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　计算学部

学　　号　1190200208

班　　级　1936602

学　　生　李旻翀

指 导 教 师　刘宏伟

实 验 地 点　G709

实 验 日 期　2021.5.27

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

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

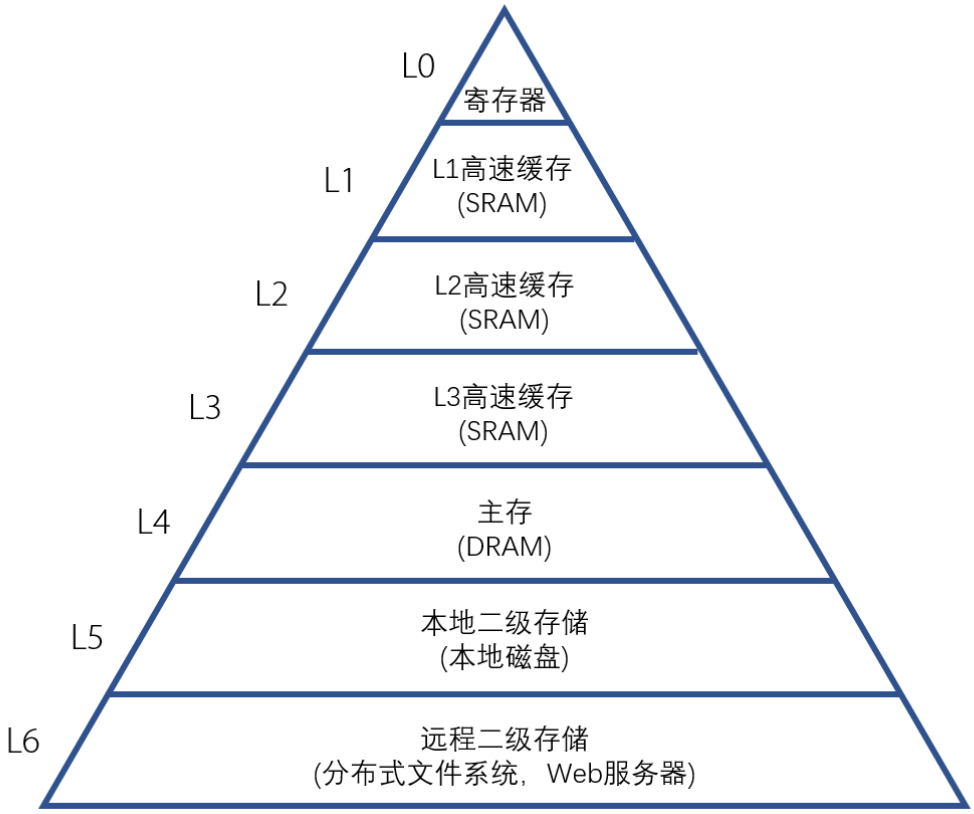
用 CPUZ 等查看你的计算机 Cache 各参数，写出 Cache 的基本结构与参数：缓存大小 C、分组数量 S、关联度/组内行数 E、块大小 B，及对应的编码位数 ：组索引位数 s、 e 、块内偏移位数 b

写出 Cache 的各种读策略与写策略

掌握 Valgrind、gprof 的使用方法

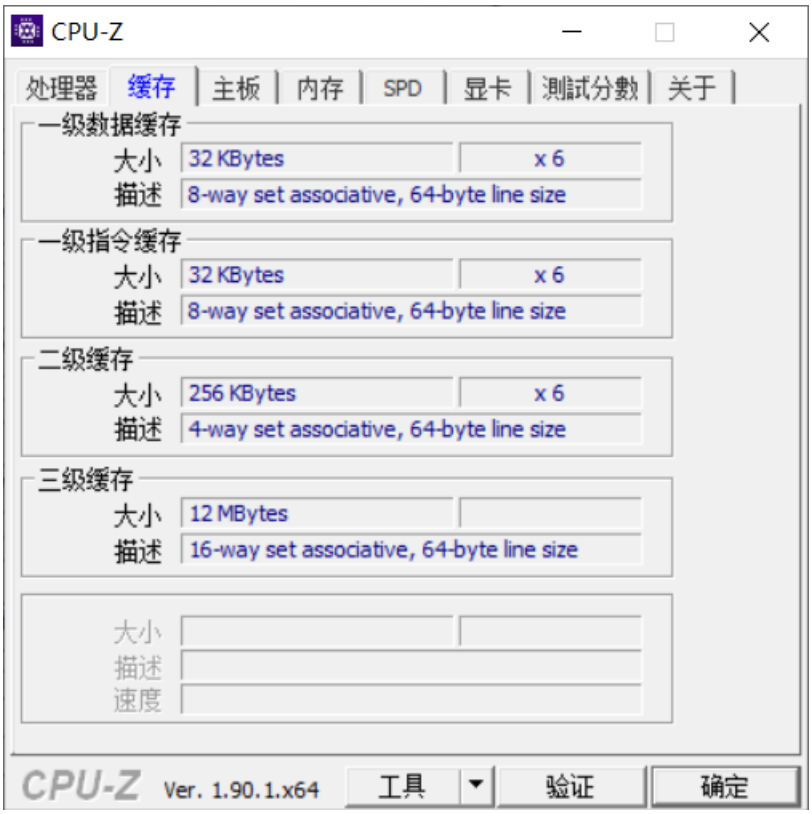
第 2 章 实验预习

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）



越往下价格越便宜，访问速度越慢，容量越大

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）



一级缓存：32KB，8 路组相连高速缓存，每块 64 字节

二级缓存：256KB，4 路组相连高速缓存，每块 64 字节

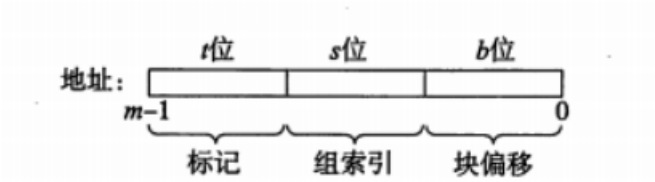
三级缓存：12MB，16 路组相连高速缓存，每块 64 字节

	C	S	E	B	s	e	b
一级缓存	32*1024	64	8	64	6	3	6
二级缓存	256*1024	1024	4	64	10	2	6
三级缓存	12*1024*1024	12288	16	64	$\ln_2 12288$	4	6

2.3 写出各类 Cache 的读策略与写策略（5 分）

1. 读策略

在读取一条地址的内容时，将该地址按照如下方式进行分解：



其中组索引表明该地址所需要的组号，块偏移位所表示的数 n 表示所需数据从该块的第 n 位开始读起。标记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，进行读取。

若命中，则直接从该层读取数据。

若不命中，则采用 LRU 策略进行驱逐和调换，空出位置从下一层存储读取数据。

2. 写策略

命中：两种策略。

①直写。立即将已经缓存了的字的高速缓存块写回到紧接着的低一层中

②写回。尽可能地推迟更新，只有当替换算法要驱逐这个更新过的块时，才把它写到紧接着的低一层中。

不命中：两种策略。

①写分配。加载相应的低一层中的块到高速缓存中，然后更新该高速缓存块

②非写分配。避开高速缓存，直接把这个字写到低一层中。

2.4 写出用 gprof 进行性能分析的方法（5 分）

使用 gcc -pg 选项，在编译程序时会在每个函数的开头添加 mcount 函数。程序在执行每个函数时都会先调用 mcount，而 mcount 则会在内存中保存一张函数调用图，通过函数调用堆栈的形式，查找子函数、父函数的地址，也保存了调用时间、调用次数等信息。最终在程序退出时，结果保存在 gmon.out 文件中，可供查看。需要注意的是，程序必须正常退出，在最后才会得到完整的 gmon.out 文件。综上，使用 gprof 进行性能分析需要以下步骤：

1. 使用 gcc -pg 选项编译程序
2. 运行程序并正常退出
3. 查看生成的 gmon.out 文件

2.5 写出用 Valgrind 进行性能分析的方法（5 分）

Valgrind 工具包包含多个工具，如 Memcheck,Cachegrind,Helgrind,Callgrind，Massif。

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：

1. initCache ( )函数

```
void initCache()
{
    //用二维数组来模拟cache，cache[i][j]代表第i组的第j行
    int i, j;
    // 初始化组
    cache = (cache_t)malloc(sizeof(cache_set_t) * S);
    for (i = 0; i < S; i++)
    {
        // 初始化行
        cache[i] = (cache_set_t)malloc(sizeof(cache_line_t) * E);
        for (j = 0; j < E; j++)
        {
            cache[i][j].valid = 0; // 有效位初始化
            cache[i][j].tag = 0;   // 标记位初始化
            cache[i][j].lru = 0;   // LRU策略标志初始化
        }
    }
}
```

先按组分配出 S 组空间，再为每一组分配 E 行的空间，分配完成后循环访问进行初始化。

2. freeCache ( )函数

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

先释放行空间，再释放组空间。

3. accessData ( )函数

算法设计思想如下：

首先，对地址 addr 中的各标志位进行提取。块偏移位在程序中没有使用过，先忽略；组索引位 s_digits 由 addr 右移 b 位，再通过移位运算将高位置 0 得到；标记位 t_digits 由 addr 右移 b+s 位得到。

之后，对每一组进行遍历，寻找是否有组索引匹配的行，若组索引匹配且有效位为 1，则命中，hit_count++；同时记录最后一次遇见的有效位为 0 的行，记为空行，便于后续在缓存不命中时读入放置数据。

在遍历完毕后，若未找到组索引匹配的行，则缓存不命中，miss_count++，同时进行下一步判断：若有空行，则将空行写入这次未命中的数据；若无空行，则执行 LRU 策略：寻找到最远最不常访问的块，并将其驱逐，再写入未命中的内容。

测试用例 1 的输出截图（5 分）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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）：

```
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
```

整体运行结果如下：

```
1190200208@MincooLee: ~/桌面/hitics/lab06_cache/cachelab-handout
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ gcc -c csim.c
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ gcc -c cachelab.c
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ gcc -o csim csim.o cachelab.o
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
hits:9 misses:8 evictions:6
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 4 -E 2 -b 4 -t traces/yi.trace
hits:4 misses:5 evictions:2
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 2 -E 1 -b 4 -t traces/dave.trace
hits:2 misses:3 evictions:1
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 2 -E 1 -b 3 -t traces/trans.trace
hits:167 misses:71 evictions:67
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 2 -E 2 -b 3 -t traces/trans.trace
hits:201 misses:37 evictions:29
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 2 -E 4 -b 3 -t traces/trans.trace
hits:212 misses:26 evictions:10
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/trans.trace
hits:231 misses:7 evictions:0
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$ ./csim -s 5 -E 1 -b 5 -t traces/long.trace
hits:265189 misses:21775 evictions:21743
1190200208@MincooLee:~/桌面/hitics/lab06_cache/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
1190200208@MincooLee:~/桌面/hitics/lab06_cache/cachelab-handout$
```

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：

cache(5,1,5)的具体参数为：

$s=5, b=5$

$S=2^5=32, E=1, B=2^5=32$

$C=2^{10}=1024$

即：

cache 共有 32 组，每组 1 行，每行存有 32byte（8 个 int 型）数据，整个 cache 一共可存 256 个 int 型数据。

我们可以先针对一个简单的条件，来研究不命中的情况。具体的分析写在了我的个人博客中：
https://blog.csdn.net/qq_45499104/article/details/117378797?spm=1001.2014.3001.5501

对于 32x32 的矩阵，整个 cache 最多存储 8 行的内容。按照我们分析的内容，可以发现：

第一次，cold miss，第 0 组保存 A[0][0]~ A[0][7]，写入 B[0][0]发生 conflict miss，第 0 组保存 B[0][0]~ B[0][7]；

第二次读入 A[0][1]，conflct miss，第 0 组保存 A[0][0]~ A[0][7]；写入 B[1][0]发生 cold miss，第 4 组保存 B[1][0]~ B[1][7]；

第三次读入 A[0][2]，命中，第 0 组仍保存 A[0][0]~ A[0][7]；写入 B[2][0]发生 cold miss，第 8 组保存 B[2][0]~ B[2][7]；

……

像这样，在一般情况的转置中，B 依次写入 1,2,3,...,8,9,10,...,32 行的某列元素，依次占用第 0,4,8,...,28,0,4,...,28 组。在写入该列元素前四分之一后，就会发生 conflict miss，miss 数因此大量增加；而如果一次只让 B 写八行的第一列，再写这八行的下一列直至写完前八列，就不会发生 miss/eviction（当然第一列会 miss）。

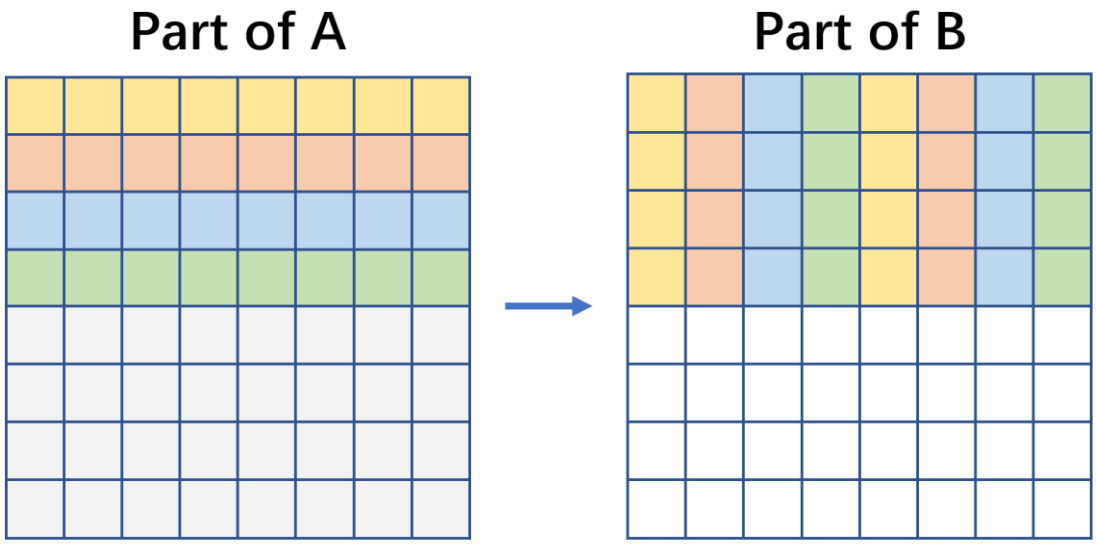
因此，为减少 miss 数，我们采用分块的思想。将矩阵划分为 8x8 的块，可以发现每一块与它对应的转置部分都映射到 cache 中不同的组，不会产生 conflict miss，这样遍历每一块中的元素，便可以极大减少 miss 数；另另外，在分块矩阵的主对角线上时，B 每写一列（八行）会与 A 冲突一次，导致两次 miss，为减少这部分的消耗，我们另设 8 个变量 t0~t7，在访问 8x8 矩阵块时，恰好可以保存一整个 cache line，取完一行以后便不再访问这行，以此减少 conflict miss。

对于 64x64 的矩阵，cache 最多只能存储 4 行的内容。如果还是按照 8x8 进行分块，读取 A 中分块的一整行，仍然只有一次 miss，但按列给 B 赋值时，写入每个元素都会有 conflict miss，这导致 miss 数会非常多；若按 4x4 分块，每块的利用

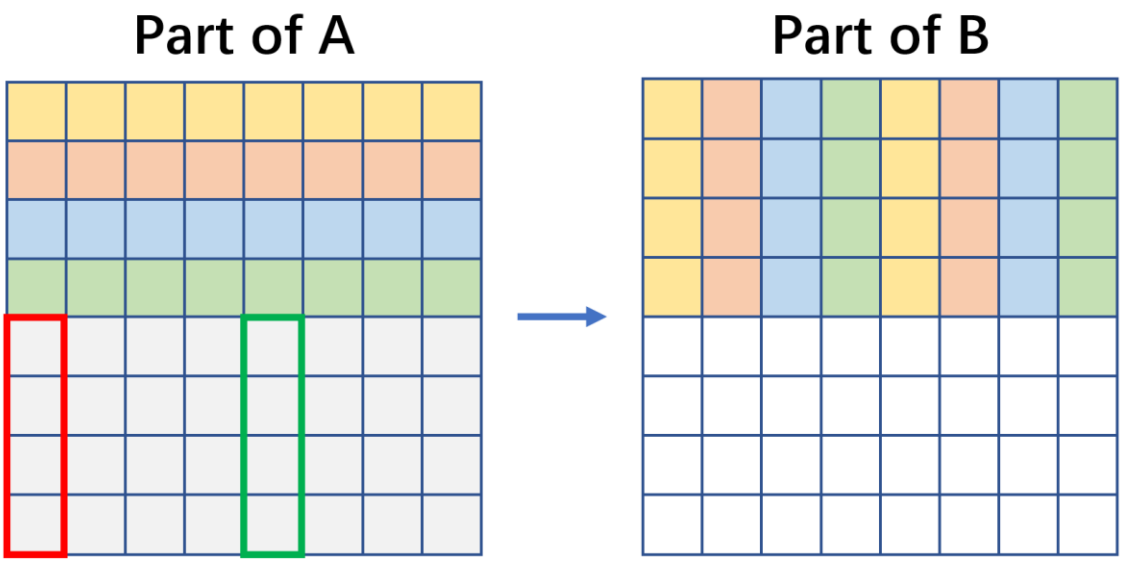
率只有一半，经检验 miss 数仍接近 2000。

具体我的处理思路是先进行 8 分块，再进行 4 分块，按以下算法处理：

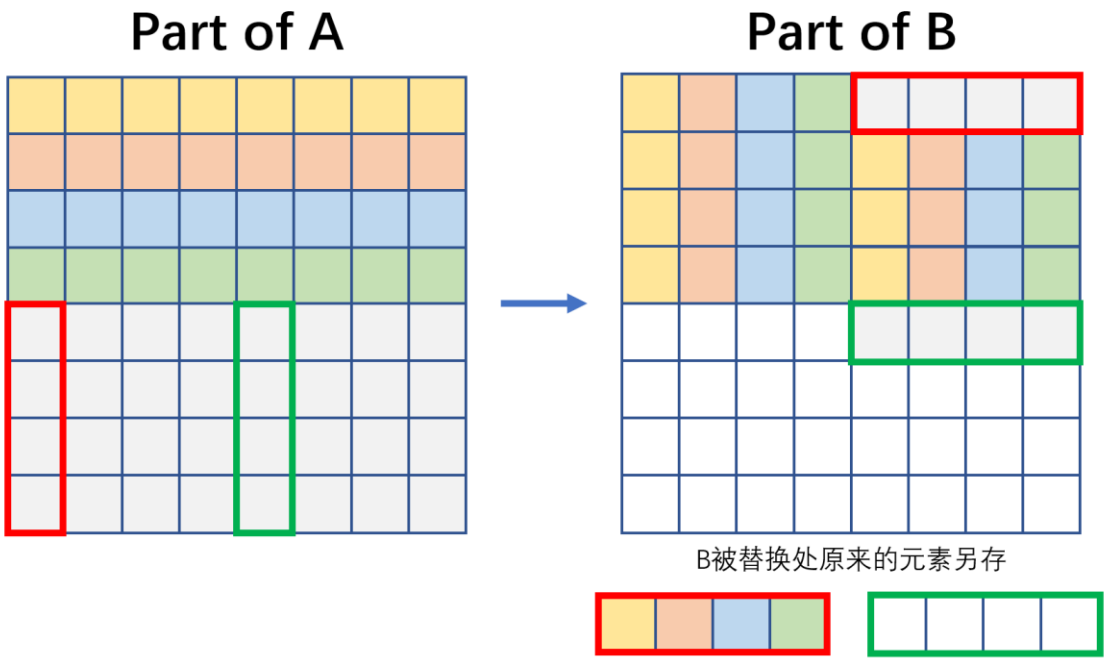
1. 对 A 前四行的前八个数分别按顺序存到 B 的前四列与后四列中



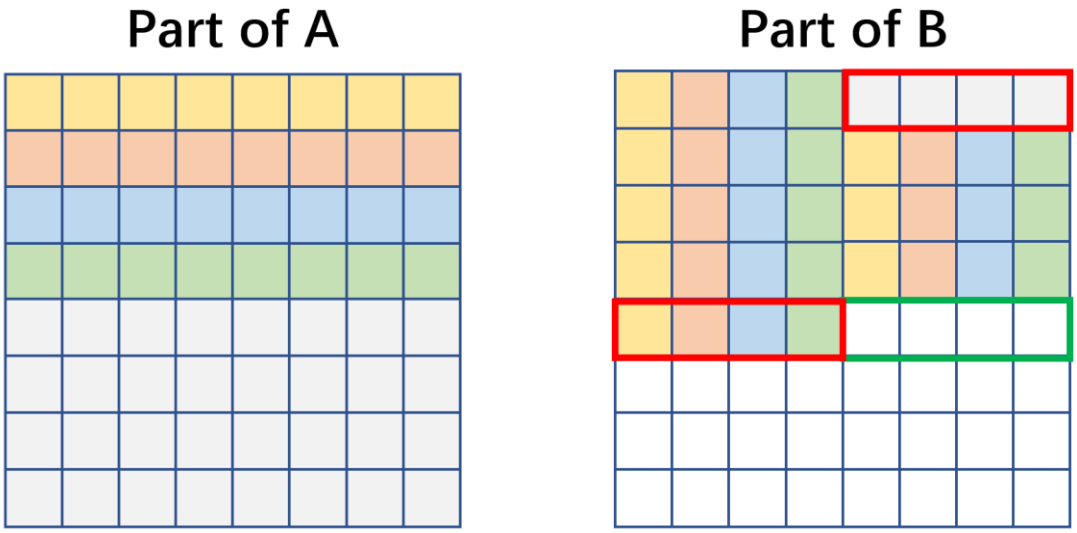
2. 读取并保存 A 后四行的第一列、第五列元素



3. 将 2 中 A 的后四行的第一列，放到 B 当前第一行的后四个元素中（并保存之前这里的四个元素）；将 2.中 A 的后四行的第五列，放到 B 第五行的后四列



4. 将 3 中存下的，原先 B 第一行的后四个元素，放到 B 第五行的前四列



5. 2,3,4 步对不同列重复做共四次，就可以实现转置

总的来说，前四行可以直接按列赋值给 B，然后对后四行，左下角部分的每一列放到右上角部分的每一行、右上角的每一行放到左下角部分的每一行、右下角的列转成行位置，就可以实现转置

对于 61x67 的矩阵，由于其与 cache 的容量不存在倍数关系，所以分析起来较为复杂。在尝试了多种分块方法后，发现 17x17 的分块方法另外再处理对角线元素可以达到 miss<2000 的效果。对于如此分块剩下的元素，单独处理即可。

32×32（10 分）：运行结果截图

```
1190200208@MincooLee:~/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

```
1190200208@MincooLee:~/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: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

```
1190200208@MincooLee:~/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:3756, misses:4423, evictions:4391

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

1. 对现代计算机系统的存储器层级结构有了更深的了解
2. 更好地认识 cache 的组织方式与读写操作
3. 对 Linux 下利用工具进行性能测试的方法有了了解
4. 深入理解 Cache 组成结构对 C 程序性能的影响
5. 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

在实验中添加部分对 cache 基本结构与操作的讲解

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.