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 1170301027

班　　级 1703010

学　　生 冯帅

指 导 教 师 史先俊

实 验 地 点 **G712**

实 验 日 期 20181202

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 Core i7-7700HQ 2.81GHz，8GB RAM，128GB SSD

1.2.2 软件环境

Windows10 64 位以上； Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

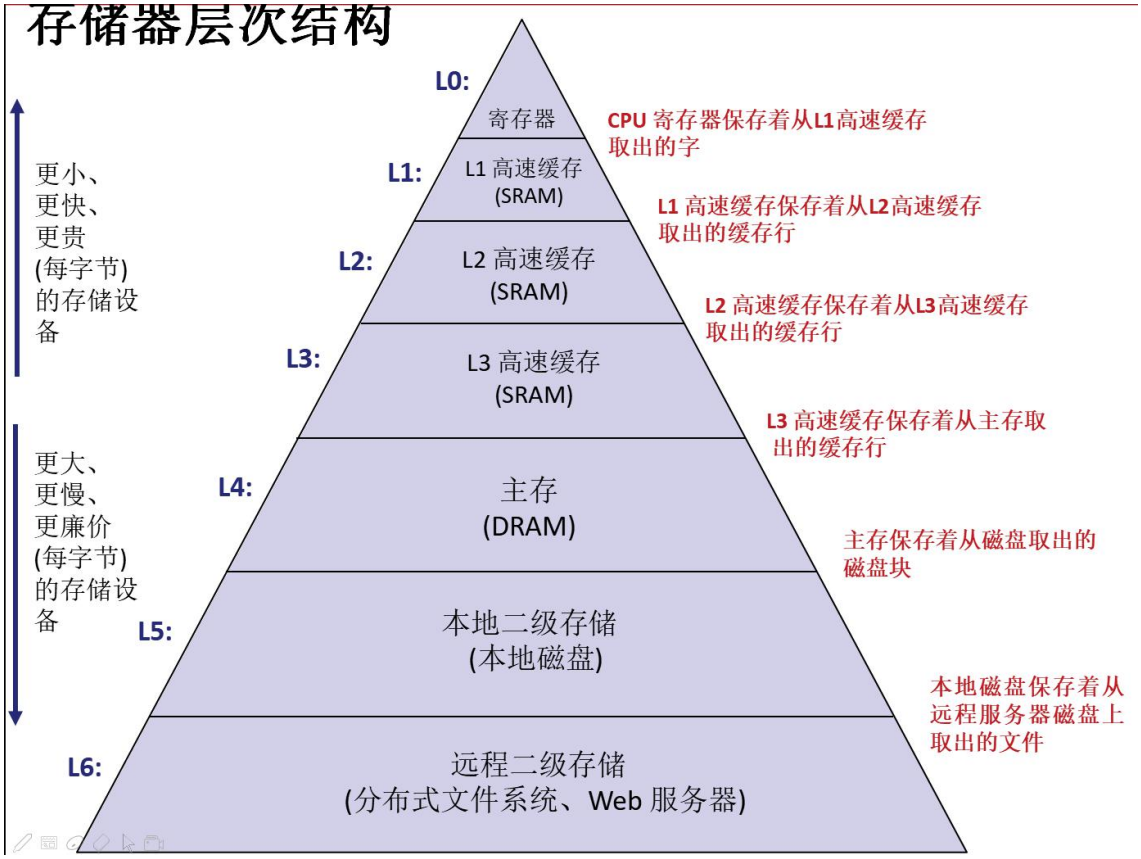
Visual Studio 2010 64 位以上；TestStudio；Gprof;Valgrind 等

1.3 实验预习

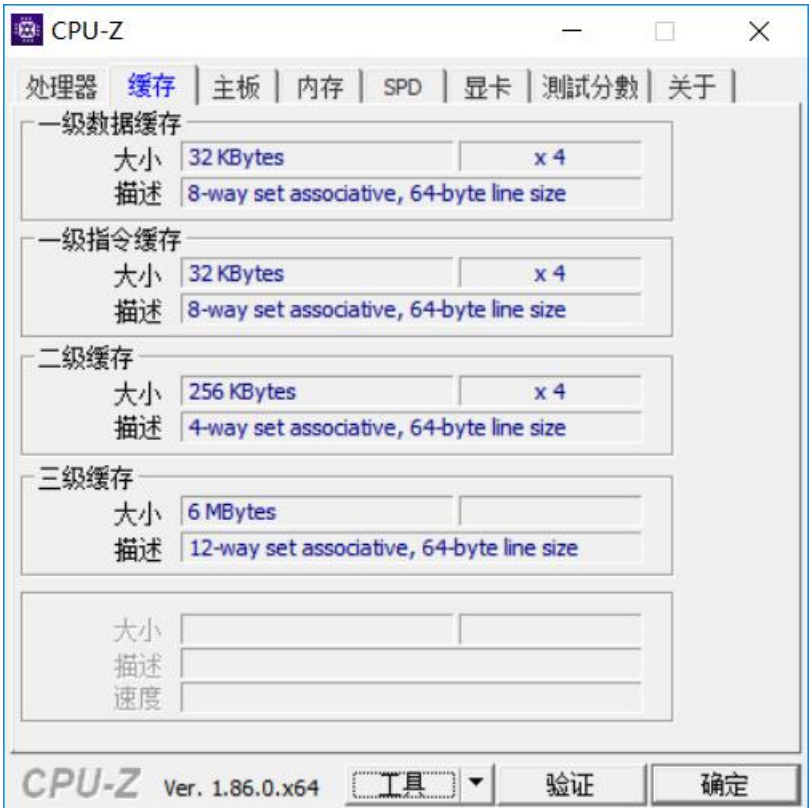
填写

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）



参数	C	S	E	B	s	e	b
一级数据缓存	32KB	64	8	64B	6	3	6
一级指令缓存	32KB	64	8	64B	6	3	6
二级缓存	256KB	1024	4	64B	10	2	6
三级缓存	6MB	8192	12	64B	13	3.58	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

读策略：

定位组

检查集合中的任何行是否有匹配的标记

是 + 行有效: 命中

定位从偏移开始的数据

写策略：

（1）写命中时，两种

直写 (立即写入存储器)，将 V=0

写回 (推迟写入内存直到行要替换)

需要一个修改位 (和内存相同或不同的行)

（2）写不命中时，两种

写分配 (加载到缓存,更新这个缓存行)

好处是更多的写遵循局部性

非写分配 (直接写到主存中,不加载到缓存中)

2.4 写出用 gprof 进行性能分析的方法（5 分）

gprof 用于分析函数调用耗时，可用之抓出最耗时的函数，以便优化程序。

gcc 链接时也一定要加-pg 参数，以使程序运行结束后生成 gmon.out 文件，供 gprof 分析。

gprof 默认不支持多线程程序，默认不支持共享库程序。

1. gcc -pg 编译程序
2. 运行程序，程序退出时生成 gmon.out
3. gprof ./prog gmon.out -b 查看输出

2.5 写出用 Valgrind 进行性能分析的方法（5 分）

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：

accessdata 的实现思想：

```
void accessData(mem_addr_t addr)
{
    set_index_mask = (addr>>(s+b));//标记位
    int flag = 0;//标记是否命中
    unsigned long long int oldlru,newlru;
    oldlru = 10000;//存储最久的
    newlru = 0;//存储最新的
    int i = ((addr>>b)&((1<<s)-1));//获取组索引
    int oldi;
    for(int j=0;j<E;j++)
    {
        if(cache[i][j].tag == set_index_mask && cache[i][j].valid)
        {
            if(verbosity) printf("hit\n");
            flag = 1;
            hit_count++;
            cache[i][j].lru = lru_counter++;
        }
    }
```

```
    if(!flag)//未命中
    {
        if (verbosity)    printf("miss");
        miss_count++;
        for(int j=0;j<E;j++)
        {
            if(cache[i][j].lru<oldlru)
            {
                oldlru = cache[i][j].lru;
                oldi = j;
            }
            if(cache[i][j].lru>newlru)
            {
                newlru = cache[i][j].lru;
            }
        }
        cache[i][oldi].lru = newlru+1;
        cache[i][oldi].tag = set_index_mask;//插入
        if(cache[i][oldi].valid)//原有值，驱逐
        {
            if(verbosity) printf(" and eviction\n");
            eviction_count++;
        }
        else//原无值插入
        {
            if(verbosity) printf("\n");
            cache[i][oldi].valid = 1;
        }
    }
}
```

用 i 代表组索引，set_index_mask 代表标记位（这里因为地址没有 f 开头的所以默认算数右移不作处理）去比较，判断命中则 hit++，修改命中标志位（flag），进而继续判断当不命中时，miss++，应用 lru 算法，将最修改时间最早也就是 lru 在赋初值为零的情况下最小的那个块记录下来，用于当前 miss 的缓存，若原来没有值那么其有效位为零，此间不做处理；但是若有效值为 1，那么说明原来有值，那么此间造成一次驱逐，eviction++；

测试用例 1 的输出截图（5 分）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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）：

```
fs1170301027@ubuntu:~/桌面/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

```
fs1170301027@ubuntu:~/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：

对于 32x32 的矩阵转置，我觉得还是简单粗暴一点好，分成 8x8 的块做，用局部变量直接将载入的 A 存储下来，然后赋值给 B，直接避免由于对角线上二次载入造成的冲突，减少 miss。

对于 64x64 的矩阵转置我没算明白，最后我是将 8x8 的 blocking 做成了 4x4 的形式存的，但是这样会造成 1/2 的损失，所以最终 miss 一千六百多，放弃了，

```
fs1170301027@ubuntu:~/桌面/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:6546, misses:1651, evictions:1619

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=1651

TEST_TRANS_RESULTS=1:1651
```

对于 61x67 的矩阵转置，因为 8x8 能充分利用一个块的空间，所以我尝试了一下，发现并不理想，miss 两千四百多，后来改成 16x16 直接就 1992 了，很蒙，因为 61x67 的矩阵并不符合一定特殊的分块思想（不是整数行之间的冲突），即当前对角线上元素并不是所有都起冲突，而且这时候算冲突很难在代码上体现出来，所以我直接就加载进 b 里面，未对冲突做处理，没想到直接实现了。

```
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    for(int i=0;i<N;i += 16)
    {
        for(int j=0;j<M;j += 16)
        {
            for(int l=i;l<i+16 && l<N;l++)
                for(int k=j;k<j+16 && k<M;k++)
            {
                        B[k][l] = A[l][k];
            }
        }
    }

}
```

32×32（10 分）：运行结果截图

```
fs1170301027@ubuntu:~/桌面/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: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

```
fs1170301027@ubuntu:~/桌面/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:6546, misses:1651, evictions:1619

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=1651

TEST_TRANS_RESULTS=1:1651
```

61×67（20 分）：运行结果截图

```
fs1170301027@ubuntu:~/桌面/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:6187, misses:1992, evictions:1960

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:6187, misses:1992, evictions:1960

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

TEST_TRANS_RESULTS=1:1992
```

第 4 章

总结

4.1 请总结本次实验的收获

```
fs1170301027@ubuntu:~/桌面/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           4.0         8        1651
Trans perf 61x67          10.0        10        1992
          Total points    49.0        53
```

熟悉了 cache 的操作

4.2 请给出对本次实验内容的建议

没什么更好的建议

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.