

**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 姓 名： |  |
| 学 院： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 班 级： |  |
| 学 号： |  |
|  |  |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2023 年. 5月. 15日

**目 录**

[1 Cache模拟器实验 3](#_Toc135063438)

[**1.1** **实验目的** 3](#_Toc135063439)

[**1.2** **实验环境** 3](#_Toc135063440)

[**1.3** **实验思路** 3](#_Toc135063441)

[**1.4** **实验结果和分析** 7](#_Toc135063442)

[2 优化矩阵转置 8](#_Toc135063443)

[**2.1** **实验目的** 8](#_Toc135063444)

[**2.2** **实验环境** 8](#_Toc135063445)

[**2.3** **实验思路** 8](#_Toc135063446)

[**2.4** **实验结果和分析** 11](#_Toc135063447)

[3 总结和体会 11](#_Toc135063448)

[4 对实验课程的建议 11](#_Toc135063449)

# Cache模拟器实验

* 1. **实验目的**

理解cache工作原理；

加深Cache缓存组成结构对C程序性能的影响的理解

* 1. **实验环境**

头歌

* 1. **实验思路**

1. 定义cache块的数据结构

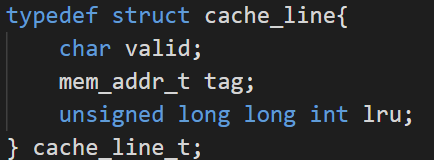


图1.1 代码1

valid：有效位

tag：标识，内存地址的高位部分

lru：计数器，用于LRU替换算法。

1. 定义全局变量

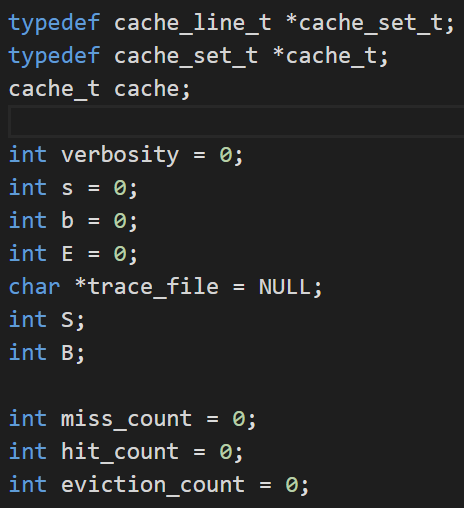


图1.2 代码2

s、b、E分别表示相联组数的位数、块内偏移的位数、组内块数的位数。

S、B分别为相联组数和块大小

Miss\_count，hit\_count，eviction\_count表示不命中次数、命中次数、淘汰次数

1. 编写cache初始化和释放函数

init：先给整个cache分配内存，再给每一个cache组分配内存，再给组里的每一块cache的参数全部设置为0

free：释放每一组cache，再释放整个cache的内存

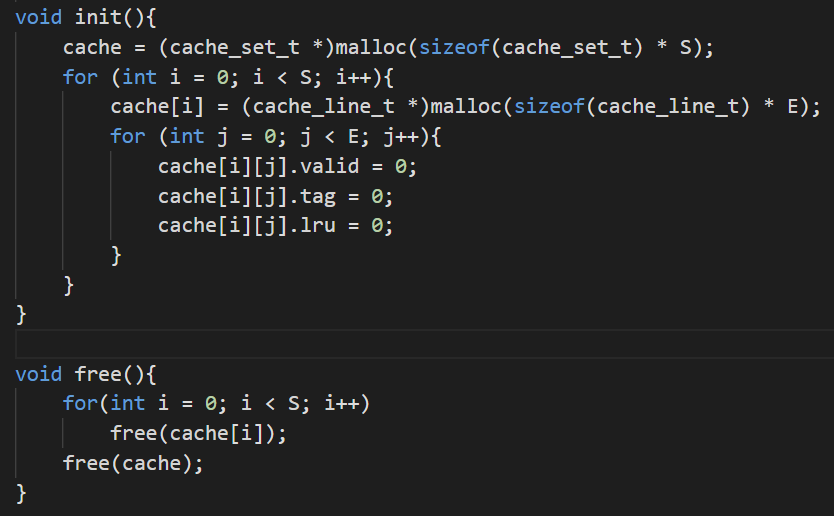


图1.3 代码3

1. accessdata函数：cache访问过程

变量：eviction\_lru：cache组中最小的lru

eviction\_line：最小的lru的行数。

hit\_index：命中的行号

invalid\_index：空闲的行号

index：cache的索引

tag：cache的标识。

1. 遍历cache组。找到lru最小的有效行，更新eviction\_lru和eviction\_line。如果命中，更新hit\_index。如果有空行，更新invalid\_index。

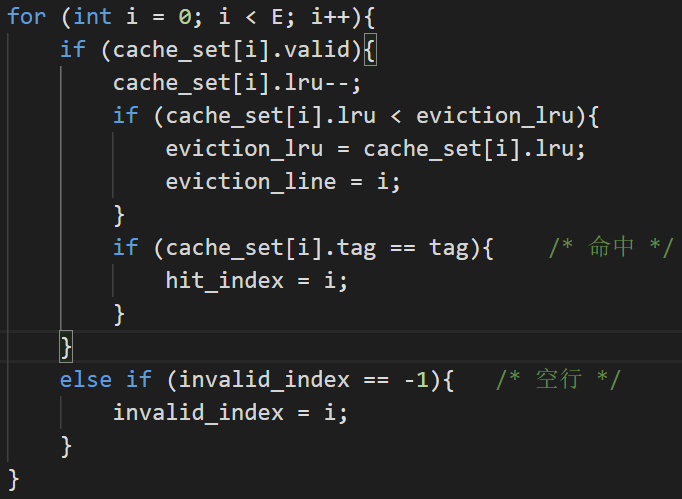


图1.4 代码4



图1.5 流程图1

1. 遍历过程中，如果有命中，输出行信息，重置hit\_index和该行的lru。

如果不命中：如果有空行，将数据写入空行，重置invalid\_index；否则将数据写入lru最小的行，重置eviction\_lru。



图1.6 代码5



图1.7 流程图2

1. replaytrace函数
2. 读取文件内容

变量：buf：操作指令

addr：访问的内存地址

len：访问的地址空间的长度。

1. 如果指令为读写指令，调用一次accessdata；如果为修改指令，需要先读再写，所以调用两次。

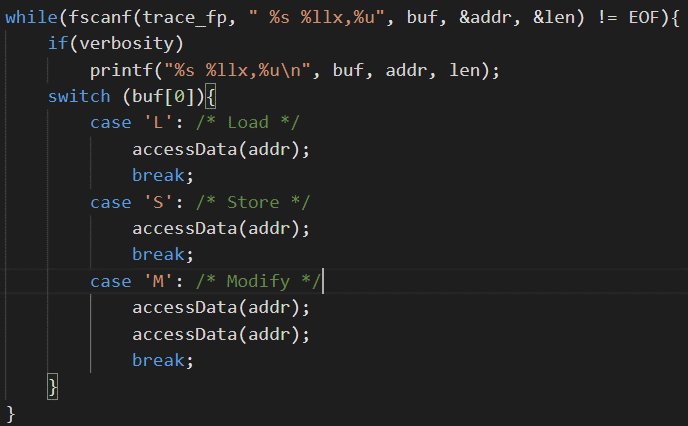


图1.8 代码6

* 1. **实验结果和分析**

符合实验要求

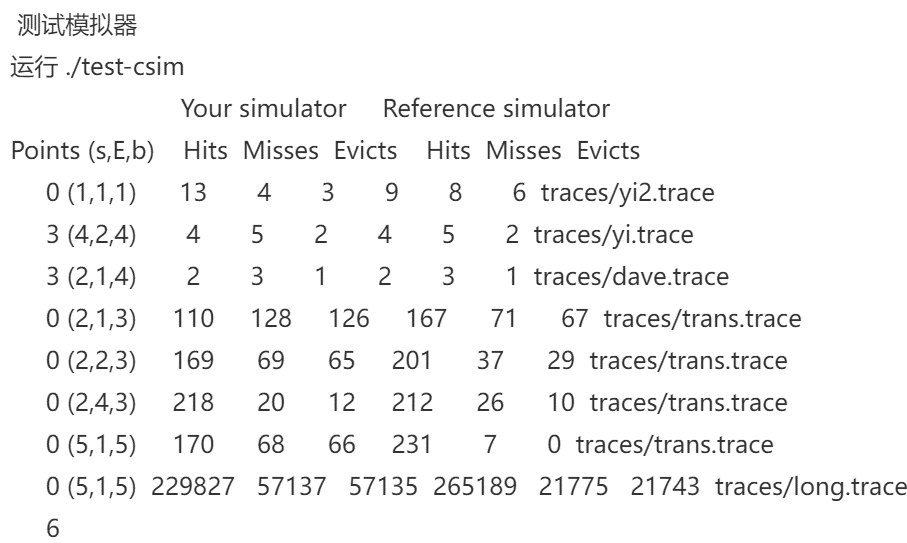


图1.9 实验一结果

# 优化矩阵转置

1. **实验目的**

理解cache工作原理。

加深cache缓存组成结构对c程序性能的影响的理解。

理解各种造成cache的不命中的原因和优化cache不命中的方法。

1. **实验环境**

头歌

1. **实验思路**
2. 32\*32矩阵

因为一个cache可以放8个int，所以一行要写进4个cache。

将A分为8\*8个数字大小的块，映射的cache为1\*8个。



图2.1 每个分块映射cache

如果不是对角线上的块，互相对应的两个8\*8的矩阵块在cache里面恰好没有冲突，miss最小。直接将A的数字写入B的对应位置。

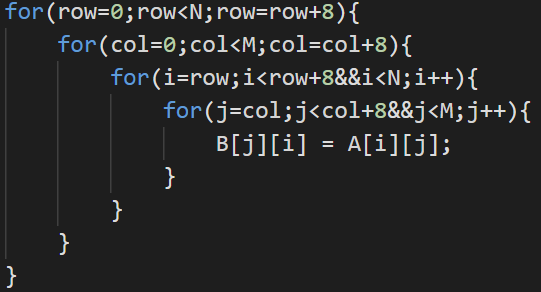


图2.2 代码7

但对角线上的块仍然会发生多次冲突不命中，因此将每行的8个数字，即映射到同一个cache的大小放入临时变量，B再读取临时变量进行转置，就不需要多次访问cache块，减少了冲突不命中。

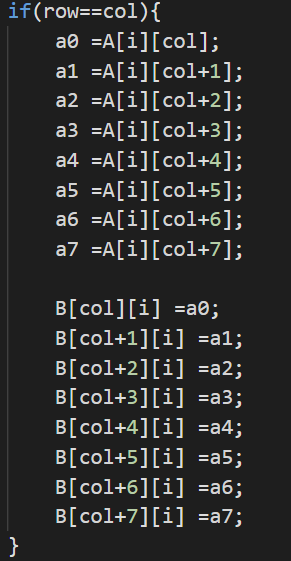


图2.3 代码8

1. 64\*64矩阵

和32\*32矩阵一样先按8\*8分块，每个分块再按4\*4分块。



图2.4 每个分块中的分块映射cache

对于每一个8\*8的分块，当我们读一个分块内，即黑框内左上块的数字时，右上块的数据也会被写到cache中，为了减少写的次数，将这两个分块同时进行操作。右上的块如果在B中直接放到转置位置，即左下块，会造成多次冲突不命中。

因此把这4个cache大小的数据移动到B中时，只转置分块内数字的位置，整个分块的位置暂时不进行转置，即放到右上块，这样B中映射的cache就只有4块，减少了冲突不命中。



图2.5 步骤1

将B中右上的16个数移动到左下，同时将A的左下16个数移动到B的右上



图2.6 步骤2

最后将A的右下块移动到B的右下块，转置完成。



图2.7 步骤3

1. 67\*61矩阵

67\*61的矩阵无法正好按块分割。运行不同的分块方案得到结果，发现块大小分割为17\*17时不命中次数最少。

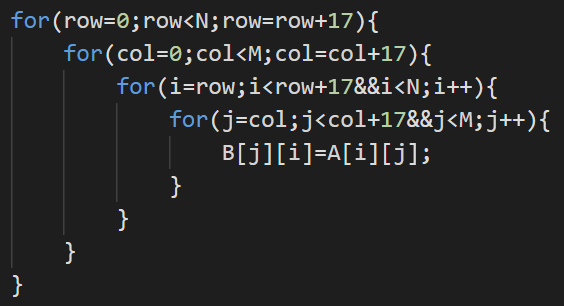


图2.8 代码9

1. **实验结果和分析**

符合实验要求



图2.9 实验二结果

# 总结和体会

这两个实验对我来说都很有挑战性。实验一是对cache基本工作原理的理解和运用。第二个实验比较抽象，需要自己画图来理解，但实验说明中也给出了解释和思路的提示，所以降低了一点难度。同时在实验二中，不管是优化的方法还是优化的数据，都需要不停尝试才能找出最优解，我在这个过程中受益匪浅。两个实验都是对课堂上的理论知识的复习和运用，我在实验过程中收获了很多。

# 对实验课程的建议

希望实验二能给出更具体的解释和思路提示，实在有点抽象。