哈爾濱工業大學

实验报告

实 验（五）

题　　目 Cachelab

高速缓冲器模拟

专　　业 计算机类

学　　号 1160300901

班　　级 1603009

学　　生 孙月晴

指 导 教 师 吴锐

实 验 地 点 G712

实 验 日 期 2017-12-05

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

X64 CPU；2GHz；2G RAM；256GHD Disk

1.2.2 软件环境

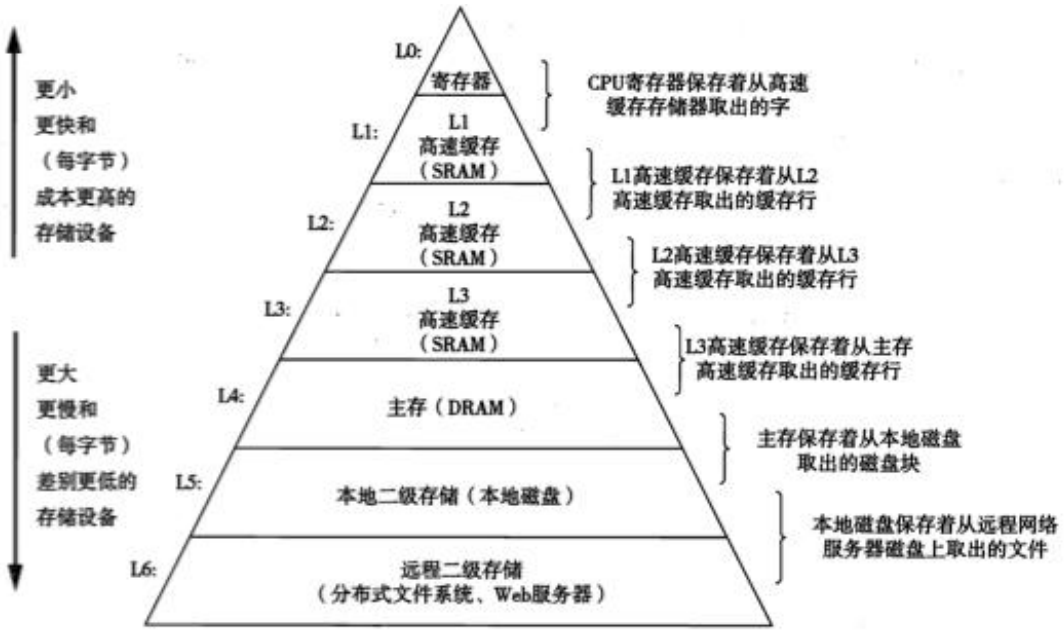
Windows10 64 位；Vmware 12；Ubuntu 16.04 LTS 64 位

1.2.3 开发工具

Visual Studio 2010 64 位；TestStudio；Gprof;Valgrind 等

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）

一级缓存：32KB 8 路组相联，64 字节每块。

C(字节)	S	E	B	s	b
32768	64	8	64	6	6

二级缓存：256KB 4 路组相联，64 字节每块。

C(字节)	S	E	B	s	b
262144	1024	4	64	10	6

三级缓存：3MB 12 路组相联，64 字节每块。

C(字节)	S	E	B	s	b
3145728	4096	12	64	12	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

读策略：
1）缓存命中：直接从该层读取数据
2）缓存不命中：替换策略，其中“随机替换策略”会随机选择一个牺牲块；“最近最少被使用（LRU）替换策略”会选择那个最后被访问的时间距现在最远的块。
写策略：1）写命中 2）写不命中
1）写命中：a. 直写，就是立即将已经缓存了的字的高速缓存块写回到紧接着的低一层中；b. 写回：尽可能地推迟更新，只有当替换算法要驱逐这个更新过的块时，才把它写到紧接着的低一层中。
2）写不命中:a. 写分配，加载相应的低一层中的块到高速缓存中，然后更新这个高速缓存块；b. 非写分配，避开高速缓存，直接把这个字写到低一层中。
直写高速缓存通常是非写分配的，写回高速缓存通常是写分配的。

2.4 写出用 gprof 进行性能分析的方法（5 分）

在编译时加上-pg 选项，编译器就会在编译程序时在每个函数的开头加一个 mcount 函数调用，在每一个函数调用之前都会先调用这个 mcount 函数，在 mcount 中会保存函数的调用关系图和函数的调用时间和被调次数等信息。最终在程序退出时保存在 gmon.out 文件中，需要注意的是程序必须是正常退出或者通过 exit 调用退出，因为只要在 exit（）被调用时才会触发程序写 gmon.out 文件。

那么，gprof 的使用方法主要以下三步：

- 用-pg 参数编译程序
- 运行程序，并正常退出
- 查看 gmon.out 文件

2.5 写出用 Valgrind 进行性能分析的方法（5 分）

Valgrind 工具包包含多个工具，如 Memcheck,Cachegrind,Helgrind,Callgrind，Massif。
用法: valgrind [options] prog-and-args [options]: 常用选项，适用于所有 Valgrind 工具
-tool=<name> 最常用的选项。运行 valgrind 中名为 toolname 的工具。默认 memcheck。
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 更详细的信息, 增加错误数统计。
-trace-children=no|yes 跟踪子线程? [no]
-track-fds=no|yes 跟踪打开的文件描述？ [no]
-time-stamp=no|yes 增加时间戳到 LOG 信息? [no]
-log-fd=<number> 输出 LOG 到描述符文件 [2=stderr]
-log-file=<file> 将输出的信息写入到 filename.PID 的文件里，PID 是运行程序的进行 ID
-log-file-exactly=<file> 输出 LOG 信息到 file
-log-file-qualifier=<VAR> 取得环境变量的值来做为输出信息的文件名。 [none]
-log-socket=ipaddr:port 输出 LOG 到 socket ，ipaddr:port

LOG 信息输出：
-xml=yes 将信息以 xml 格式输出，只有 memcheck 可用
-num-callers=<number> show <number> callers in stack traces [12]
-error-limit=no|yes 如果太多错误，则停止显示新错误? [yes]
-error-exitcode=<number> 如果发现错误则返回错误代码 [0=disable]
-db-attach=no|yes 当出现错误，valgrind 会自动启动调试器 gdb。[no]
-db-command=<command> 启动调试器的命令行选项[gdb -nw %f %p]

适用于 Memcheck 工具的相关选项：
-leak-check=no|summary|full 要求对 leak 给出详细信息? [summary]
-leak-resolution=low|med|high how much bt merging in leak check [low]
-show-reachable=no|yes show reachable blocks in leak check? [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：我使用了老师提供的代码框架，所以只有三个函数需要完成，即初始化 cache：void initCache()、释放 cache:：void freeCache()、访问内存地址 addr 处的数据：void accessData(mem_addr_t addr)。框架中大多变量都定义为了全局变量，函数不再传参。

1. 函数 void initCache()：根据 S（组的数目）申请一个数组，该数组元素是 cache_line_t 即每一行的入口的指针。接着循环 S 次每次申请 E 个 cache_line_t 数据结构，并让刚刚的指针数组的元素指向它们。

2. 函数 void freeCache()：每组释放，每行释放。

3. void accessData(mem_addr_t addr)：遍历搜索，在 set 中完整遍历一遍，找到时间参量最小（应该被替换的 line）的值及其对应的 line，并找到最新的时间参量，用来替换时更新那个最老的时间参量。找到最老的时间参量，更新对应的 line，找到最新的时间参量，将其+1 作为更新的数。

测试用例 1 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 1 -E 1
 -b 1 -t traces/yi2.trace
DEBUG: S:2 E:1 B:2 trace:traces/yi2.trace
DEBUG: set_index_mask: 1
hits:9 misses:8 evictions:6
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 2 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 4 -E 2
 -b 4 -t traces/yi.trace
DEBUG: S:16 E:2 B:16 trace:traces/yi.trace
DEBUG: set_index_mask: 15
hits:4 misses:5 evictions:2
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 3 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 2 -E 1
 -b 4 -t traces/dave.trace
DEBUG: S:4 E:1 B:16 trace:traces/dave.trace
DEBUG: set_index_mask: 3
hits:2 misses:3 evictions:1
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 4 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 2 -E 1
 -b 4 -t traces/dave.trace
DEBUG: S:4 E:1 B:16 trace:traces/dave.trace
DEBUG: set_index_mask: 3
hits:2 misses:3 evictions:1
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 2 -E 1
 -b 3 -t traces/trans.trace
DEBUG: S:4 E:1 B:8 trace:traces/trans.trace
DEBUG: set_index_mask: 3
hits:167 misses:71 evictions:67
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 5 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 2 -E 2
 -b 3 -t traces/trans.trace
DEBUG: S:4 E:2 B:8 trace:traces/trans.trace
DEBUG: set_index_mask: 3
hits:201 misses:37 evictions:29
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 6 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 2 -E 4
 -b 3 -t traces/trans.trace
DEBUG: S:4 E:4 B:8 trace:traces/trans.trace
DEBUG: set_index_mask: 3
hits:212 misses:26 evictions:10
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 7 的输出截图（5 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 5 -E 1
 -b 5 -t traces/trans.trace
DEBUG: S:32 E:1 B:32 trace:traces/trans.trace
DEBUG: set_index_mask: 31
hits:231 misses:7 evictions:0
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

测试用例 8 的输出截图（10 分）：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./csim -s 5 -E 1
 -b 5 -t traces/long.trace
DEBUG: S:32 E:1 B:32 trace:traces/long.trace
DEBUG: set_index_mask: 31
hits:265189 misses:21775 evictions:21743
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

总截图：

```
syq1160300901@syq1160300901-virtual-machine:~/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
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

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：题目给的 Cache 大小只有 1024B，s=5，E=1，b=5。给的数据大小分别是 32×32,64×64,61×67，要求只能只用 12 个 int 变量，miss 越小越好。

1. 32×32

为使 miss 次数在 300 以下，首先，Cache 的一个块只有 32B，也就是只能容纳 8 个 int。这个 Cache 可以容纳这个 matrix 的前 8 行。分块取 8×8。先读取 A 的一行，然后放入 B 的一列。12 个 int 变量，4 个用来循环，其余 8 个用来存 A 中块的一行。对于 32 × 32 的矩阵，总共存在 1024 次读和 1024 次写。对于非对角线的分块（总共 12 个），其缓存不命中率是 1/8（仅强制不命中），对于对角线的分块（总共 4 个），其写的缓存不命中率是 1/8（强制不命中），其读的缓存不命中率为 1/4（强制不命中和冲突不命中各一半）。因此，理论上优化之后的总缓存不命中数为：

2048 × 0.75 × 0.125 + 1024 × 0.25 × 0.125 + 1024 × 0.25 × 0.25 = 288 次。

2. 64×64

用 8×8 的块来做，虽然 A 数组不能变换，但是 B 数组可以任意操作。先把数字移动到 B 中，然后在 B 中做变化。考虑用同样的 miss 次数，把更多的数据移动到 B 中，但是不一定是正确的位置，然后再用同样的 miss 次数，把 A 中部分数据移动到 B 中时，完成把 B 中前面位置错误数据的纠正。

3. 61×67

不规则的 matrix，本质也是用分块来优化 Cache 的读写，但是不能找到比较显然的规律看出来间隔多少可以填满一个 Cache，但是要求比较松，可以尝试一些分块的大小，直接进行转置操作。尝试到 16 左右 ，可以小于 2000 次 miss

32×32（10 分）：运行结果截图

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./test-trans -M32 -N32

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
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

64×64（10 分）：运行结果截图

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./test-trans -M64 -N64

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:9026, misses:1219, evictions:1187

Summary for official submission (func 0): correctness=1 misses=1219

TEST_TRANS_RESULTS=1:1219
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

61×67（20 分）：运行结果截图

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./test-trans -M61 -N67

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:6187, misses:1992, evictions:1960

Summary for official submission (func 0): correctness=1 misses=1992

TEST_TRANS_RESULTS=1:1992
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

总截图：

```
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$ ./driver.py
Part A: Testing cache simulator
Running ./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


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         287
Trans perf 64x64           8.0         8        1219
Trans perf 61x67          10.0        10        1992
          Total points    53.0        53
syq1160300901@syq1160300901-virtual-machine:~/cachelab-handout$
```

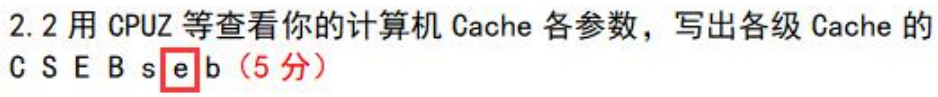
第 4 章 总结

4.1 请总结本次实验的收获

1. 编写 Cache 模拟器比较简单，主要是模拟出 Cache 的功能。优化矩阵转置操作比较有难度，思考了很久，同时在做出来之后，还通过定量的计算 miss 次数，和实验结果接近，让我对 Cache 理解更深了。Cache 这套理论非常重要，要在以后编程中编写对 Cache 友好的代码，我还有很多路需要走。
2. 这次实验中我认为比较重要/难的地方：一是对于缓存的理解以及矩阵元素在缓存中的排布问题；二是位于对角线上的分块不仅是内部而且在分块间存在的冲突不命中问题；三是虽然是矩阵，但是也要明确统一控制行列的变量。

4.2 请给出对本次实验内容的建议

1. e 没有实际意义



2. 可以适当减少缺失率

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.