

**计算机系统结构实验报告**

|  |  |
| --- | --- |
| 姓 名： | 王嘉成 |
| 学 院： | 计算机科学与技术 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS2207 |
| 学 号： | U202215576 |

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

2025年 5月 6日

**目 录**

[1. Cache模拟器实验 3](#_Toc16893)

[1.1. 实验目的 3](#_Toc13499)

[1.2. 实验环境 3](#_Toc17572)

[1.3. 实验思路 3](#_Toc22514)

[1.4. 实验结果和分析 5](#_Toc24743)

[2. 优化矩阵转置 5](#_Toc14088)

[2.1. 实验目的 5](#_Toc10185)

[2.2. 实验环境 5](#_Toc16583)

[2.3. 实验思路 6](#_Toc7993)

[2.4. 实验结果和分析 7](#_Toc27220)

[3. 总结和体会 8](#_Toc6354)

[4. 对实验课程的建议 8](#_Toc20685)

# Cache模拟器实验

* 1. **实验目的**

本实验旨在通过实现一个Cache模拟器，深入理解计算机系统中Cache的工作原理及其性能影响因素。通过编写能够处理不同配置参数的Cache模拟程序，我们将掌握直接映射、组相联等Cache组织结构的特点，并实践实现LRU替换算法来处理Cache冲突。实验要求模拟器能够分析给定的内存访问轨迹，统计命中、缺失和替换次数，从而评估Cache性能。

* 1. **实验环境**

操作系统环境：Linux;

编程语言：C语言;

* 1. **实验思路**

这个Cache模拟器的核心思想是通过软件方式重现CPU访问内存时Cache的工作机制。程序首先会读取用户通过命令行输入的Cache配置参数，包括组索引位数(s)、相联度(E)和块偏移位数(b)，基于这些参数，程序会动态构建一个模拟的Cache结构，然后读取指定的内存访问轨迹文件，逐条分析其中的内存访问操作。对于每个有效的数据访问操作，程序都会模拟其在Cache中的查找过程，并根据查找结果更新命中、缺失和替换的统计信息，最后输出这些统计结果，从而评估该Cache配置下的性能表现。

程序使用了一个二维数组Cache来模拟组相联Cache的物理结构，这个结构的第一维度代表Cache的组数，第二维度代表每组的行数。每个Cache行是一个结构体，包含三个关键字段：valid位表示该行是否存储了有效数据；tag字段存储标记位，用于地址匹配；count字段是实现LRU替换策略的计数器，记录该行最近被使用的情况。这种设计既反映了真实的Cache硬件结构，又便于在软件中实现各种Cache操作和策略。

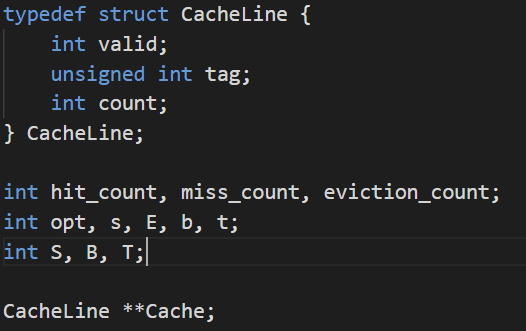


图 1 数据结构

searchCache函数实现了Cache的核心查找逻辑。它首先从内存地址中提取出tag和组索引，然后在对应的组内遍历所有行，检查是否有匹配的tag。如果找到匹配且valid为1，就是命中，此时要重置该行的LRU计数器。如果没有命中，则需要考虑分配新的Cache行：优先使用组内invalid的行，如果没有可用行，则执行LRU替换。LRU策略通过维护每个行的count值来实现，每次访问都递增组内所有有效行的计数器，被访问的行计数器清零，替换时选择计数器最大的行。

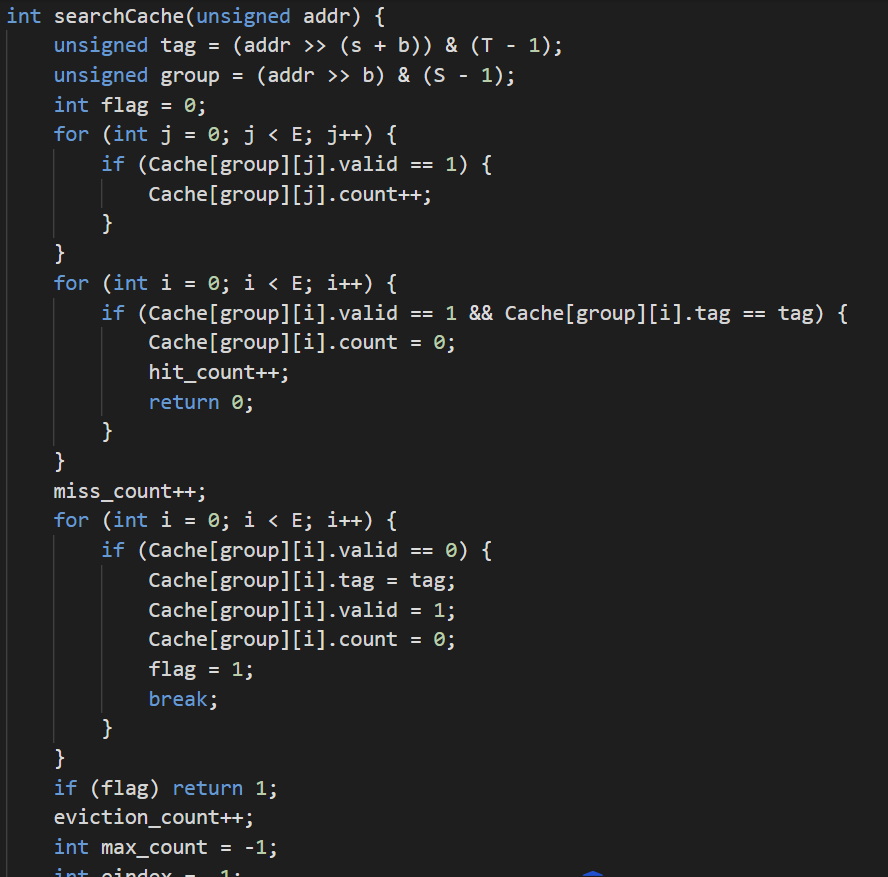


图 2 searchCache函数

程序逐行读取trace文件，每行包含一个操作类型和内存地址。对于L（加载）和S（存储）操作，执行一次Cache查找过程，对于M（修改）操作，相当于先加载后存储，因此需要执行两次Cache查找。程序会智能地忽略所有以I开头的指令加载操作，因为实验只关注数据Cache的性能。每个地址访问都会经过完整的Cache查找流程，并实时更新各项统计计数器。

程序维护了三个关键的性能统计计数器：hit\_count记录Cache命中的次数，miss\_count记录Cache缺失的次数，eviction\_count记录因Cache满而需要替换行的次数。这些计数器会在每次Cache查找过程中根据结果进行更新，最终的统计结果可以直观反映当前Cache配置下的性能表现，命中率越高说明Cache效率越好，而频繁的替换则可能表明Cache容量不足或相联度不合适。

* 1. **实验结果和分析**

实验结果如图所示。

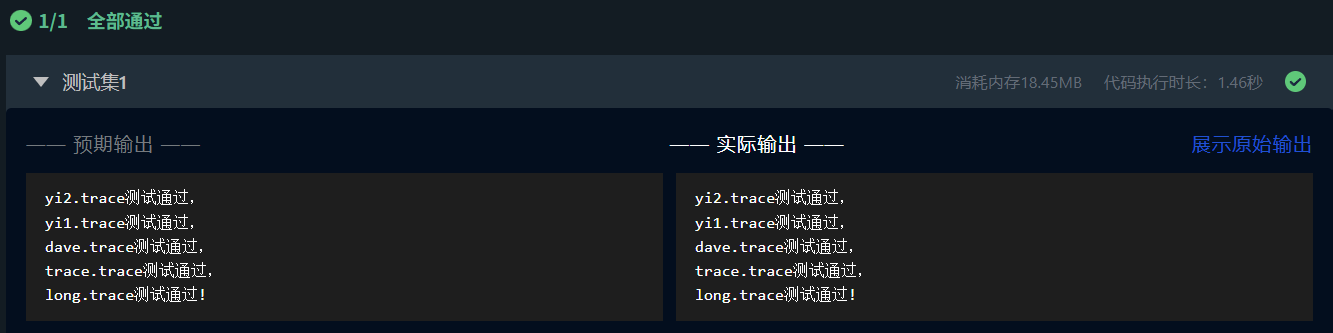


图 3 实验结果

# 优化矩阵转置

* 1. **实验目的**

实验目的是通过编写一个高效的矩阵转置函数transpose\_submit，将给定的M×N矩阵A转置为N×M矩阵B（即B[j][i] = A[i][j]），同时优化函数对缓存的访问模式，尽可能减少缓存不命中次数。实验要求严格限制局部变量的数量、禁止递归或栈帧变量超额、禁止修改矩阵A或定义额外矩阵/动态内存分配，从而使专注于优化矩阵访问的局部性，提升缓存命中率。

* 1. **实验环境**

操作系统环境：Linux;

编程语言：C语言;

* 1. **实验思路**

对于32×32矩阵，采用8×8分块策略。外层循环以步长8遍历矩阵的行和列，内层循环一次性读取A的8个连续元素，并直接转置存储到B的对应位置。这种方法充分利用了缓存行的存储特性，使得A和B的访问模式更加友好，减少了缓存冲突不命中，从而提升整体性能。

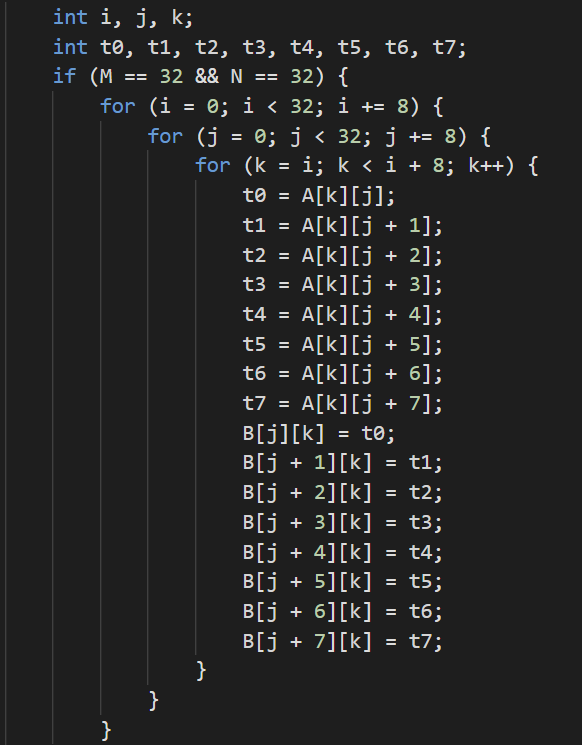


图 4 32 x 32矩阵处理策略

由于64×64矩阵直接使用8×8分块会导致A和B的缓存行冲突，因此采用两阶段分块策略。第一阶段处理A的前4行8列，转置后存入B的前4行8列，但部分数据暂存；第二阶段交换B的右上和左下部分，避免缓存冲突；第三阶段处理A的后4行8列，完成转置。这种方法通过局部转置+数据交换方式减少缓存行冲突，提高命中率。

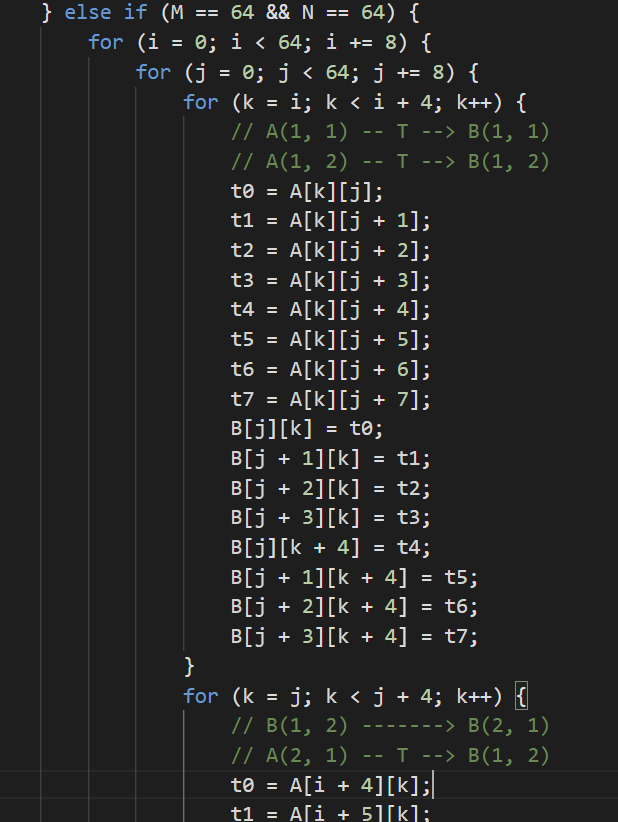


图 5 64 x 64矩阵处理策略

对于不规则尺寸的61×67矩阵，采用16×16分块策略。外层循环以步长16遍历行和列，内层循环处理当前块内的所有元素，并在边界处检查剩余元素，防止越界访问。这种方法在非对齐矩阵下仍能保持较好的局部性，减少缓存不命中，同时确保正确转置所有数据。

* 1. **实验结果和分析**

实验结果如图所示。

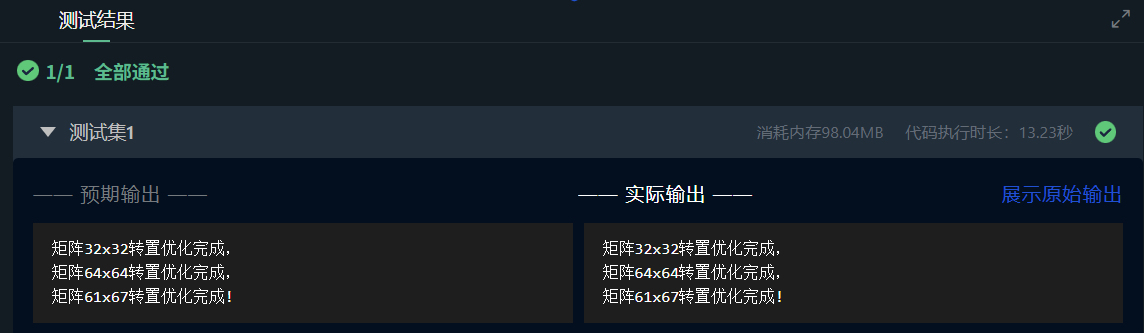


图 6 实验结果

# 总结和体会

通过本次Cache模拟器实验，我深刻理解了计算机体系结构中Cache的工作原理和性能影响因素。实验过程中，我不仅掌握了组相联Cache的组织结构和LRU替换算法的具体实现，还通过分析不同配置下的命中率变化，直观认识到Cache大小、相联度和块大小对系统性能的关键作用。在调试过程中，处理边界条件和验证LRU策略正确性的经历让我体会到计算机系统设计中精确性的重要性。而优化矩阵实验转置实验让我深入理解了缓存优化的重要性。针对不同矩阵尺寸，我采用分块策略来提升局部性，减少缓存不命中。实验证明，优化内存访问模式比算法本身更能显著提升性能，这让我认识到结合底层存储特性进行优化是关键。

# 对实验课程的建议

建议实验课程增加更多实际缓存优化案例，并引入性能分析工具来直观展示优化效果，帮助同学更深入理解计算机体系结构对程序性能的影响。

