cachelab实验报告

实验目的

- Part A. 用 C 语言实现一个 cache simulator。
- Part B. 优化矩阵转置算法减少其 cache miss, 并利用 Part A 中实现的算法计算该算法的 cache miss 次数。

实验内容

partA

定义cache数据结构

per_block为cache中一组中的一行；set为元素为行的数组；cache包含set数，每个set的line数，以及两者对应的比特数还有set列表；operation是测试文件指令的数据结构，包含操作类型，访问字节数，和开始访问的地址。

定义相关函数

- getSet：先将address右移b位，后对2的s次方取模取出后s位。相当于取出了地址中组索引的s个字节。
- getTag：address右移b+s位，获取前t=(m-b-s)位的行标识。
- allocateset：先通过s计算出组数，然后为每组和每组的每行分配空间。最后利用结构体赋值初始化cache。
- freecache：cache的析构函数。
- fetchoperation：读入op的函数。
- find_match_block：找到tag相同但是isvalid标识位表示这行为空的行。
- find_empty_block：找到空的行。
- find_recent_block：时间戳不是真正意义上的时间戳，是一个全局变量，每次load或者store的时候就加一。这是找到相对意义上最近的行，用于组冲突时的选择驱逐块。
- loadorstore：传入一个cache和一个operation，以及两个布尔值代表是否应该打印以及是否由主函数调用。首先调用getSet和getTag函数计算获得地址里的s和t。接下来在找到的s组中寻找t匹配的行，如果匹配上了就hitcount++，没匹配上就misscount++接着去找空的行，如果没有空，那就考虑驱逐最近的行的问题。在寻找匹配的过程中判断是否为主函数调用以及是否需要输出，最后使时间戳加一。

- modify：实现L(load)或者S(save)。
- get_arguments：将指令转为具体数值。
- 主函数：首先是指令处理。h参数是帮助信息选项，这里我没有具体实现。然后初始化参数，再调用get_arguments去读取参数。利用读取的参数初始化cache，获取文件指针，设置缓冲区，然后根据读取的指令进行加载或者存储操作。

partA实验结果

	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
TEST_CSIM_RESULTS=27			

根据实验文档，partA符合实验要求。

partB

32x32

实验中使用的cache参数是：(s=5, E=1, b=5)，类似于直接映射缓存。从参数可以得知，B等于32，意思是每一行有32个字节，一共有32个组，地址里后五位被解释为在行中的偏移，倒数第二个五位被解释为位索引。因为一个int型整数占4字节，所以一次load可以缓存进8个int类型的变量。如果A矩阵被放在地址为0的位置，则A[0] [0]地址是xxxx、00000、00000，被映射到s=0的组。以此类推，A[0] [8]被放在s=1的组，A[1] [0]在s=2...算下来A矩阵的每八个整数元素可以刚好被映射到00到31这32个块。由于内存的对齐，B矩阵的对应元素会被映射到完全对应的组中。

为了解决miss太多的问题，我采取了先把整个矩阵分成几大块，之后以“块”为单位进行转置的办法。将整个矩阵分为四大块，首先转置右上和左下，根据前文分析，它们对应的组完全没有重合，因此不会发生cachemiss。对于对角线上的矩阵块，采取先直接复制过去，再在B矩阵里转置的办法。

64x64

对64x64的数组进行分析时，采用和32x32一样的分块方法是不可行的，因为在按列访问B矩阵时会有前四行和后四行反复冲突的情况发生，于是考虑4x4分块。对于处于反对角线的两个8x8区域，先将右上角的前四行全部存入左下角的前四行。后逐步进行左下角后四行前四列的转置，即将A右上角后四行第一列的内容变为B左下角后四列第一行的内容，以此类推。随后完成左下角后四行后四列的转置。这样的思路也会导致前面优化前的对角线的问题，但是由于不做相关优化已经进入最优<1300范围，便不再进一步优化。

61x67

经过实验和资料查询，发现分块尺寸为17x4时miss最少。

partB实验结果

	Points	Max pts	Misses
Csim correctness	27.0	27	
Trans perf 32x32	8.0	8	259
Trans perf 64x64	8.0	8	1179
Trans perf 61x67	10.0	10	1848
Total points	53.0	53	

实验总结

总结和收获

1. 熟悉cache工作原理，理解通过代码优化减少cachemiss的产生。
2. 练习运用git进行工作。
3. 在实验中遇到了不熟悉main函数参数的问题，经过查阅资料已经解决。

结果总结

第一部分获得了27的评分，和题目所给的csim-ref达到一致。第二部分三个实例分别有259、1179、1848的miss，分别小于题目要求的最优miss300、1300、2000。在题目提供的评分文件里得到满分53分。