2021/1/7 过程

Part A

12/28

相关文件均被找到并被放在此处,服务器内基本配置也已经就绪

• make用以编译, make clean用以清除旧文件

阅读

- 使用valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -1来验证正确
- [./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace给予了参考的可执行二进制文件csim-ref
- ./csim-ref -h to see the help infomation
- trace文件格式 [操作] [地址] [大小]

i表示指令加载, I表示数据加载, s表示数据存储, m表示数据修改

I前面无空格, LSM前有空格

函数

参考网站

仅列部分

atoi(const char * nptr)

函数说明 atoi()会扫描参数nptr字符串,跳过前面的空格字符,直到遇上数字或正负符号才开始做转换, 转换为int

itoa

整数转换为字符串

double atof(const char *nptr)

字符串转换为double。atof()会扫描参数nptr字符串,跳过前面的空格字符,直到遇上数字或正负符号才开始做转换,而再遇到非数字或字符串结束时('\0')才结束转 换,并将结果返回。

long atol(const char *nptr)

将字符串转换成长整型数

参考网站

• int getopt(int argc,char * const argv[],const char * optstring); 头文件unistd.h, liunx c文件

• opstring是选择字符串

形如a:b:cd:e:,两个:表示参数可选,一个表示后面有一个参数,连续的字母表示这两个参数可以放在一起,比如-rf。

过程

• oprang时当前argc中的参数,比如-b 'sad', 程序运行到-b时,然后参数会被存储在oprang中。

过程

- 总共有如下参数, cache地址tag, set index, block。 然后被读取文件中的op, addr, opblock(没用)。
- cache的结构类似二次数组。
- 本题结构体cache中有三个参数, s, E, b。E是每个set的行的数量, b表示每个Block的大小, s表示set index bits, 与实际地址的关系如下。每个set含有一个line, 有valid(表示是否可用), tag(地址的高位,区别不同组映射), lru(处理Lru,最近最少使用算法)。
- 实际提供的参数位E, b, op, addr, opblocak; 要用到的数据为tag, set index, block
- 本处中间的存储在addr(long unsigned int)中, set index = (addr >> b) & (s 1); tag = addr >> (s+b);
 cacge.s = 2 << s。
- 当I为0时,直接跳过,如Ppt所言。
- 然后按照读取的op数据来调用函数

检查

- 运行./compare.sh
- 运行 /test-csim
- ./drive.py

Part B

1/02

相关文件均被找到并被放在此处,服务器内基本配置也已经就绪

• make用以编译, make clean用以清除旧文件

阅读

- 主要任务是编写一个转置矩阵的函数, 在trans.c中的transpose_submit函数。
- 对于int类型的局部变量有限制,一般不超过12个,不可以使用递归。
- 不能使用malloc函数

2021/1/7 过程

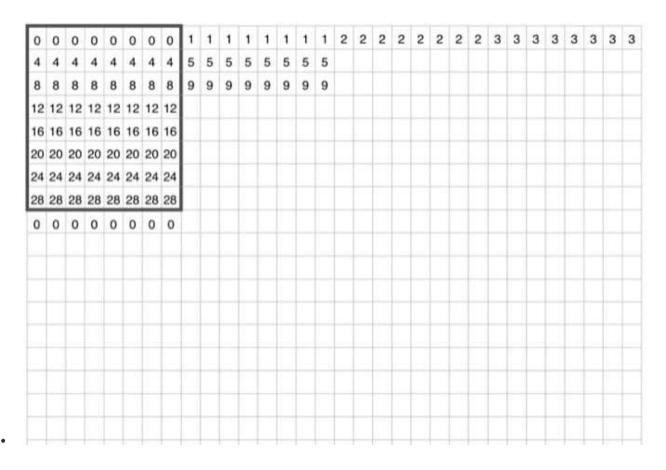
- 测试使用(s, E, b) = (5, 1, 5)的cache进行操作, 分别测试(N, M) = (32, 32), (64, 64), (61, 67)的矩阵。
- 对于测试结果Miss的要求可以参考Pdf
- 由Pdf中可以知道阻塞技术的pdf

过程

- 由阻塞技术的pdf中的代码可以知道实现的方法。
- blocking的大致概念为以数据块的形式读取数据,完全利用后丢弃,然后读取下一个,这样防止block利用的不全面, 所以需要不替换!就是在A读完写B的时候,让B不会覆盖掉A的block
- 由于我们的block能存8个int(32/4),所以blocking的数据块最好是以它为单位的,这样能尽可能利用block,例如8 * 8或者16 * 16。
- 在32*32的情况中,一行是32个int,也就是4个block,所以cache可以存8行,所以只要两个int之间相差8 行的整数倍,那么读取这两个元素所在的block就会发生替换。
- 此时一次读取的次数为AB两行,然后在cache交换之后再读取下一个
- 但是切分成8行后发许miss数为343,这时发现对角线的位置如果能先处理的话会减少Miss,因为i = j时命中的是同一个block,所以修改后成功。

因为Iru所以对角线处的A在cache里会被B替换

• cache中一个set有一行,一行一个block有8个int



• 综上,对于M的正方形矩阵,其分块大小为32*8/M。

- 后来发现M == 64时不太行, 最终Miss超过2000。
- 认为第三种情况下比第二种宽松得多,(矩阵大小类似)在之前的思想下采用暴力求解的方式算出来16符合要求。(2-32中2的倍数)。
- 由网上的思路可以知道,把A中前四个放在B之后,A最后4个先存放在B,减少miss。

A[0][0]		A[0][4]
A[0][1]		A[0][5]
A[0][2]		A[0][6]
A[0][3]	https://blog.csdn.net/weix	A[0][7]

MISS计算

• 32时A每行2次(对角线一次, 初次读入), B读取每行8次.

检查

- · ./compare.sh
- ./test-trans
- /driver.py