Cachelab Discussion

蔡绘霓

Part A

- 目标：写一个程序来模拟cache
- 一些注意事项：
- 1、在程序开头的注释中注明姓名和学号
- 2、需要使用malloc/calloc函数为cache的数据结构分配存储
- 3、必须调用cachelab.h中定义的函数printSummary来进行输出，头文件中需要包含#include “cachelab.h”。
- 4、建议使用getopt函数来解析命令行参数，头文件中需要包含#include<getopt.h> #include<stdlib.h> #include<unistd.h>
- 5、使用LRU（最近最少使用）策略来进行行替换
- 总体来说难度不大，思路上没有难点，注意细节即可。

Part A

- Tips:
- 1、我们只是模拟组选择、行匹配这两个过程，而不考虑对block的操作。所以L（数据加载）、S（数据存储）其实是等价的，M（数据修改）也就等价于连续两次L/S，还可以进一步简化为L/S后hit++即可。
- 2、同上理，size这个变量对于我们来说没有用处。
- 3、执行LRU策略时，我们可以为每一行维护一个时间变量T代表上次命中的时间；遍历所有行后，最小的T即是最近最少使用的行（同理维护未命中时间也可以）。但注意，如果该组内还有空行，应优先存储在空行中。
- 4、将输入的字符串转化为整数可利用atoi函数。

Part B

- 目标：编写一个矩阵转置函数，使得不命中数尽量少。
- 一些注意事项：
- 1、在程序开头的注释中注明姓名和学号。
- 2、每个转置函数最多允许定义12个int类型的局部变量，也不允许通过使用任何类型为long的变量或使用任何位技巧将多个值存储到单个变量来绕过这条规则。
- 3、检测程序时只有这三种输入：32×32、64×64、60×68，所以我们只需要考虑这三种情况，可以检查输入大小从而实现针对每种情况优化的单独代码。
- 4、利用linux> ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f0命令可以查看函数每次S（数据存储）和L（数据加载）的地址以及命中情况。

Part B：32×32

- Tips:
- 1、cache的参数是s=5，E=1，b=5。这说明一行可以储存32字节，即8个int的数据。于是我们可以将矩阵分割成8×8的小矩阵来处理（writeup中也提到了blocking的思想）。
- 2、注意到目标矩阵和原矩阵的地址之差是32的倍数，这意味着同时加载和存储相同位置的元素会造成驱逐，于是增大了不命中次数。而这种情况会在转置对角线元素时发生。所以利用临时变量一次性取出A中8个元素再放入B中可以一定程度上避免这个问题。

Part B：64×64

- Tips:
- 1、思路与32×32类似。但出现了新的问题：cache中只有32组，只能存储4×64的元素，于是处理每个8×8的块时，前4行会与后4行相互驱逐。如果分割成4×4的块处理，就浪费了cache一次可以储存8个int的空间，也无法做到满分。于是，我们可以还是分割成8×8的矩阵，但在处理时再细分成4个4×4的小矩阵，先处理矩阵的前4行以及转置到的目标矩阵的前四行，再处理后四行（或者说处理第五行时，第一行元素已经不会再使用；处理第六行时，第二行元素不会再使用），从而最大程度避免前4行和后4行的互相驱逐。
- 2、32×32矩阵中转置对角线元素时会互相驱逐的问题仍然存在。处理方法也基本相同。

Part B：60×68

- Tips:
- 1、基本思想仍是分块。但由于大小并不规则，所以块的大小不易确定。经过尝试不同的分块大小，也很难发现分块大小与不命中次数之间明显的规律。但是，经过大（bao）量（li）尝试，可以发现一些分块大小是可以做到满分的。

Evaluation for Style

- 1、添加注释。
- 2、注意缩进。
- 3、单行代码不要太长（>80字符）。
- 4、记得考虑会使程序出错的情况。比如malloc分配空间后检查是否分配成功。
- 5、尽量使代码的可读性良好，包括但不限于变量、函数的命名，简化逻辑等等。

Thanks for watching !