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　计算机科学与技术

学　　号　1180300829

班　　级　1803008

学　　生　余涛

指 导 教 师　吴锐

实 验 地 点　G709

实 验 日 期　2019.11.16

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

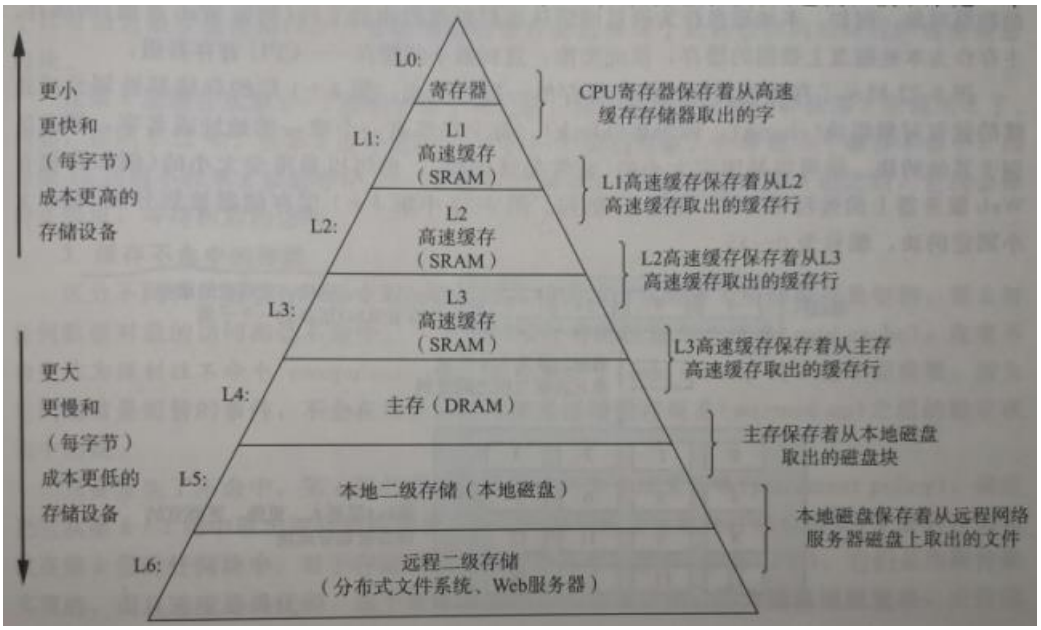
1.3 实验预习

- **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 C S E B s e b**
- **写出 Cache 的基本结构与参数**

- 写出各类 **Cache** 的读策略与写策略
- 掌握 **Valgrind** 与 **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 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）

L1: C:128KB S:128 E: 16 B:64 s:7 e:4 b:6

L2: C:512KB S:2048 E:4 B:64 s:11 e:2 b:6

L2: C:3MB S:4096 E:12 B:64 s:12 e:$\log_2 12$ b:6

2.3 写出各类 Cache 的读策略与写策略（5 分）热电池

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。

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：

本次实验要求设计一个 cache 模拟器，在输入参数 s、E、b 设置为任意值时均能正确工作，并且已经提供了该模拟器的 c 语言实现的部分代码，要求补充

void initCache()，void freeCache()，void accessData(mem_addr_t addr)的代码并在主函数用命令行方式计算 S、E、B 的值，分析模板及模板的注释可知：

1.void initCache()函数：要求为 cache 分配内存，并且用 0 来表示有效，同时标记和 LRU，还要计算 set_index_mask：

整个 cache 块分配内存可用 cache = malloc(S * sizeof(cache_set_t))；

每一组 cache 分配内存可用 cache[i] = malloc(E * sizeof(cache_line_t)；

然后全部都写为 0：cache[i][j].valid = '0';

cache[i][j].lru = 0;

cache[i][j].tag = 0;

2. void freeCache()函数：要求释放空间，则只需要把刚才 initCache()里面申请的空间全部释放即可，先释放每一组的空间，然后释放整个 cache 块的空间：

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

3. void accessData(mem_addr_t addr)函数：要求访问内存地址 addr 中的数据，

如果它已经在缓存中，就执行 hit_count++;如果它不在缓存中，就执行 miss_count++;如果一条线被驱逐，就执行 eviction_count++。

为了实现此函数，需要进行几种情况的讨论：

（1）.若命中，则在组索引中存在某一组标记匹配，并且有效位为 1，这个时候执行 hit_count++；

（2）.若不命中，此时则执行 miss_count++；

（3）.然后考虑是否被驱逐，对于驱逐来说，遍历每一组，判断当前组是否已满，若有一个有效位为 0，则说明未满，若有效位为全为 1，则需要驱逐，则执行 eviction_count++；

（4）.若某一组未满，则经有效位不是 1 的那一行更新为有效位为 1，同时更新标记位和 lru 计数值。

4.用命令行方式计算 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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 1 -E 1 -b 1 -
t traces/yi2.trace
hits:9 misses:8 evictions:6
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 4 -E 2 -b 4 -
t traces/yi.trace
hits:4 misses:5 evictions:2
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 2 -E 1 -b 4 -
t traces/dave.trace
hits:2 misses:3 evictions:1
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 2 -E 1 -b 3 -
t traces/trans.trace
hits:167 misses:71 evictions:67
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 2 -E 2 -b 3 -
t traces/trans.trace
hits:201 misses:37 evictions:29
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 2 -E 4 -b 3 -
t traces/trans.trace
hits:212 misses:26 evictions:10
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -
t traces/trans.trace
hits:231 misses:7 evictions:0
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -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）：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -
t traces/long.trace
hits:265189 misses:21775 evictions:21743
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ ./csim-ref -s 5 -E 1 -b
 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
```

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：

```
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/cachelab-handout$ make
# Generate a handin tar file each time you compile
tar -cvf yt1180300829-handin.tar  csim.c trans.c
csim.c
trans.c
yt1180300829@ubuntu:~/hitics/cachelab/cachelab-handout/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


TEST_CSIM_RESULTS=27
```

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

程序要求我们对不同规格的矩阵转置操作进行优化，分别要求：

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

需要我们的 cache 的参数为 s=5，E=1，b=5。所有对于这个缓存，一共有 32 组，每组一个块，块的大小为 32 字节，也就是说一个块里面能装入 8 个 int 型变量，cache 总共能装入 32*8=256 个 int 型的变量，我们发现待转换矩阵可以发现矩阵大小大于 cache 大小。

1.对于 32*32 的矩阵：

为了减少 A、B 两个矩阵访问相同元素产生的冲突不命中，我们可以一次性访问块的很多元素，分析得 cache 里面一次可以装入 8 行矩阵里面的值，我们可以考虑把数组分块成 8*8，然后每次转置这 8*8 块里面的元素。

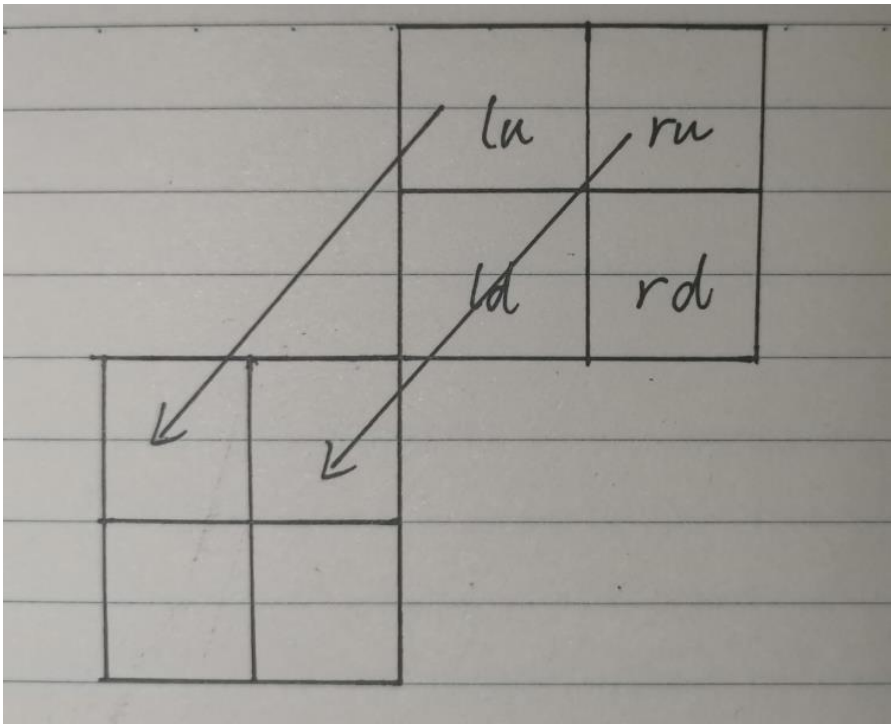
2.64*64 的矩阵：

分析可得 cache 里面一次性可以装入 4 行矩阵里面的元素，于是我首先想到用 4*4 分块的方式进行测试，发现无法达到不命中数的要求。

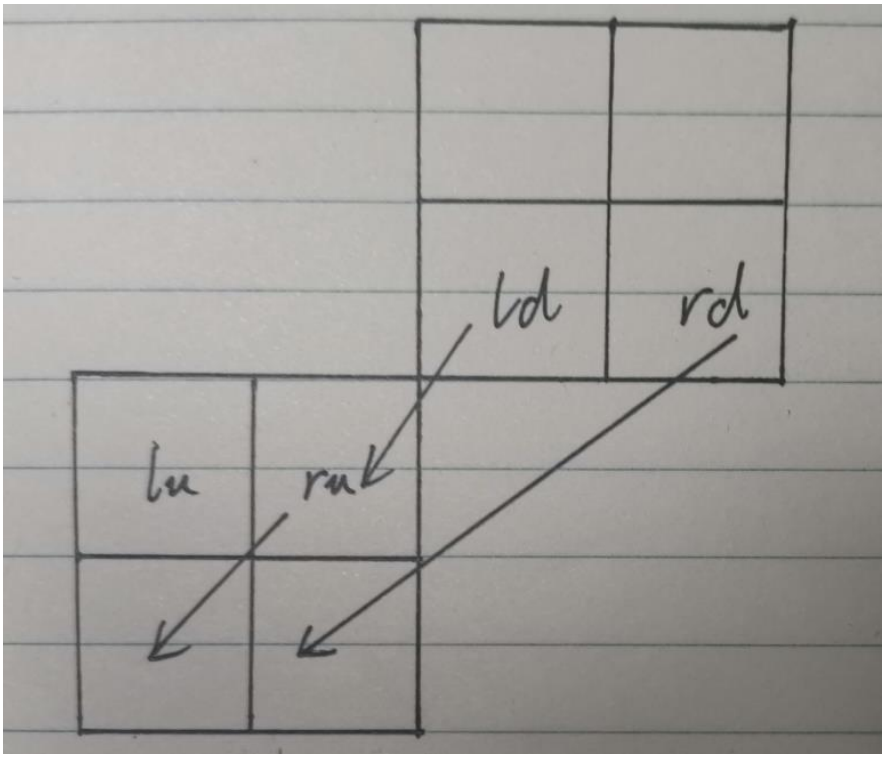
于是我准备尝试将 8*8 分块和 4*4 分块结合起来。

对 A，我们把它分成 8*8 的矩阵，然后把 8*8 自身分解成四个 4*4 的矩阵，命名为左上 lu，右上 ru，左下 ld，右下 rd。为了更好解释，用如下图解释：

首先将 lu 移动到箭头指向位置，然后将 ru 移动到箭头指向位置：



再将 ru 移动到 lu 下面，然后将 ld 和 rd 移动到箭头指向位置：



3.61*67 的矩阵：

直接使用简单的分块操作，设置一个可以改变的大小，从 4 开始改变，发现 23 时 miss 最小，可行

32×32（10 分）：运行结果截图

```
yt1180300829@ubuntu:~/Desktop/cachelab-handout$ ./test-trans -M 32 -N 32

Function 0 (1 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255

Summary for official submission (func 0): correctness=1 misses=287

TEST_TRANS_RESULTS=1:287
```

64×64（10 分）：运行结果截图

```
yt1180300829@ubuntu:~/Desktop/cachelab-handout$ ./test-trans -M 64 -N 64

Function 0 (1 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:9066, misses:1179, evictions:1147

Summary for official submission (func 0): correctness=1 misses=1179

TEST_TRANS_RESULTS=1:1179
```

61×67（20 分）：运行结果截图

```
yt1180300829@ubuntu:~/Desktop/cachelab-handout$ ./test-trans -M 61 -N 67

Function 0 (1 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:6251, misses:1928, evictions:1896

Summary for official submission (func 0): correctness=1 misses=1928

TEST_TRANS_RESULTS=1:1928
```

第 4 章 总结

4.1 请总结本次实验的收获

了解了缓存的相关结构及知识；

对缓冲命中的原理有了深入理解；

学会了通过对代码的优化实现增加缓存命中率的方法。

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.