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　1190200523

班　　级　1903002

学　　生　石翔宇

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
- 1TB HDD + 512G SSD

1.2.2 软件环境

- Windows 10 21H1
- Ubuntu 20.04 LTS

1.2.3 开发工具

- VSCode，CodeBlocks，gcc+gdb

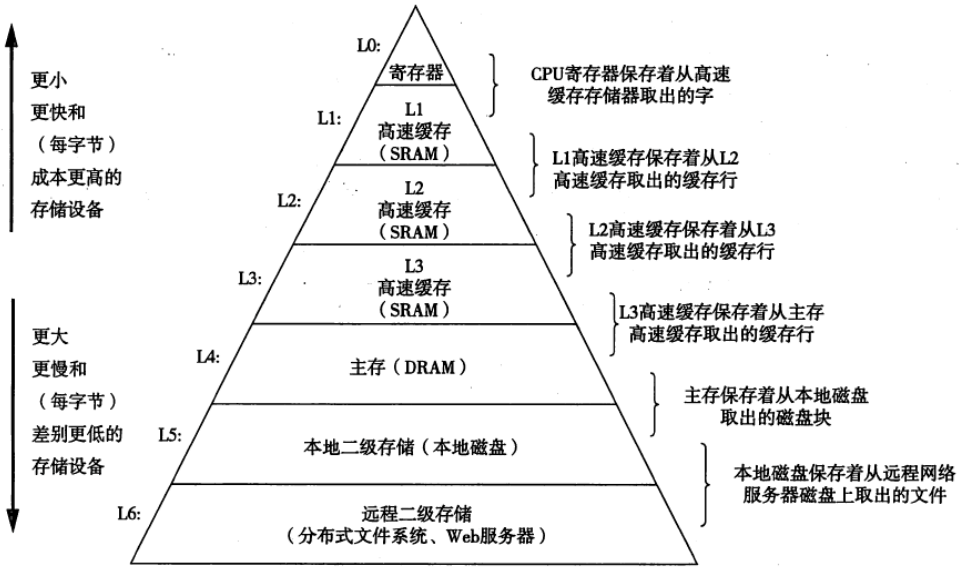
1.3 实验预习

- 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 计算机 Cache 的参数查看与分析（5 分）

Name	C(大小)	S(组数)	E(路数)	B(块大小)	s（组编码位数）	b（块内偏移地址位数）
L1 Data cache	32 KBytes	64	8-way set associative	64-byte line size	6	6
L1 Instruction cache	32 KBytes	64	8-way set associative	64-byte line size	6	6
L2 cache	256 KBytes	1024	4-way set associative	64-byte line size	10	6
L3 cache	12 MBytes,	2048	16-way set associative	64-byte line size	11	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

1. 读策略：
   a) 命中：将 cache 中的数据存储到 CPU 或者上一级 cache 中。
   b) 未命中：从下一级 cache 中读取数据，存储在当前层级的 cache 中。
2. 写策略：

a) 命中：

i. Write-through: 命中后更新缓存，同时写入到内存中。

ii. Write-through: 命中后更新缓存，同时写入到内存中。

b) 未命中：

i. Write-allocate: 载入到缓存中，并更新缓存。

ii. No-write-allocate: 直接写入到内存中，不载入到缓存。

2.4 写出用 gprof 进行性能分析的方法（5 分）

Profiling 可以使我们看到程序运行时程序的调用关系、函数的消耗时长等。这些信息可以使我们了解程序中那块代码耗时高于预期。

使用 Profiling 主要包括如下三步：

1. 编译链接程序时要添加-pg 选项使能 profiling。
2. 执行编译处的可执行文件，产生 profile 数据文件 gmon.out，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 分析 profile 数据。

根据产生的 profile ，可以产生各种不同实行的分析输出。如 The Flat Profile、The Call Graph、The Annotated Source Listing。

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，灵活轻巧而又强大。

在 linux 下，使用命令 valgrind --tool=callgrind execname 会在当前目录下生成文件，文件名字为“callgrind.out.进程号"

生成的文件有两种处理方式：

1. 生成 dot 文件

   gprof2dot.py -f callgrind -n10 -s callgrind.out.31113 > valgrind.dot

2. 将 dot 文件转换成图片

   dot -Tpng valgrind.dot -o valgrind.png

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. 维护全局变量 lru_counter， 每次操作完成后加一。
2. 操作时：
   a) 先找出组索引和标记。
   b) 扫描整个块的数组，若匹配到了数据（valid=1 且 tag 相同），则 hit，注意此时要更新当前行的 lru，将当前的 lru_counter 赋值给它即可，操作完成直接结束函数。
   c) 扫描整个块的数组，找到 lru 最小的行，若扫描完之后还没有进行 b 步的操作，则 miss。若 lru 最小的行的 valid=1，则 eviction。更新当前行的数据。

测试用例 1 的输出截图（5 分）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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）：

```
dedsec@ubuntu:~/CSAPP/Lab/Lab6/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
dedsec@ubuntu:~/CSAPP/Lab/Lab6/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：

```
dedsec@ubuntu:~/CSAPP/Lab/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
```

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

缓存一共有 32 组，每个组有一个 cache line，每个块的大小为 32 字节，也就是说一个 cache line 里面能装入 8 个 int。缓存总共能装进 256 个 int。

1. 32×32:

鉴于同一个矩阵中的缓存冲突问题，我们考虑分块，由于一个 cache line 中能装进 8 个 int，且缓存可装进 8 行的数据，则选择 8*8 的块。对每一块操作时，我们先依次访问当前行的 8 个 int，将这 8 个数据放入 8 个局部变量中，再将这 8 个放到转置后的矩阵当中。

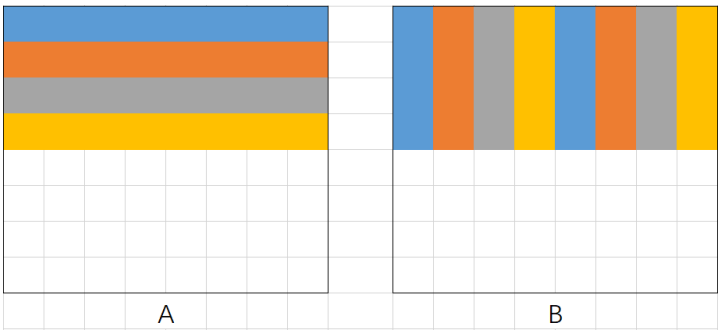
2. 64×64:

每一行的数据个数增加到了 64，这时若还使用上述的 8*8 分块，则块中第 0 行和第 4 行的数据会发生冲突，这是因为 64×64 的矩阵每个行需要 8 个缓存块，每四行缓存 index 会重复一次。

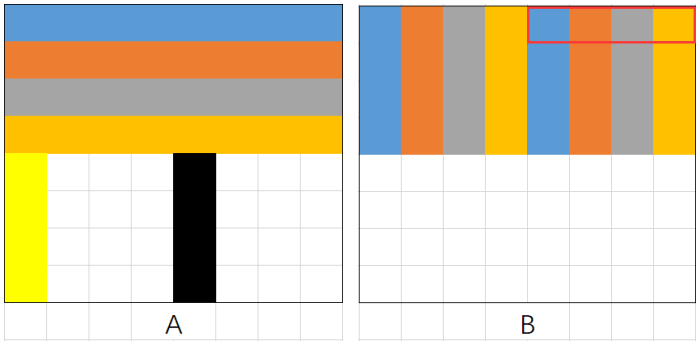
若使用 4*4 分块，则没有充分利用到每次加载到缓存中的数据。

考虑新的方法，将 8*8 和 4*4 分块结合起来。

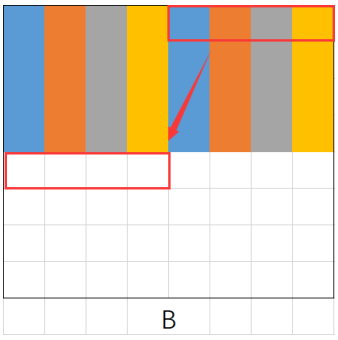
1. 8*8 分块。
2. 遍历 A 的前 4 行，将每行的前 4 个数字放在 B 中相应的位置。将每行的后 4 个数字放在前 4 个数字位置列数+4 的位置，如图所示。放置时同样使用上述的策略。



3. 遍历 k 从 0 到 3。k 表示列数。（**图示均为 k=0 时的情况**）
   a) 将 8*8 块中的左下角 4*4 的小块的第 k 列数据（图中黄色部分）复制到局部变量 tmp[0~3]中，将右下角 4*4 的小块的第 k 列数据（图中黑色部分）复制到局部变量 tmp[4~7]中。



b) 则 tmp[1~3]中的数据应该放到上图中红色框所示部分。将 tmp[1~3]分别与上图红色框中数据交换。



c) 将 tmp[1~3]中数据放到左下角 4*4 小块的第 k 行，如上图所示。

d) 将 tmp[4~7]中数据放到右下角 4*4 小块的第 k 行。

e) 重复上述步骤。

3. 61×67:

使用与 64×64 相同的方法。注意分块后会有块中元素个数不满的情况。

32×32（10 分）：运行结果截图

```
dedsec@ubuntu:~/CSAPP/Lab/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:289, evictions:257

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

TEST_TRANS_RESULTS=1:289
```

64×64（10 分）：运行结果截图

```
dedsec@ubuntu:~/CSAPP/Lab/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:9138, misses:1109, evictions:1077

Summary for official submission (func 0): correctness=1 misses=1109

TEST_TRANS_RESULTS=1:1109
```

61×67（20 分）：运行结果截图

```
dedsec@ubuntu:~/CSAPP/Lab/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:6182, misses:1999, evictions:1967

Summary for official submission (func 0): correctness=1 misses=1999

TEST_TRANS_RESULTS=1:1999
```

第 4 章 总结

4.1 请总结本次实验的收获

1. 更加深入的了解了缓存的相关知识和缓冲命中的原理；
2. 学会了通过优化代码实现增加缓存命中率，加快程序运行速度的方法。

4.2 请给出对本次实验内容的建议

1. 希望实验指导书能够更加详细，更加有条理。

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.