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 1170300821

班　　级 1703008

学　　生 罗瑞欣

指 导 教 师 郑贵滨

实 验 地 点 G712

实 验 日 期

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

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

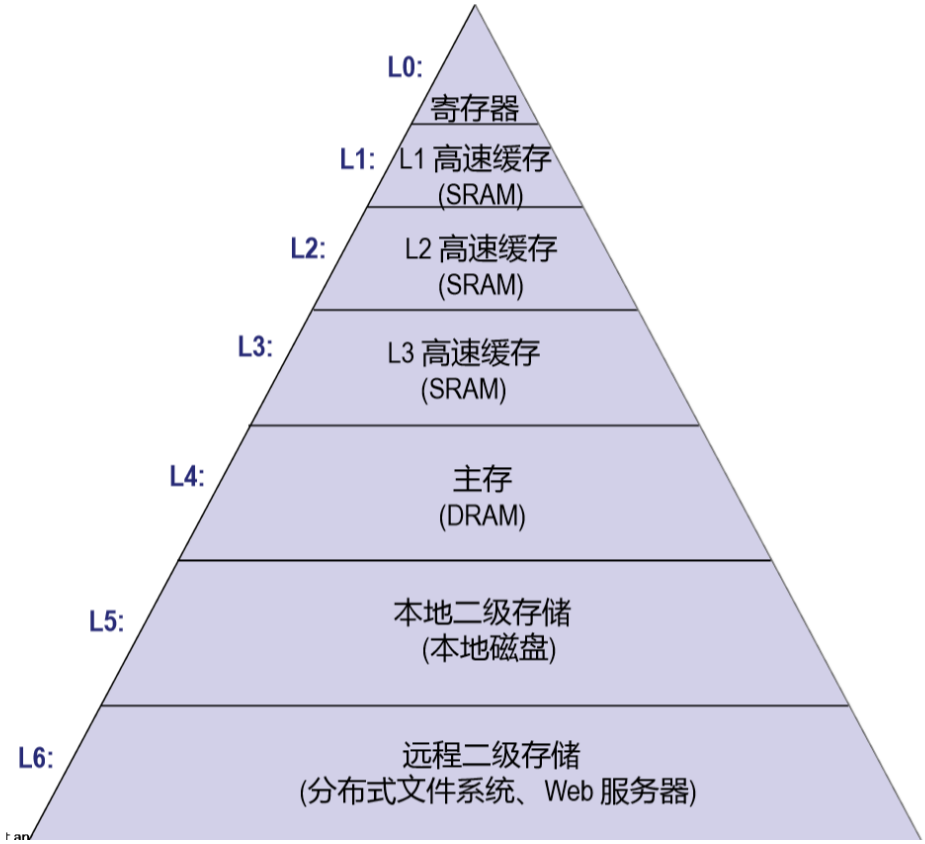
用 CPUZ 等查看你的计算机 Cache 各参数，写出 Cache 的基本结构与参数：C S E B s e b

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）



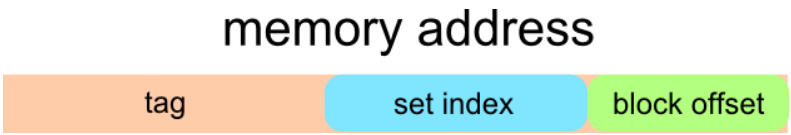
从上到下，价格越来越便宜，访问速度越来越慢，容量越来越大

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）

Intel Core i7-6700HQ Skylake							
	C	S	E	B	s	e	b
L1	32KB*4*2	64	8	64	6	3	6
L2	256KB*4	262144	4	64	18	2	6
L3	6MB	8192	12	64	13	ln12/ln2	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

在中读取一条地址的内容时，将该地址按照如下方式进行分解



其中 set index 表明该地址所需要的组号，block offset 表示所需数据在该行的偏移位，tag 用来确定读取该组的哪一行，只有设置了有效位和 tag 位都能对应上才能命中该数据。如果发生不命中，则采用 LRU 策略进行驱逐和调换。

写数据时，采用直写和写回两种方式，直写就是将数据直接写入对应的 cache 块的低一层中，写会是要等到该块呗驱逐和替换时才写入数据到紧接着的低一层中。发生不命中时，采用两种方案，一种是写分配，一种是非写分配，写分配是要加载低一层的块到 cache 中，然后更新这个 cache 块，非写分配是要将数据直接写到低一层的数据块中。

2.4 写出用 gprof 进行性能分析的方法（5 分）

编译程序

使用 gcc/cc 编译和链接时需要加入-pg 选项
使用 ld 链接时需要用/lib/gcrt0.o 代替 crt0.o 作为第一个 input 文件
如果要调试 libc 库需要使用-lc_p 代替-lc 参数

运行程序生成统计信息

正常运行编译好的程序，程序正常结束后会在当前目录生成统计信息文件 gmon.out。

程序必须正常退出（调用 exit 或从 main 中返回）才能生成统计信息。

当前目录下如果有另外叫 gmon.out 的文件，内容将被本次运行生成的统计信息覆盖，多次运行统一程序请将前一次的 gmon.out 改名。

使用 gprof 查看统计结果

命令格式：

gprof options [executable-file [profile-data-files...]] [> outfile]

常用参数介绍：

symspec 表示需要加入或排除的函数名，和 gdb 指定断点时的格式相同。

1） 输出相关：

a) -A[symspec]或--annotated-source[=symspec]：进行源码关联，只关联 symspec 指定的函数，不指定为全部关联。

b) -I dirs 或--directory-path=dirs：添加搜索源码的文件夹，修改环境变量 GPROF_PATH 也可以。

c) -p[symspec]或--flat-profile[=symspec]：默认选项，输出统计信息，只统计 symspec 指定的函数，不指定为全部统计。

d) -P[symspec]或--no-flat-profile[=symspec]：排除统计 symspec 指定的函数

e) -q[symspec]或--graph[=symspec]：默认选项，输出函数调用信息，只统计 symspec 指定的函数，不指定为全部统计。

f) -Q[symspec]或--no-graph[=symspec]：排除统计 symspec 指定的函数

g) -b 或--brief：不输出对各个参数含义的解释；

2） 分析相关：

a) -a 或--no-static：定义为 static 的函数将不显示，函数的被调用次数将被计算在调用它的不是 static 的函数中；

b) -m num 或--min-count=num：不显示被调用次数小于 num 的函数；

c) -z 或--display-unused-functions：显示没有被调用的函数；

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

1. -tool=<name> 最常用的选项。运行 valgrind 中名为 toolname 的工具。默认 memcheck。

2. h –help 显示帮助信息。

3. -version 显示 valgrind 内核的版本，每个工具都有各自的版本。
4. q –quiet 安静地运行，只打印错误信息。
5. v –verbose 更详细的信息, 增加错误数统计。
6. -trace-children=no|yes 跟踪子线程? [no]
7. -track-fds=no|yes 跟踪打开的文件描述？[no]
8. -time-stamp=no|yes 增加时间戳到 LOG 信息? [no]
9. -log-fd=<number> 输出 LOG 到描述符文件 [2=stderr]
10. -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
11. -log-file-exactly=<file> 输出 LOG 信息到 file
12. -log-file-qualifier=<VAR> 取得环境变量的值来做为输出信息的文件名。[none]
13. -log-socket=ipaddr:port 输出 LOG 到 socket ，ipaddr:port

LOG 信息输出

1. -xml=yes 将信息以 xml 格式输出，只有 memcheck 可用
2. -num-callers=<number> show <number> callers in stack traces [12]
3. -error-limit=no|yes 如果太多错误，则停止显示新错误? [yes]
4. -error-exitcode=<number> 如果发现错误则返回错误代码 [0=disable]
5. -db-attach=no|yes 当出现错误，valgrind 会自动启动调试器 gdb。[no]
6. -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：

1. -leak-check=no|summary|full 要求对 leak 给出详细信息? [summary]
2. -leak-resolution=low|med|high how much bt merging in leak check [low]
3. -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：

（1）

A. 模拟器必须适应不同的 s, E, b，所以数据结构必须动态申请。。

B. 实验假设内存全部对齐，即数据不会跨越 block，所以测试数据里面的数据大小也可以忽略。

（2）根据给出的代码框架，明确自己要写的几个函数：void initCache() ，void freeCache() ， void accessData(mem_addr_t addr)

（3）将 cache 当做是一个特殊的二维数组，每个元素需要利用 tag 和组号还有 tag 进行确定。

（4）initcache 函数用来将 cache 初始化完成，于是利用 mallloc 函数为 cache 分配内存空间，并将 cache 中的各个元素初始化为零。

（5）accessdata（）函数，利用掩码和移位操作得到没次内存地址的 tag 值和组索引的信息，接着在这个组中进行一次遍历，找到符合 tag 值和标记位的行中的数据，即为命中，hit_count+1，同时将自己设置的 flag 标志位设置为 1 用以区分，并且将该 cache 行中的 lru 标记量+1 方便后续的驱逐替换，并且每次查找的时候都进行一次判断 cache 中是否有空余行，即是否需要替换，如果有空余行，就改变自己设置的 flag2 标记变量。

（6）如果发生不命中，则将 miss_count+1，如果此时还有空余行，则一次查找操作结束，如没有空余行则将发生驱逐替换，利用 lru 策略，在 cache 当前组中中搜索 lru 最大和最小的两行，进行调整，将最小的一行替换为最大的一行，并将驱逐替换计数变量+1。即可完成 partA。

测试用例 1 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 2 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 3 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 4 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 5 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 6 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 7 的输出截图（5 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

测试用例 8 的输出截图（10 分）：

```
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
1170300821@luoruixin:~/hitics/lab6/cachelab-handout$
```

```
1170300821@luoruixin:~/hitics/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
```

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

3.2 矩阵转置设计

提交 trans.c

程序设计思想：

（1）

A. miss 的最低限度是 1/8。cache 的大小为 32*32，32 个 block，128 个 int。

B. 实验依赖 blocking，以数据块的形式读取数据，完全利用后丢弃，然后读取下一个，防止 block 利用的不全面。

C. 尽量使用刚刚使用的 block，hit 的概率会很大。

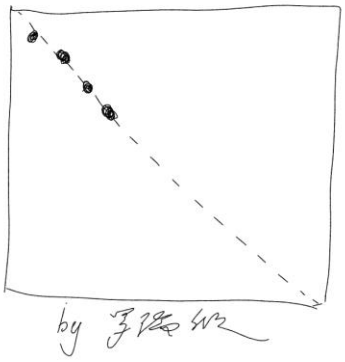
D. 读出和写入的时候注意判断这两个位置映射在 cache 中的位置是否相同。

根据三个测试条件分为三种不同的情况进行编写

（2）M = 32, N = 32

A. 此情况，一行是 32 个 int（4 个 block），cache 可以存 8 行，由此可以推出映射冲突的情况：只要两个 int 之间相差 8 行的整数倍，那么读取这两个元素所在的 block 就会发生替换，再读后面连续的元素也会不断发生替换

B. 但。。。。A[i][j]中 i = j，那么 B 也会是 B[i][j]，即映射遇到同一个 block 中，而当 i = j 的时候，就是对角线



C. 处理方法：由于可以使用 12 个局部变量，所以我们可以用 8 个局部变量一次性将包含对角线 int 的 block 全部读出，这样即使写入的时候替换了之前的 block 也不要紧。

D. 具体实现

```
tmp0 = A[i][j];
tmp1 = A[i][j+1];
tmp2 = A[i][j+2];
tmp3 = A[i][j+3];
tmp4 = A[i][j+4];
tmp5 = A[i][j+5];
tmp6 = A[i][j+6];
tmp7 = A[i][j+7];
```

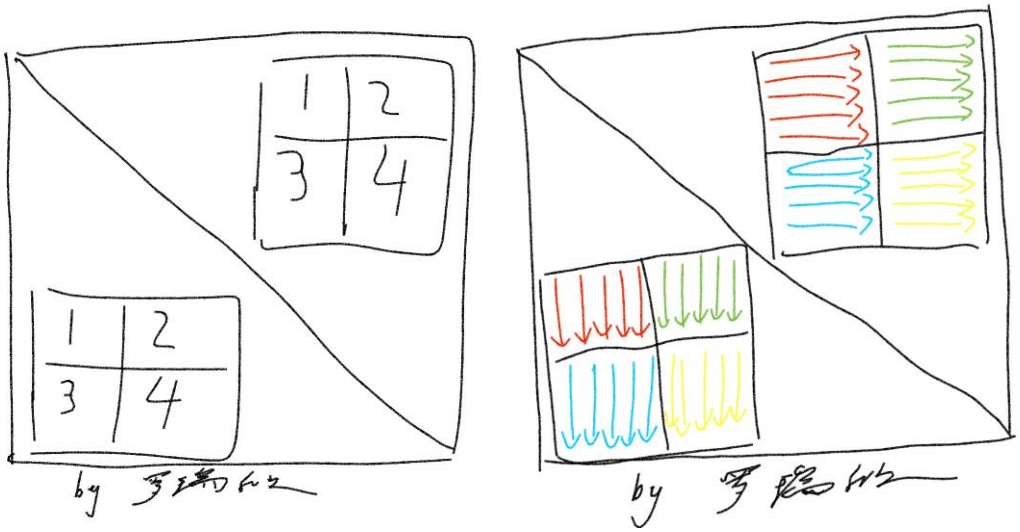
```
B[j][i] = tmp0;
B[j+1][i] =tmp1;
B[j+2][i] = tmp2;
B[j+3][i] = tmp3;
B[j+4][i] = tmp4;
B[j+5][i] = tmp5;
B[j+6][i] = tmp6;
B[j+7][i] = tmp7;
```

（2）M = 64, N = 64

A. 此时，数组一行有 64 个 int（8 个 block），每四行就会填满一个 cache，即两个元素相差四行就会发生冲突。

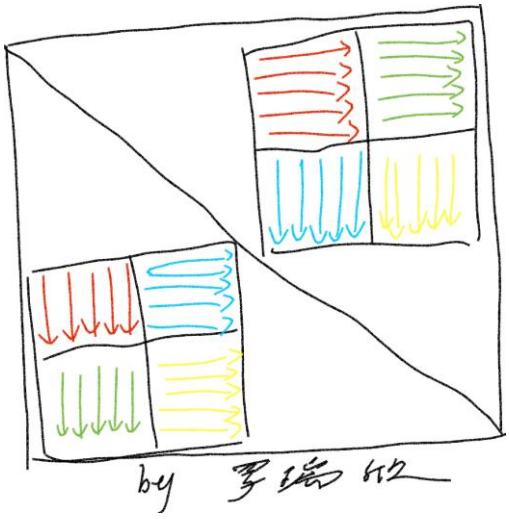
B. 如果使用 4*4 的 blocking，每次都会有 1/2 的损失，优化不够。如果使用 8*8 的 blocking，那么在写入的时候就会发生冲突

C. 将右上角的 2 移动到左下角的 3 的，但是为了防止冲突先把他们移动到 2 的位置，对于 3 和 4，采取一样的策略。最后再将 23 互换



D. （又）但。。。。。测试以后并不能满足优化的要求

应该在将右上角的 34 转换的过程中将 2 的位置复原。尽量使用刚刚使用的 block，因为它们很可能还没有被替换，hit 的概率会很大。在转换 2 的时候逆序转换，同时在读取右上角 34 的时候按列来读，这样的好处就是把 2 换到 3 的过程中是从下到上按行换的，因为这样可以先使用最近使用的 block



E. 具体实现

```
for (i = k; i < k + 4; i++)
{

    tmp0 = A[i][j];
    tmp1 = A[i][j+1];
    tmp2 = A[i][j+2];
    tmp3 = A[i][j+3];
    tmp4 = A[i][j+4];
    tmp5 = A[i][j+5];
    tmp6 = A[i][j+6];
    tmp7 = A[i][j+7];

    B[j][i] = tmp0;
    B[j+1][i] = tmp1;
    B[j+2][i] = tmp2;
    B[j+3][i] = tmp3;

    B[j][i+4] = tmp4;
    B[j+1][i+4] = tmp5;
    B[j+2][i+4] = tmp6;
    B[j+3][i+4] = tmp7;
}
```

```
for (i = j; i < j + 4; i++)
{
    tmp0 = B[i][k+4];
    tmp1 = B[i][k+5];
    tmp2 = B[i][k+6];
    tmp3 = B[i][k+7];
    tmp4 = A[k+4][i];
    tmp5 = A[k+5][i];
    tmp6 = A[k+6][i];
    tmp7 = A[k+7][i];
    B[i][k+4] = tmp4;
    B[i][k+5] = tmp5;
    B[i][k+6] = tmp6;
    B[i][k+7] = tmp7;
    B[i+4][k] = tmp0;
    B[i+4][k+1] = tmp1;
    B[i+4][k+2] = tmp2;
    B[i+4][k+3] = tmp3;
}
```

```
for (i = j + 4; i < j + 8; i ++)
{

    tmp0 = A[k+4][i];
    tmp1 = A[k+5][i];
    tmp2 = A[k+6][i];
    tmp3 = A[k+7][i];
    B[i][k+4] = tmp0;
    B[i][k+5] = tmp1;
    B[i][k+6] = tmp2;
    B[i][k+7] = tmp3;
}
```

（3）M = 61, N = 67

A. 和 64*64 的情况大致相同，只是在最后一小块矩阵进行转置时，需要利用 tmp 变量进行单独处理即可。

B. 具体实现

```
for (i = k; i < k + 4; i++) {
    tmp0 = A[i][j];
    tmp1 = A[i][j + 1];
    tmp2 = A[i][j + 2];
    tmp3 = A[i][j + 3];
    tmp4 = A[i][j + 4];
    tmp5 = A[i][j + 5];
    B[j][i] = tmp0;
    B[j + 1][i] = tmp1;
    B[j + 2][i] = tmp2;
    B[j][i + 4] = tmp3;
    B[j + 1][i + 4] = tmp4;
    B[j + 2][i + 4] = tmp5;
    if (j == 54) {
        tmp0 = A[i][60];
        B[60][i] = tmp0;
    }
}
```

```
for (i = j; i < j + 3; i++) {
    tmp0 = B[i][k + 4];
    tmp1 = B[i][k + 5];
    tmp2 = B[i][k + 6];
    tmp7 = B[i][k + 7];
    tmp3 = A[k + 4][i];
    tmp4 = A[k + 5][i];
    tmp5 = A[k + 6][i];
    tmp6 = A[k + 7][i];
    B[i][k + 4] = tmp3;
    B[i][k + 5] = tmp4;
    B[i][k + 6] = tmp5;
    B[i][k + 7] = tmp6;
    B[i + 3][k] = tmp0;
    B[i + 3][k + 1] = tmp1;
    B[i + 3][k + 2] = tmp2;
    B[i + 3][k + 3] = tmp7;
}
```

```
for (i = j + 3; i < j + 6; i++) {
    tmp0 = A[k + 4][i];
    tmp1 = A[k + 5][i];
    tmp2 = A[k + 6][i];
    tmp3 = A[k + 7][i];
    B[i][k + 4] = tmp0;
    B[i][k + 5] = tmp1;
    B[i][k + 6] = tmp2;
    B[i][k + 7] = tmp3;
    if (i == 59) {
        tmp0 = A[k + 4][i + 1];
        tmp1 = A[k + 5][i + 1];
        tmp2 = A[k + 6][i + 1];
        tmp3 = A[k + 7][i + 1];
        B[i + 1][k + 4] = tmp0;
        B[i + 1][k + 5] = tmp1;
        B[i + 1][k + 6] = tmp2;
        B[i + 1][k + 7] = tmp3;
    }
}
```

小矩阵块用 temp 处理：

```
if (k == 56) {
    for (i = j; i < j + 6; i++) {
        tmp0 = A[64][i];
        tmp1 = A[65][i];
        tmp2 = A[66][i];
        B[i][64] = tmp0;
        B[i][65] = tmp1;
        B[i][66] = tmp2;
        if (i == 59) {
            tmp0 = A[64][60];
            tmp1 = A[65][60];
            tmp2 = A[66][60];
            B[60][64] = tmp0;
            B[60][65] = tmp1;
            B[60][66] = tmp2;
        }
    }
}
```

32×32（10 分）：运行结果截图

```
1170300821@luoruixin:~/C_work/cachelab-handout/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

```
1170300821@luoruixin:~/C_work/cachelab-handout/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:9026, misses:1219, evictions:1187

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=1219

TEST_TRANS_RESULTS=1:1219
```

61×67（20 分）：运行结果截图

```
1170300821@luoruixin:~/C_work/cachelab-handout/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:8113, misses:1986, evictions:1954

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=1986

TEST_TRANS_RESULTS=1:1986
```

第 4 章 总结

4.1 请总结本次实验的收获

（1）理解现代计算机系统存储器层级结构
（2）掌握 Cache 的功能结构与访问控制策略
（3）培养 Linux 下的性能测试方法与技巧
（4）深入理解 Cache 组成结构对 C 程序性能的影响
（5）充分理解了 cache 的工作原理
（6）学会了应该如何编写对 cache 友好的代码

4.2 请给出对本次实验内容的建议

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.