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　计算机系

学　　号　1180300811

班　　级　1803008

学　　生　孙骁

指 导 教 师　吴锐

实 验 地 点　G712

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

1. 理解现代计算机系统存储器层级结构
2. 掌握 Cache 的功能结构与访问控制策略
3. 培养 Linux 下的性能测试方法与技巧
4. 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

i7-8550U X64 CPU；1.80GHz；16G RAM；1T SSD

1.2.2 软件环境

Windows10 64 位；Vmware 15.1.0；Ubuntu 18.04 LTS

1.2.3 开发工具

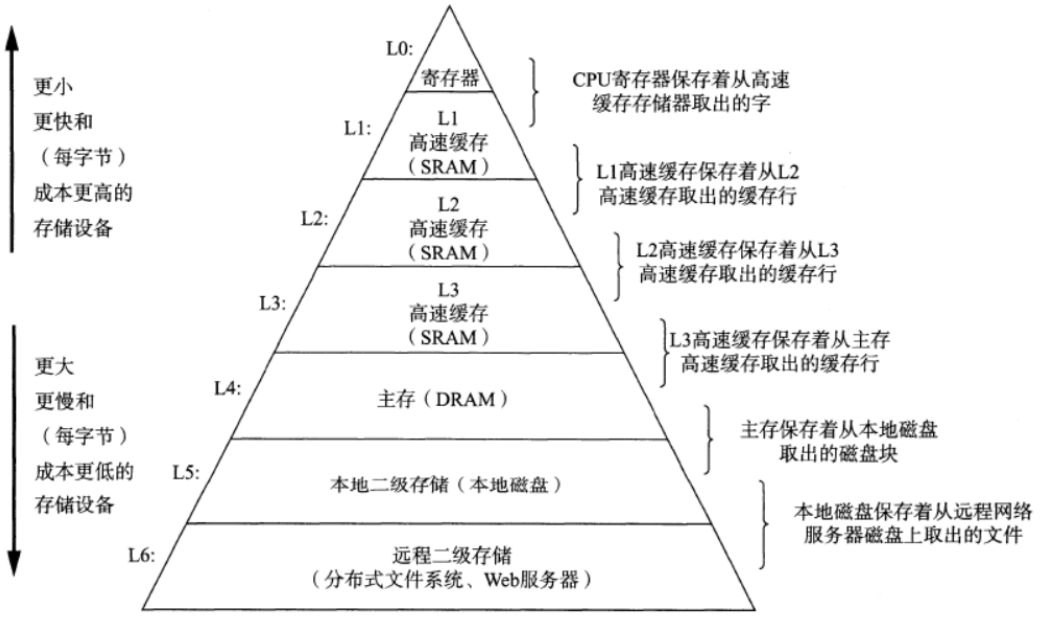
Visual Studio 2019 ；CodeBlocks；gcc

1.3 实验预习

1. 上实验课前，必须认真预习实验指导书（PPT 或 PDF）
2. 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。
3. 画出存储器的层级结构，标识其容量价格速度等指标变化
4. 用 CPUZ 等查看你的计算机 Cache 各参数，写出 C S E B s e b
5. 写出 Cache 的基本结构与参数
6. 写出各类 Cache 的读策略与写策略
7. 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）



L0：寄存器

L1：高速缓存

L2：高速缓存

L3：高速缓存

L4：主存

L5：本地二级存储

L6：远程二级储存

随着层级的增加，容量增大，速度更慢，成本更低

随着层级的减少，容量减小，速度更快，成本更高

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 S=64 E=8 B=64 s=6 b=6

二级数据缓存：C=32KB S=64 E=8 B=64 s=6 b=6

三级数据缓存：C=3MB S=4096 E=12 B=64 s=12 b=6

2.3 写出各类 Cache 的读策略与写策略（5 分）

读策略：

若要读取 k+1 层的数据，先到第 k 层寻找，如果第 k 层缓存中有该数据所在的块，直接传给 CPU；如果缓存不命中，则需要把 k+1 层中的数据传给 k 层，此时 CPU 等待。当被请求的块到达内存时，高速缓存器将这行放在对应的告诉缓存行中，根据偏移量读取数据，返回 CPU。

写策略：

1. 写回法：

   当 CPU 写 cache 命中时，只修改 cache 内容，而不是直接写入主存，当此内存块被替换时才会写回主存。

   此时 cache 更新与写主存异步进行，存在数据不一致的隐患。为此，每个 cache 必须配置一个修改位，以检测此块是否被修改过。

2. 全写法：

   当写 cache 命中时，cache 与主存同时发生写修改。

此时写 cache 与写主存同步进行。此时 cache 不需要设置修改位，以及相应的判断逻辑，但也相当于写回操作中无 cache 的功能。功效较低。

3. 写一次法：

基于写回法与全写法的写操作策略，写命中与写未命中的处理方法与写回法基本相同，只是第一次写命中时要同时写入主存，以维护系统全部 cache 的一致性。

2.4 写出用 gprof 进行性能分析的方法（5 分）

1. 使用 gcc、g++、xlc 编译程序时，使用-pg 参数，如：gcc -pg -o test test.c 编译器会自动在目标代码中插入用于性能测试的代码片段，会在程序运行时采集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执行时间。
2. 执行编译后的可执行程序，如：./test 程序运行结束后，会在程序路径下生成一个缺省文件为 gmon.out 的文件，此文件记录了程序运行的性能、调用关系、调用次数等信息数据。
3. 使用 gprof 命令分析记录程序运行信息的 gmon.out 文件，如：gprof test gmon.out 可以在显示器上看到函数调用相关的统计、分析信息。也可以采用：gprof test gmon.out > gprofresult.txt 重定向到文本文件进行后续分析。

2.5 写出用 Valgrind 进行性能分析的方法（5 分）

Valgrind 工具详解

1.Memcheck

最常用的工具，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。所以，它能检测以下问题：

1、对未初始化内存的使用；

2、读/写释放后的内存块；

3、读/写超出 malloc 分配的内存块；

4、读/写不适当的栈中内存块；

5、内存泄漏，指向一块内存的指针永远丢失；

6、不正确的 malloc/free 或 new/delete 匹配；

7、memcpy()相关函数中的 dst 和 src 指针重叠。

2.Callgrind

和 gprof 类似的分析工具，但它对程序的运行观察更是入微，能给提供更多的信息。和 gprof 不同，它不需要在编译源代码时附加特殊选项，但加上调试选项是推荐的。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。

3.Cachegrind

Cache 分析器，它模拟 CPU 中的一级缓存 I1，D1 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。这对优化程序有很大的帮助。

4.Helgrind

它主要用来检查多线程程序中出现的竞争问题。Helgrind 寻找内存中被多个线程访问，而又没有一贯加锁的区域，这些区域往往是线程之间失去同步的地方，而且会导致难以发掘的错误。Helgrind 实现了名为“Eraser”的竞争检测算法，并做了进一步改进，减少了报告错误的次数。

5. Massif

堆栈分析器，它能测量程序在堆栈中使用了多少内存，反映堆块，堆管理块和栈的大小。Massif 能帮助减少内存的使用，在带有虚拟内存的现代系统中，它还能够加速程序的运行，减少程序停留在交换区中的几率。

Massif 对内存的分配和释放做 profile。程序开发者通过它可以深入了解程序的内存使用行为，从而对内存使用进行优化。这个功能对 C++尤其有用，因为 C++有很多隐藏的内存分配和释放

三 使用 Valgrind

Valgrind 使用起来非常简单，不需要重新编译你的程序就可以使用。

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：

选项	作用

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=<*toolname*> [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=<yes|no> [default: no]	绑定到调试器上，便于调试错误。

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：

模拟 cache 的过程包括设计 cache 的结构，分为组，行。每组中考虑有效位、标志位、缓存块。通过比较标志位模拟是否命中，用 lru 模拟 cache 各行的优先级，最后对 cache 进行数据的更新。统计 hit，miss，eviction 的数量。

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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
```

测试用例 1 的输出截图（5 分）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/cachelab-handout$ ./csim-ref -v -s 1 -E 1 -b 1 -t traces/yi2.trace
L 0,1 miss
L 1,1 hit
L 2,1 miss
L 3,1 hit
S 4,1 miss eviction
L 5,1 hit
S 6,1 miss eviction
L 7,1 hit
S 8,1 miss eviction
L 9,1 hit
S a,1 miss eviction
L b,1 hit
S c,1 miss eviction
L d,1 hit
S e,1 miss eviction
M f,1 hit hit
hits:9 misses:8 evictions:6
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/cachelab-handout$ ./csim-ref -v -s 4 -E 2 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:2
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/cachelab-handout$ ./csim-ref -v -s 2 -E 1 -b 4 -t traces/dave.trace
L 10,4 miss
S 18,4 hit
L 20,4 miss
S 28,4 hit
S 50,4 miss eviction
hits:2 misses:3 evictions:1
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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）：

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/cachelab-handout$ ./csim -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

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

针对 32×32 的矩阵，因为 cache 每行可以放入 8 个 int 类型，基本思想是将矩

阵分成 8×8 的块，这样可以最大限度的利用 cache。

针对 64×64 的矩阵，分成 8×8 的时候，写入的时候第一行后四列写入第一列后四行时会发生冲突，所以采取 4×4 的分块策略。

针对不规则矩阵，直接采取试探性写法，在 16 17 18 19 分块结果中，18 的效果最好，采取 18×18 的分块策略。

32×32（10 分）：运行结果截图

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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

```
demerzel@demerzel-virtual-machine:~/ccode/lab6/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:6354, misses:1825, evictions:1793

Summary for official submission (func 0): correctness=1 misses=1825

TEST_TRANS_RESULTS=1:1825
```

第 4 章 总结

4.1 请总结本次实验的收获

本次实验让我对于 cache 的结构、运行机制有了比较全面的认识，明白了可以通过设计对 cache 友好的程序来提高 cache 的性能，对于计算机底层的实现了解更多了。

4.2 请给出对本次实验内容的建议

本次实验中如何编译执行 csim.c 以及 trans.c 的过程，实验指导书上说明不清，不少同学不知道通过链接 cachelab.c 与 cachelab.h 即可，很多人直接在 csim.c 中自己编写的 printSummary 函数，浪费了不少时间，希望实验指导的 ppt 能增加此部分的讲解。

参考文献

[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.