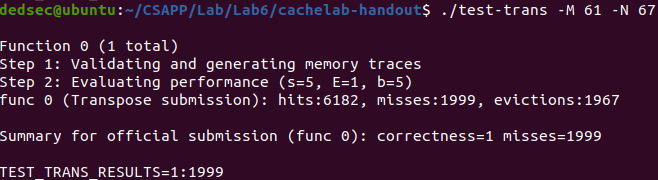
**32×32（10分）：运行结果截图**



**64×64（10分）：运行结果截图**



**61×67（20分）：运行结果截图**



# 第4章 总结

## 4.1 请总结本次实验的收获

1. 更加深入的了解了缓存的相关知识和缓冲命中的原理；
2. 学会了通过优化代码实现增加缓存命中率，加快程序运行速度的方法。

## 4.2 请给出对本次实验内容的建议

1. 希望实验指导书能够更加详细，更加有条理。

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.