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 1190201423

班　　级 1903004

学　　生 顾海耀

指 导 教 师 史先俊

实 验 地 点 G709

实 验 日 期 2021/5/26

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

X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位;

1.2.3 开发工具

Visual Studio 2010 64 位以上；TestStudio；Gprof;Valgrind 等

1.3 实验预习

填上实验课前，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 C、S、E、B、s、e、b

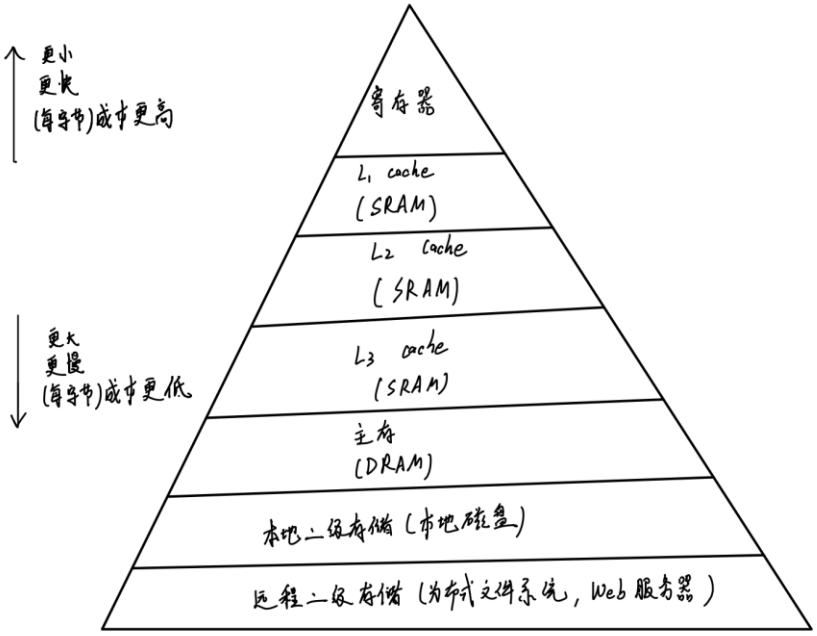
写出 Cache 的基本结构与参数

写出各类 Cache 的读策略与写策略

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）



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 数据 cache	32KB	64	8	64	6	3	6
L1 指令缓存	32KB	64	8	64	6	3	6
L2 cache	256KB	1024	4	64	10	2	6
L3 cache	12MB	12288	16	64	$Log_2 12288$	4	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

Cache 读策略：

- 缓存命中，读取相应的数据加载到 CPU 或上一级 cache 中
- 缓存不命中，从贮存或者下一级 cache 中读取数据，并替换数据

Cache 写策略：

- 直写，立即将高速缓存块写回到紧挨着的低一级 cache 中
- 写回，尽可能推迟更新，只有当替换算法要驱逐这个更新过的块时，才把它写到紧挨着的低一层中
- 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

• 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.

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= [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]

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：

本设计主要是实现一个 cache 模拟器，整体的框架已经提供了，只需要完成 initCache、freeCache、accessData 这 3 个函数即可。

initCache 设计思路：本函数主要实现内存分配，可以看到 cache_set_t 是每个块的结构，而 cache_line_t 是一行的结构，于是因为 S,E 是已知的，就只要定义成二位数组即可，需要注意就是要给 cache_line_t 这个结构体的每个元素赋初值。

```
void initCache()
{
        cache = malloc(S * sizeof(cache_set_t));
        for (int i = 0; i < S; i++)
        {
                cache[i] = malloc(E * sizeof(cache_line_t));
                for (int j = 0; j < E; j++)
                {
                        cache[i][j].valid = '0';
                        cache[i][j].tag = 0;
                        cache[i][j].lru = 0;
                }
        }
}
```

freeCache 设计思路：先释放每个块的内存，在释放 cache 的即可

```
void freeCache()
{
        for (int i = 0; i < S; i++)
        {
                free(cache[i]);
        }
        free(cache);
}
```

accessData 设计思路：因为已经得知了 m 位的地址，因为不需要取出这个值，所以只需要知道 s，t 即可，b 不需要知道，可以设置一个标志，若中了，则 hit_count++、lru_counter++即可，若遍历 E 都发现没有，则说明可能出现其他两种情况，若不命中，miss_count++、lru_counter++即可，若出现驱逐，只有可能

是遍历每一组，若出现全满的情况则需要驱逐，eviction_count++即可。

```
void accessData(mem_addr_t addr)
{
        int flag=0;
        int lru_flag=0;
        unsigned long long int m_m = addr;
        int lru_zero;
        unsigned long long int ai;
        unsigned long long int m_s;
        unsigned long long int m_t;
        m_s = m_m << (64 - b - s);
        m_s = m_s >> (64 - s);
        m_t = m_m >> (s + b);
        for (int i = 0; i < E; i++)
        {
                if ((cache[m_s][i].valid == '1') && (m_t == cache[m_s][i].tag))
                {
                        lru_counter++;
                        hit_count++;
                        cache[m_s][i].lru = lru_counter;
                        flag = 1;
                        break;
                }
                if (cache[m_s][i].valid != '1')
                        lru_flag = 1;
        }
        if (flag == 0)
        {
                miss_count++;
                if (lru_flag == 1)
                {
                        for (int i = 0; i < E; i++)
                        {
                                if (cache[m_s][i].valid != '1')
                                {
                                        lru_counter++;
                                        cache[m_s][i].valid = '1';
                                        cache[m_s][i].lru = lru_counter;
                                        cache[m_s][i].tag = m_t;
                                        break;
                                }
                        }
                }
                else
                {
                        eviction_count++;
                        lru_zero = cache[m_s][0].lru;
                        for (int i = 1; i < E; i++)
                        {
                                if (cache[m_s][i].lru < lru_zero)
                                {
                                        lru_zero = cache[m_s][i].lru;
                                        ai = i;
                                }
                        }
                        lru_counter++;
                        cache[m_s][ai].valid = '1';
                        cache[m_s][ai].lru = lru_counter;
                        cache[m_s][ai].tag = m_t;
                }
        }
}
```

测试用例 1 的输出截图（5 分）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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）：

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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

```
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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 ghy1190201423-handin.tar  csim.c trans.c
csim.c
trans.c
ghy1190201423@ubuntu:/mnt/hgfs/hitics/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：

32*32：cache 的 S = 5,E = 1,B = 5，所以是一个直接映射高速缓存，有 32 个组，每个组只有一行，每行能存 32 字节，相当于就是 8 个 int 类型的数据，所以我们可以定义 8 个变量，t1~t8，相当于把数据分成 8*8 的几个块。

```
int i,j,t1,t2,t3,t4,t5,t6,t7,t8;
if(M==32&&N==32)
{
        for(i=0;i<M;i=i+8)
        {
            for(j=0;j<N;j++)
            {
                t1 = A[j][i];
                t2 = A[j][i+1];
                t3 = A[j][i+2];
                t4 = A[j][i+3];
                t5 = A[j][i+4];
                t6 = A[j][i+5];
                t7 = A[j][i+6];
                t8 = A[j][i+7];
                B[i][j] = t1;
                B[i+1][j] = t2;
                B[i+2][j] = t3;
                B[i+3][j] = t4;
                B[i+4][j] = t5;
                B[i+5][j] = t6;
                B[i+6][j] = t7;
                B[i+7][j] = t8;
            }
        }
}
```

64*64：如果还是使用上面的方法并不是很好，因为对 A 数组的访问依然是第一个不命中；对 B 数组的访问，前 4 行和后四行所映射的块是相同的，于是访问完前四行的第一列后，访问后四行的第一列会冲突不命中，导致原来的块被驱逐，再访问前四行的第二列，由于之前的块已经被驱逐，因此又会 miss 且驱逐，如此反复下去，B 数组中所有的元素皆会不命中。而每一行元素会占 8 个组，因此 4 行元素即可占满 cache。于是可以选用 8*8 分组和 4*4 结合的方法：

```
if(M==64&&N==64)
{
        for(i = 0;i < N;i = i + 8)
        {
                for(j = 0;j < M;j = j + 8)
                {
                        for(k = 1;k < i + 4;k++)
                        {
                                t1 = A[k][j];
                                t2 = A[k][j+1];
                                t3 = A[k][j+2];
                                t4 = A[k][j+3];
                                t5 = A[k][j+4];
                                t6 = A[k][j+5];
                                t7 = A[k][j+6];
                                t8 = A[k][j+7];
                                B[j][k] = t1;
                                B[j+1][k] = t2;
                                B[j+2][k] = t3;
                                B[j+3][k] = t4;
                                B[j+4][k] = t5;
                                B[j+5][k] = t6;
                                B[j+6][k] = t7;
                                B[j+7][k] = t8;
                        }
                        for(p = j;p < j + 4;p++)
                        {
                                t1=A[i+4][p];
                                t2=A[i+5][p];
                                t3=A[i+6][p];
                                t4=A[i+7][p];
                                t5=B[p][i+4];
                                t6=B[p][i+5];
                                t7=B[p][i+6];
                                t8=B[p][i+7];
                                B[p][i+4]=t1;
                                B[p][i+5]=t2;
```

```
                                B[p][i+6]=t3;
                                B[p][i+7]=t4;
                                B[p+4][i]=t5;
                                B[p+4][i+1]=t6;
                                B[p+4][i+2]=t7;
                                B[p+4][i+3]=t8;
                        }
                        for(k=i+4;k<i+8;k++)
                        {
                                t1=A[k][j+4];
                                t2=A[k][j+5];
                                t3=A[k][j+6];
                                t4=A[k][j+7];
                                B[j+4][k]=t1;
                                B[j+5][k]=t2;
                                B[j+6][k]=t3;
                                B[j+7][k]=t4;
                        }
                }

        }

    }
```

61*67：选用分块策略，不断进行测试，最终发现选用 18 时效果最佳

```
if(M==61&&N==67)
{
        for(i=0;i<N;i=i+17)
        {
                for(j=0;j<M;j=j+17)
                {
                        for(k=i;k<i+17 && k<N;k++)
                        {
                                for(p=j;p<j+17 && p<M;p++)
                                        {
                                                t1 = A[k][p];
                                                B[p][k] = t1;
                                        }
                        }
                }
        }
}
```

32×32（10 分）：运行结果截图

```
ghy1190201423@ubuntu:~/Documents/cachelab-handout$ sudo ./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

```
ghy1190201423@ubuntu:~/Documents/cachelab-handout$ sudo ./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:9066, misses:1179, evictions:1147

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=1179

TEST_TRANS_RESULTS=1:1179
```

61×67（20 分）：运行结果截图

```
ghy1190201423@ubuntu:~/Documents/cachelab-handout$ sudo ./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:6218, misses:1961, evictions:1929

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=1961

TEST_TRANS_RESULTS=1:1961
```

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。

理解了现代计算机系统存储器层级结构

掌握了 Cache 的功能结构与访问控制策略

培养了 Linux 下的性能测试方法与技巧

4.2 请给出对本次实验内容的建议

无

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.