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 1190200122

班　　级 1903001

学　　生 袁野

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2020-5-28

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

Legion Y7000P 2019 PG0
CPU:Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz (12 CPUs), ~2.6GHz
RAM: 16384MB

1.2.2 软件环境

Windows 10 家庭中文版 64-bit
Ubuntu 20.04.2 LTS
VMware® Workstation 16 Player 16.1.0 build-17198959

1.2.3 开发工具

Microsoft Visual Studio Community 2019 版本 16.9.2
Microsoft Visual 1.54.3
GCC 9.3.0

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。

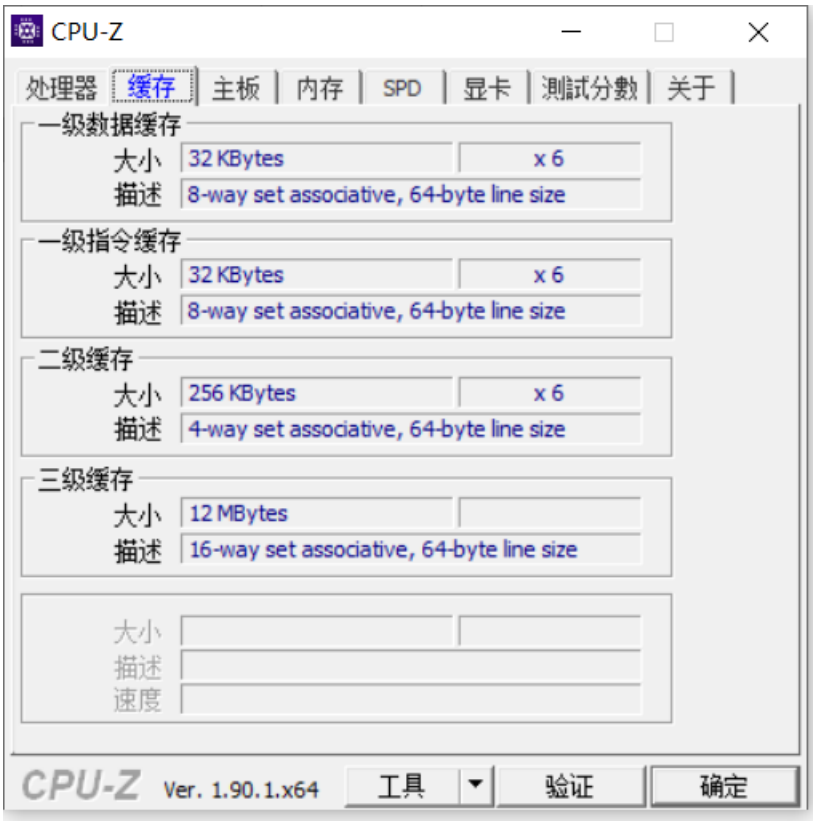
- 画出存储器的层级结构，标识其容量价格速度等指标变化
- 用 CPUZ 等查看你的计算机 Cache 各参数，写出 Cache 的基本结构与参数：缓存大小 C、分组数量 S、关联度/组内行数 E、块大小 B，及对应的编码位数 ：组索引位数 s、 e 、块内偏移位数 b
- 写出 Cache 的各种读策略与写策略
- 掌握 Valgrind、gprof 的使用方法

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	12*32KB	64	8	64	6	3	6
L2	6*256KB	1024	4	64	10	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. 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。

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。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] 。一些常用的选项如下：

选项	作业
-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]	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：

阅读 csim-codeframe.c 我们大致可以明白代码中的一些主要信息：ADDRESS_LENGTH 为地址的长度，cache_line_t 为一个高速缓存行，其中包含的 valid 表示是否有效，tag 表示标识位，lru 表示上一次访问该高速缓存行的时间戳。cache_set_t 表示一个高速缓存组，cache_t 表示一个高速缓存。紧接着我们需要补全 initCache()、freeCache()、accessCache()三个函数。

1、 initCache()：

首先我们可以根据输入的 s 和 b 求出 S 和 B，紧接着我们需要进行内存分配：

cache = malloc(sizeof(cache_set_t) * S); 为 cache 分配 S 个高速缓存组。

cache[i] = malloc(sizeof(cache_line_t) * E); 为每一个高速缓存组分配 E 个高速缓存行。

然后对每一个高速缓存行进行初始化：

```
cache[i][j].lru = 0;
cache[i][j].tag = 0;
cache[i][j].valid = 0;
```

2、 freeCache()：

首先释放每一个高速缓存组，然后释放整个 cache。

3、 accessData()：

根据给定的地址求出该地址对应的 tag 和 index，tag 地址的高（64-s-b）位，index 位 tag 后 s 位:

```
mem_addr_t index = ((addr >> b) & ((1 << s) - 1));
mem_addr_t tag = addr >> (s+b);
```

然后我们根据 index 找到对应的高速缓存组 now，并且让时间戳加一。我们随后在该高速缓存组中枚举每一个高速缓存行，并查看对应的 valid 是否为 1 且 tag 是否相等，符合上述条件即为命中，然后将命中的计数器+1，更新当前时间戳并且返回。

```
for (i=0; i<E; ++i) {
    if (!now[i].valid || now[i].tag != tag) continue;
    hit_count++;
    now[i].lru = lru_counter;
    return;
}
```

若未命中，则我们需要采取策略来向主存中获取所需数据块。首先枚举 now 的每一个高速缓存行，如果有高速缓存行的 valid 为 0，那么我们就将该高速缓存行内的数据复制成为当前需要的数据，即将 valid 赋值为 1，tag 赋值为传入地址的 tag，lru 赋值为当前的时间戳。

```
for (i=0; i<E; ++i) {
        if (now[i].valid) continue;
        now[i].valid = 1;
        now[i].tag = tag;
        now[i].lru = lru_counter;
        return;
    }
```

最后如果空的槽没有找到，那么我们就需要根据 LRU 的原则进行

驱逐。即我们枚举 now 中的每一个高速缓存行，并在其中寻找时间戳最小的高速缓存行，将该高速缓存行的数据进行替换即可，将 tag 和 lru 分别替换为该地址的 tag 和当前的时间戳。

```
int min_id = 0;
for (i=1; i<E; ++i) if (now[min_id].lru > now[i].lru) min_id = i;
now[min_id].lru = lru_counter;
now[min_id].tag = tag;
```

测试用例 1 的输出截图（5 分）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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）：

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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

```
yuanye@1190200122-yuanye:/mnt/hgfs/d/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：

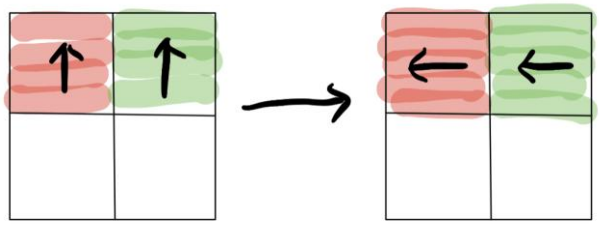
1、32*32：

由 ppt 我们可知 cache 的参数为 s=5，E=1，b=5，也就是 cache 有 32 个组，每组有块，每块内存 32B 的信息，即 8 个 int，因此我们可以将整个矩阵分块的宽度定为 8，同时为了写 B 时减少 miss，我们也需要将分块的长度定为 8，也就是说，我们将整个矩阵分为 16 个 4*4 的矩阵分别进行转置，同时，当块位于对角线上时，A[i][j]和 B[i][j]映射向的组编号相同导致冲突，为了尽可能减少这样带来的不必要的开销，我们可以先将 A 每行 8 个元素存放到中间变量中，再将他们放到 B 的每一列中。这样一来，A 数组每隔 8 次就会有一次 miss，而对于未在对角线上的 B 数组的块前 8 次全部 miss，后 24 次全部 hit，在对角线上的 B 数组的块每一行第一次访问的时候会 miss，而当访问到 A 数组相同标号的一行时会将其驱逐，以至于下一次访问 B 数组该行时会 miss，也就是 B 数组前 7 行会有 2 次 miss，第 8 行会有 1 次 miss。经过计算，miss 次数为 287。

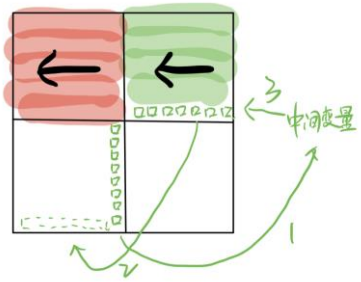
2、64*64：

此时如果我们还采用 8*8 的分块方式的话，由于我们的矩阵边长扩大了两倍，因此 cache 最多只能存放四行矩阵，这就导致了访问 A 和 B 的一块前四

行与后四行发生冲突，从而大大降低了命中率，而如果采用 4*4 分块，虽然可以降低冲突的发生，但是 catch 的使用率只有一半，miss 数量仍然达不到要求，这样的话我们可以考虑将两种方式结合。我们仍然以 8*8 的矩阵为块大小，但是对于其四个 4*4 的子矩阵我们采用不同的操作。我们首先考虑对上半部分 4*8 的矩阵进行转置，且 A 左上角放置于 B 的左上角，A 的右上角放置于 B 的右上角（暂时是错误位置）。



然后时减少 miss 的重要一步，我们依次取出 A 左下角的第 i 列四个数字以及 B 右上角的第 i 行四个数字， 将他们放置在中间变量中，然后再将 A 中取出的数字放置到 B 右上角（绿色区域）正确位置中，此时由于之前已经访问过这些位置，因而它们现在在仍在 cache 中，不会发生 miss，再将 B 中右上角之前取出的暂时放错的四个数字放到 B 左下角的正确位置上。



最后我们将 A 的右下角转置为 B 的右下角即可。

3、61*67

我们枚举块大小进行尝试，发现当块大小为 19*19 时，miss 次数小于 2000。

32×32（10 分）：运行结果截图

```
yuanye@1190200122-yuanye:~/Desktop/aa$ ./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:287, evictions:255

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=287

TEST_TRANS_RESULTS=1:287
```

64×64（10 分）：运行结果截图

```
yuanye@1190200122-yuanye:~/Desktop/aa$ ./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:9074, misses:1171, evictions:1139

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=1171

TEST_TRANS_RESULTS=1:1171
```

61×67（20 分）：运行结果截图

```
yuanye@1190200122-yuanye:~/Desktop/aa$ ./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:6200, misses:1979, evictions:1947

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=1979

TEST_TRANS_RESULTS=1:1979
```

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，深入理解了 Cache 组成

结构对 C 程序性能的影响。

4.2 请给出对本次实验内容的建议

PPT 写的有点离谱，感觉把很简单的实验过程（只是指过程不是指难度）描述的十分复杂，建议简化。

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.