



华中科技大学

计算机系统结构实验报告

姓 名：龙际全

学 院：计算机科学与技术

专 业：计算机科学与技术

班 级：CS1603

学 号：U201614577

指导教师：陈俭喜

分数	
教师签名	

2019 年. 04 月. 25 日

目 录

1. 第一部分：Cache 模拟器	3
1.1. 实验目的	3
1.2. 实验环境	3
1.3. 实验思路	3
1.4. 实验结果和分析	5
2. 第二部分：矩阵转置优化	6
2.1. 实验目的	6
2.2. 实验环境	6
2.3. 实验思路	6
2.4. 实验结果和分析	7
3. 总结和体会	9
4. 对实验课程的改进建议	9

1. 第一部分：Cache 模拟器

1.1. 实验目的

- (1) 理解 cache 工作原理；
- (2) 加深 Cache 缓存组成结构对 C 程序性能的影响的理解。

1.2. 实验环境

- (1) 操作系统：Manjaro 64 bit；
- (2) 编译器：GCC 8.2.1；
- (3) IDE：Visual Studio Code。

1.3. 实验思路

- (1) 内存地址采用组相联映射，一个内存地址具有如下图 1-1 所示形式，
set_index 用于确定内存地址映射到的组，tag 用于在组里用全相联的方式寻找所在的 cache 行，block_offset 为行内偏移地址。



图 1-1

- (2) 该实验只是模拟 cache 工作原理中的一小部分(Hit/Miss/Eviction)，因此我们可以看到实验框架对 cacheline 进行了非常简单的抽象，如下图 1-2 所示：



图 1-2

- (3) Cache 替换采用 lru 算法 ,lru 即最近最经常使用(least recently used)。检查 cache 某一组内所有 cacheline 的 tag 是否与输入的 tag 相等 , 如果相等且 cacheline 的 valid 位为 1 则表示 cache 访问命中 , 此时 lru 计数器清零 , 相反 , 对于不命中的 cacheline , lru 计数器加 1 ; 如果没有命中的 cacheline , 即 cache 不命中 , 则需要把主存块调入 cache set(cache 组)中 , 先去找 cache 空闲即 valid == 0 的那些 cacheline ; 如果没有空闲的 cacheline , 则此时需要对 cache set 进行替换 , 替换策略是将 lru 计数器最大的 cacheline 替换掉。
- (4) 具体到本实验 , cache 为二维结构如下图 1-3 所示 , 先通过 address 和 set_index 定位到特定的 cacheset , 然后用循环遍历 cacheset 里所有的 cacheline 即可判断是否命中并对 lru 计数器作相应处理 , 对于 Miss 的两种情况 : 1 , 有空闲 , 直接将 tag 写入空闲 cacheline 并置 valid 位为 1 ; 2 , 没有空闲 , 将 tag 写入 lru 最大的 cacheline 并置 lru 为 0 即可。

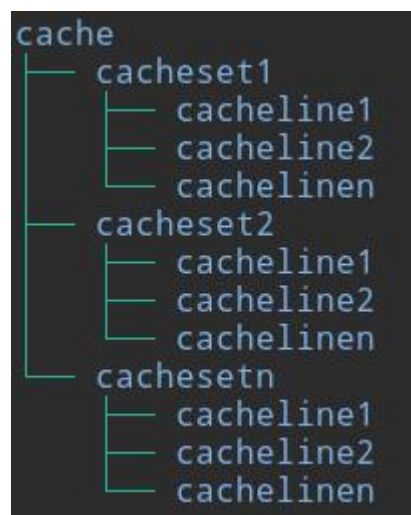


图 1-3

- (5) 此外 , 还需要编写 replayTrace 对输入指令进行相应的处理 , 对于 Load/Store 指令 , 只需要访问主存一次 , 因此只需要调用一次 accessData ; 由于实验仅关心数据 Cache 的性能 , 因此模拟器应忽略所有指令 cache 访问 ; Modify

指令需要访问两次主存，因此需要调用两次 `accessData`。

1.4. 实验结果和分析

(1) 单例 `./csim(-ref) -v -s 4 -E 1 -b 4 -t traces/yi.trace` 运行结果如下图 1-4：

```
[ljq@ljq-pc cachelab-handout]$ ./csim -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
[ljq@ljq-pc cachelab-handout]$ ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
[ljq@ljq-pc cachelab-handout]$
```

图 1-4

(2) 全部样例 `./test-csim` 运行结果如下图 1-5：

```
[ljq@ljq-pc 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
[ljq@ljq-pc cachelab-handout]$
```

图 1-5

(3) 命中数 Hits、不命中数 Misses、淘汰数 Evicts 均与标答吻合。

2. 第二部分：矩阵转置优化

2.1. 实验目的

- (1) 加深对 cache 工作环境的理解；
- (2) 能针对系统 cache 结构对矩阵转置程序进行优化，使得程序运行过程中 cache 不命中数较少以此提高程序性能。

2.2. 实验环境

- (1) 操作系统：Manjaro 64 bit；
- (2) 编译器：GCC 8.2.1；
- (3) IDE：Visual Studio Code；
- (4) 软件包：valgrind。

2.3. 实验思路

- (1) 矩阵转置的常规思路是用两层 for 循环，将一行直接转为一列，这种方法实现简单，代码量小，在不考虑性能的情况下是最优的。不过，要考虑 cache 命中率的话，这种方法的性能就非常差了，因为行遍历能够很好地利用 cache，但是列访问不能，所以要采取一种折中的方法，使得行和列的大小都不太大，从而减少 cache 的缺失。很容易想到对矩阵进行分割。具体分成多大一块要根据矩阵的大小来决定，不妨先设为 8×8 ，因为一个 cache 行的大小为 8 个字节。不过，cache 总共只有 8 行，行访问要占用 1 行，如果 8 个列访问各占一行，总共会有 9 行，必有一行被淘汰，为了防止淘汰，要对对角线上的

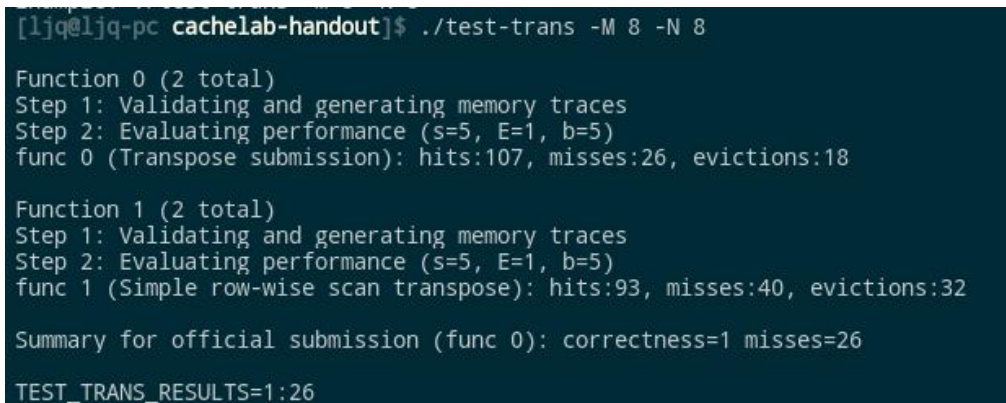
元素另行处理，用变量保存。

- (2) 用两个 for 循环，将大矩阵分割成 8×8 的小矩阵，对于每个小矩阵，再用一个 for 循环，将 A 矩阵的行转到 B 矩阵的列。每次循环都要判断是否为对角线上的元素，若是则用局部变量存起来，循环结束后再写入 B。部分代码如下：

```
for(int k = 0; k < M; k+=8) {
    for(int l = 0; l < N; l+=8) {
        for(int i = k; i < k+8; i++) {
            for(int j = l; j < l+8; j++) {
                if(i != j) {
                    B[j][i] = A[i][j];
                } else {
                    tmp = A[i][j];
                    index = i;
                }
            }
            if(l == k) {
                B[index][index] = tmp;
            }
        }
    }
}
```

2.4. 实验结果和分析

- (1) 运行 ./test-trans -M 8 -N 8 结果如下图 2-1 所示：



```
[ljq@ljq-pc cachelab-handout]$ ./test-trans -M 8 -N 8

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:107, misses:26, evictions:18

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:93, misses:40, evictions:32

Summary for official submission (func 0): correctness=1 misses=26

TEST_TRANS_RESULTS=1:26
```

图 2-1

(2) 运行./driver.py 结果如下图 2-2 所示：

```
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	8.0	8	1219
Trans perf 61x67	10.0	10	1985
Total points	53.0	53	

```
[ljq@ljq-pc cachelab-handout]$
```

图 2-2

(3) 优化后的程序满足实验要求。

3. 总结和体会

- (1) 这次实验进一步加深了我对 cache 的理解,我之前在组成原理课程中分别用 verilog 和 logisim 实现过 cache,但这次实验仍然有小小的翻车,因为具体电路比较是否命中是全部并行比较 tag 的,而在代码中为串行循环比较,我在 cache 命中后 break 跳出了循环导致后面没有命中的 cacheline 的 lru 计数器没有发生改变,导致最终淘汰算法出错。
- (2) 这次实验让我了解到底层原理和实现对上层应用软件和算法的重要性,一个好的算法能很大程度提升软件性能。

4. 对实验课程的改进建议

- (1) 注意到实验选自 CSAPP(深入理解计算机系统)第三次 cache 实验,希望可以给出完整的实验文档和实验链接。
- (2) 实验 2 矩阵转置优化可以计算 cache 理论上的 hit 和 miss 数,可以将这个作为任务之一。