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

学　　号　1190202105

班　　级　1903002

学　　生　傅浩东

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

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；

1 TB SSD

1.2.2 软件环境

Windows 10 21H1；VirtualBox；Ubuntu 20.04 LTS

1.2.3 开发工具

EDB；GDB；CodeBlocks；vi/vim/gpedit+gcc；VSCode

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

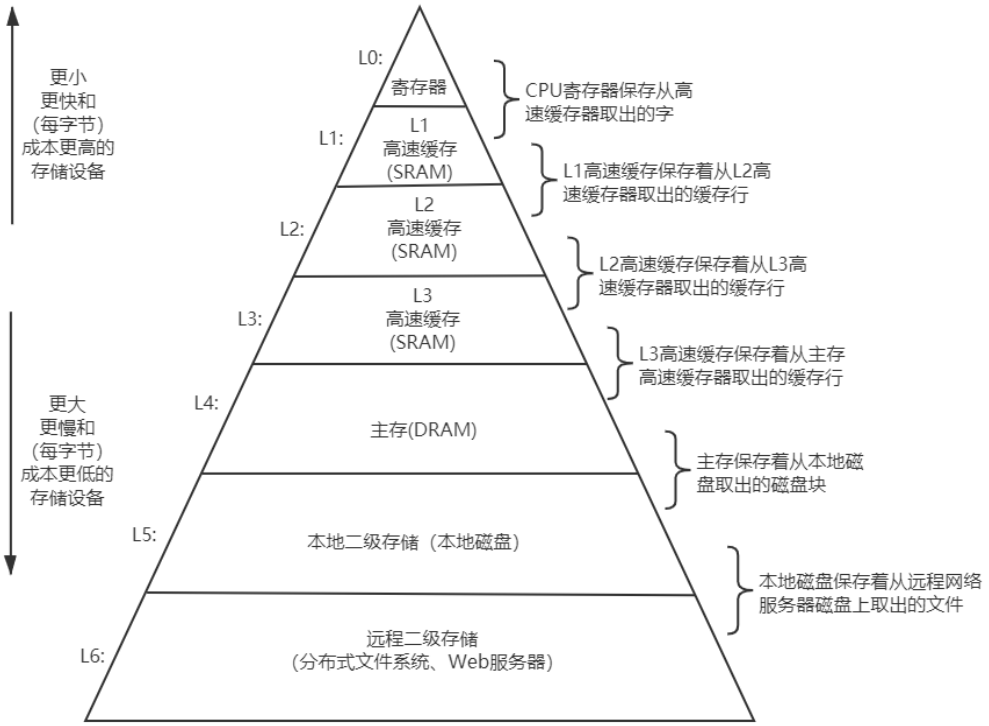
用 CPUZ 等查看你的计算机 Cache 各参数，写出 Cache 的基本结构与参数：缓存大小 C、分组数量 S、关联度/组内行数 E、块大小 B，及对应的编码位数 ：组索引位数 s、 e 、块内偏移位数 b

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 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）

	C	S	E	B	s	e	b
L1	384KB	384	16	64	9	4	6
L2	1536KB	6144	4	64	13	2	6
L3	12MB	12288	16	64	14	4	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. 命中
   (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：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. 不命中
   (1) 写分配：载入到缓存中，更新缓存。先写数据，再为所写数据分配 cache line；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) 非写分配：直接写入到内存，不载入缓存。不分配 cache line 给被修改的数据，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. 编译程序时使用-pg 参数，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。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，根据产生的 profile 可以产生不同实行的分析输出。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。

2.5 写出用 Valgrind 进行性能分析的方法（5 分）

```
ifu@ifu-VirtualBox:cachelab-handout$ valgrind --tool=memcheck ./tracegen
==2970== Memcheck, a memory error detector
==2970== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==2970== Using Valgrind-3.17.0 and LibVEX; rerun with -h for copyright info
==2970== Command: ./tracegen
==2970==
==2970==
==2970== HEAP SUMMARY:
==2970==     in use at exit: 0 bytes in 0 blocks
==2970==   total heap usage: 2 allocs, 2 frees, 4,568 bytes allocated
==2970==
==2970== All heap blocks were freed -- no leaks are possible
==2970==
==2970== For lists of detected and suppressed errors, rerun with: -s
==2970== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ifu@ifu-VirtualBox:cachelab-handout$
```

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：

1. 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）
2. 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（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）
4. Helgrind（用来检查多线程程序中出现的竞争问题）
5. 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] 。在 Linux 下使用命令行 valgrind –tool=callgrind execname 会在当前目录下生成文件，文件名为“callgrind.out.进程号”。生成的文件可以转化为 dot 文件，再转化为图片。

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. void initCache()

初始化 cache，按要求用 malloc 为 cache 分配内存，将 valid、tag 和 LRU 初始为 0（valid 初始化为 ’0’），最后计算 set_index_mask 这个全局变量。

为整个 cache 块分配内存：cache = (cache_t) malloc(sizeof (cache_set_t) * S);

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

计算 set_index_mask = (1 << s) – 1;

2. void freeCache()

释放分配的内存，首先要先释放每一组 cache 内存，然后释放整个 cache 块。

3. void accessCache(mem_addr_t addr)

访问 addr 的数据，若在缓存中即命中则 hit_count++，若不命中 miss_count++，若一条线被驱逐 eviction_count++。

4. Compute S, E and B from command line args

S = 1 << s; B = 1 << b; E = E;

```
ifu@ifu-VirtualBox:cachelab-handout$ make clean
rm -rf *.o
rm -f *.tar
rm -f csim
rm -f test-trans tracegen
rm -f trace.all trace.f*
rm -f .csim_results .marker
ifu@ifu-VirtualBox:cachelab-handout$ make
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c
gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o

gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c
# Generate a handin tar file each time you compile
tar -cvf ifu-handin.tar  csim.c trans.c
csim.c
trans.c
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 1 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 2 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 3 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 4 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 5 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 6 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 7 的输出截图（5 分）：

```
ifu@ifu-VirtualBox:cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
ifu@ifu-VirtualBox:cachelab-handout$ ./csim-ref -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
ifu@ifu-VirtualBox:cachelab-handout$
```

测试用例 8 的输出截图（10 分）：

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

```
ifu@ifu-VirtualBox: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
ifu@ifu-VirtualBox:cachelab-handout$
```

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

1. 32×32 矩阵

对于 int 类型，我们知道每个 cache line 可存放 8 个数据，缓存可装进 8 行数据。那么对于大矩阵，将其划分为 8×8 的小矩阵，那么位于同一列的小矩阵，相差 8 行的数字在 cache 中是属于同一块的。依次访问当前行 8 个 int，放入局部变量，再放入转置矩阵。

2. 64×64 矩阵

对于 64×64 矩阵，每行需要 8 个缓存块，每 4 行 index 会重复一次，因此不能采用 8×8 分块，若采用 4×4 分块没用充分利用 cache。于是将 8×8 和 4×4 分块结合使用。8×8 分块后，遍历 A 前四行，每个 4×4 块进行转置，再将第二块转置矩阵移动到第一块下面。重复上述过程直到结束。

3. 61×67 矩阵

仍采取分块的方法，但是 61 和 67 都不是 8 的整数倍，找不到相差 N 行对应同样的块，尝试使用 16×16 时发现符合条件，小于 2000.

32×32（10 分）：运行结果截图

```
ifu@ifu-VirtualBox: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:1766, misses:289, evictions:257

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

Summary for official submission (func 0): correctness=1 misses=289

TEST_TRANS_RESULTS=1:289
ifu@ifu-VirtualBox:cachelab-handout$
```

64×64（10 分）：运行结果截图

```
ifu@ifu-VirtualBox: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:9002, misses:1245, evictions:1213

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3474, misses:4723, evictions:4691

Summary for official submission (func 0): correctness=1 misses=1245

TEST_TRANS_RESULTS=1:1245
ifu@ifu-VirtualBox:cachelab-handout$
```

61×67（20 分）：运行结果截图

```
ifu@ifu-VirtualBox: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:6363, misses:1818, evictions:1786

Function 1 (2 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:3756, misses:4423, evictions:4391

Summary for official submission (func 0): correctness=1 misses=1818

TEST_TRANS_RESULTS=1:1818
ifu@ifu-VirtualBox:cachelab-handout$
```

全部过程运行结果：

```
ifu@ifu-VirtualBox: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         289
Trans perf 64x64           8.0         8        1245
Trans perf 61x67          10.0        10        1818
          Total points    53.0        53
ifu@ifu-VirtualBox:cachelab-handout$
```

第 4 章 总结

4.1 请总结本次实验的收获

1. 进一步了解 Cache 的读写策略，以及储存的层次结构；
2. 了解了 Cache 的功能结构与访问控制策略；
3. 理解 Cache 组成结构对 C 程序性能的影响；
4. 程序在共享文件夹下运行存在问题，运行速度异常慢。

4.2 请给出对本次实验内容的建议

1. 进一步介绍 valgrind 的使用方法

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.