哈爾濱工業大學

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　1190201816

班　　级　1903012

学　　生　樊红雨

指 导 教 师　史先俊

实 验 地 点　G709

实 验 日 期　2021.5.26

计算机科学与技术学院

目 录

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

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

■ 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

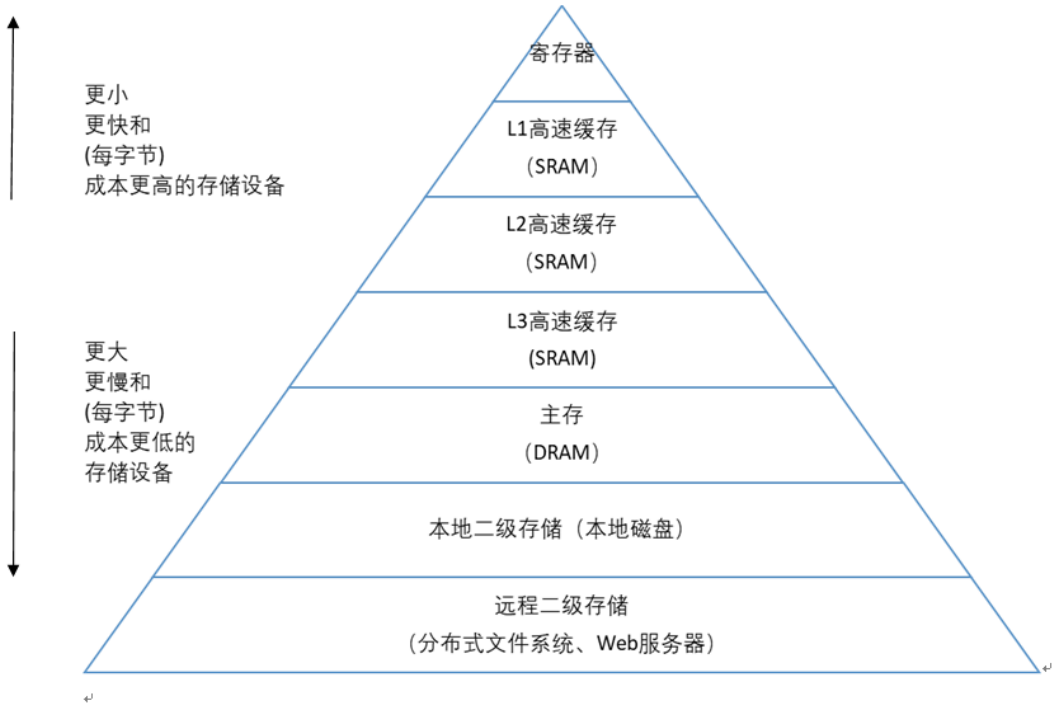
对应的编码位数 ：组索引位数 s、 e 、块内偏移位数 b

■ 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）



从上到下分别为 L0 L1 L2 L3 L4 L5 L6 L6

高层的存储器保存着从底层的存储器取出的缓存行。

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）

一级数据缓存：C = 32KB*4=128KB　　S = 8　　E = 64　s = 3　　e = 6

一级指令缓存：C = 32KB*4=128KB　　S = 8　　E = 64　s = 3　　e = 6

二级缓存：C = 256KB*4=1024KB　S = 4　　E = 64　s = 2　　e = 6

三级缓存：C =8MB　　S = 16　E = 64　s = 4　　e = 6

2.3 写出各类 Cache 的读策略与写策略（5 分）

Cache 读策略

1：缓存命中，则从 cache 中读相应数据到 CPU 或上一级 cache 中。

2：缓存不命中，则从主存或下一级 cache 中读取数据，并替换出一行数据。

Cache 写策略

1、写回

当 CPU 写 Cache 命中时，只修改 Cache 的内容，而不是立即写入主存；只有当此块被换出时才写回主存。

2、直写

立即将一个已经缓存了的字 w 的高速缓存块写回到紧接着的第一层中。。

3、写分配

加载相应的低一层的块到高速缓存中，然后更新这个高速缓存块。

4、非写分配

避开高速缓存，直接把这个字写到低一层中去。

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。

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] 。

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：

初始化 Cache：

```
void initCache()
{
    //先申请S个组的地址
    cache = (cache_set_t*)malloc(sizeof(cache_set_t)*S);
    if (cache == NULL)
    {
        printf("申请组空间失败。\n");
    }
    int i,j;
    //再申请每个组中，E行的地址
    for ( i = 0; i < S; i++)
    {
        cache[i] = (cache_line_t*)malloc(sizeof(cache_line_t)*E);
        if (cache[i] == NULL)
        {
            printf("申请行空间失败\n");
        }
    }
    //对每一组，每一行中的值进行初始化
    for ( i = 0; i < S; i++)
    {
        for ( j = 0; j < E; j++)
        {
            cache[i][j].valid = 0;
            cache[i][j].tag = 0;
            cache[i][j].lru = 0;
        }

    }
    set_index_mask = (1 << s) -1;
```

首先为 Cache 申请 S 个组的空间，然后为每一组申请 E 行的空间。

Cache 类似于二维数组，第一维表示组数，第二维表示每组的行数。申请空间

后，对每一行进行初始化，将有效位，标签，和 LRU 计数都初始化为 0。

最后将掩码 set_index_mask 初始化为 (1 << s) -1。

这样将地址的组索引与该掩码对齐后进行按位与操作即可获得地址中 set 部分。

freeCache：

```
void freeCache()
{
    int i;
    for ( i = 0; i < S; i++)
    {
        free(cache[i]);
    }
    free(cache);
}
```

先释放每一组中的空间，最后释放申请的所有空间。

模拟 cache 工作的函数：

```
mem_addr_t set = (addr >> b) & set_index_mask;
mem_addr_t tag = addr >> (b+s);

int i;
int j;
int flag = 0;
unsigned long long int maxLru = 0;
int maxLruIndex;
for ( i = 0; i < E; i++)
{
    if ((cache[set][i].valid == 1) && (cache[set][i].tag == tag))
    {
        flag =1;
        hit_count++;
        if (verbosity)
        {
            printf("hit ");
        }
        for ( j = 0; j < E; j++)
        {
            cache[set][j].lru++;
        }
        cache[set][i].lru = 0;
        break;
    }
```

首先通过

```
set = (addr >> b) & set_index_mask;
```

获得地址中的组号，

```
tag = addr >> (b+s);
```

获得地址中的标记位。

然后进入循环，对该组的所有行进行搜索，当某一行的有效位为 1，并且标记位与地址中的标记位相同时，说明该次搜索命中。

令命中标志 flag 为 1，并且将命中次数加 1.将该组中除命中行的 LRU 全部加 1，表示这次未使用该行，将命中行的 LRU 置为 0，说明该行刚刚被读。

若搜索了该组中所有的行，都没有找到，则将不命中次数加 1.并且找到 LRU 最大的某一行，将数据加载到这一行，若这一行的有效位为 0，说明是冷不命中，将有效位置为 1，若这一行的有效位为 1，说明是冲突不命中，将该行原有数据驱逐，将驱逐计数器加 1。同时将其他行的 LRU 加 1，将刚刚被加载行的 LRU 置为 0.

测试用例 1 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 1
 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
```

测试用例 2 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 4
 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
```

测试用例 3 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 2
 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
```

测试用例 4 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 2
 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
```

测试用例 5 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 2
 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
```

测试用例 6 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 2
 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
```

测试用例 7 的输出截图（5 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 5
 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
```

测试用例 8 的输出截图（10 分）：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./csim -s 5
 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
```

总体：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./test-csim
                        Your simulator     Reference simulator
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27
```

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

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

1. 32*32

   我们可以知道，Cache 有 32 组，每组 1 行，每行存放 32 个字节。每个 int 类型的变量占用 4 个字节，则每一行可以存放 8 个 int 类型的变量。即需要四个块来存放矩阵中每一行的数据。即存储矩阵中每一行的数据需要 4

个组，所以矩阵中 8 行的数据就将 Cache 填满。则我们可以知道，矩阵中相差为 8 的行占用 Cache 中相同的块。

我们可以使用分块的思想，将两个大矩阵分为多个 8*8 的小矩阵，进行转置运算。如图：

A`：

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

B`：

1`	5`	9`	13`
2`	6`	10`	14`
3`	7`	11`	15`
4`	8`	12`	16`

（每个数字代表一个 8*8 的矩阵）

每次将这个 8*8 的小矩阵进行转置操作。

每次读入 A 中某一行的 8 个数据到 Cache，这一行刚刚好占用 Cache 的一个块。将这一行利用 8 个临时变量保存，然后写入 B 中对应的一列上。

每一次进行这样的操作，从 A 中读取数据只会有一次冷不命中，其他都会命中，虽然第一次写入 B 中会造成 8 次冲突不命中，但分块后 B 的一块已经全部存储在 Cache 中，之后继续写入 B 都是命中。

2. 64*64

对于 64*64 的矩阵，若继续使用 8*8 的分块矩阵，则会导致写入 B 中的数据全部是冲突不命中。如果按照 4*4 的分块矩阵，那么在同一行中，两个左右相邻的小矩阵，可能用的是 Cache 中的同一个块，会造成冲突不命中。这在上面 32*32 的矩阵中是不存在的，那么我想，可以先把 64*64 的矩阵分为 4 个 32*32 个矩阵，然后依次将这四个 32*32 的矩阵按照刚刚进行的操作转置。

3. 61*67

仍然采用分块的办法，由于 61 与 67 并不是 8 的整数倍，所以分块方式较

难寻找，只好尝试几个分块方法进行比较，最后发现，当使用 17*17 时符合 miss 小于 2000 的条件。

总体截图：

```
Part B: Testing transpose function
Running ./test-trans -M 32 -N 32
Running ./test-trans -M 64 -N 64
Running ./test-trans -M 61 -N 67

Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         288
Trans perf 64x64           8.0         8        1244
Trans perf 61x67          10.0        10        1817
        Total points      53.0        53
```

32×32（10 分）：运行结果截图：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./test-trans -M 32 -N 32

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1765, misses:288, evictions:256
```

64×64（10 分）：运行结果截图：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./test-trans -M 64 -N 64

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:9001, misses:1244, evictions:1212
```

61×67（20 分）：运行结果截图：

```
fhy-1190201816@fhy1190201816-virtual-machine:/home/cachelab-handout$ ./test-trans -M 61 -N 67

Function 0 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6362, misses:1817, evictions:1785
```

第 4 章 总结

4.1 请总结本次实验的收获

我对于 cache 的工作方式不仅仅停留在课本上，通过实践，实现了 cache 的模拟，帮助我对于其如何处理命中和不命中，以及如何进行替换，有了更加清晰的了解；
了解了 gprof、valgrind 等工具的基本使用方法；
让我了解了矩阵转置中利用分块进行更快操作的方法。

4.2 请给出对本次实验内容的建议

在做实验时，发现测试矩阵转置时，如果将文件夹放在共享文件夹时测试会超时，但是将其放到别的地方时就可以成功测试，希望将这点加到 PPT 中，以便同学解决问题。

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.