中国海洋大学 计算机科学与技术系

实验报告

姓名：陈岳阳 学号：21020007009　专业：计算机科学与技术

科目：计算机系统基础　题目：lab4

实验时间：2022/12/7

实验成绩：　实验教师:范浩

实验要求：

Cache Lab食用方法：

- partA要求写一个模拟cache的C文件csim.c，partB是对三种不同大小的矩阵，分别写矩阵转置函数
- 详情可以看课件pdf，要点挺多，集中在4和6的位置

实验内容：

```
polariscyy@ubuntu:~/Desktop/CSAPP/all/csapplab/cachelab/cachelab-handout$ ./driver.py
Part A: Testing cache simulator
Running ./test-csim
                        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
    27


Part B: Testing transpose function
Running ./test-trans -M 32 -N 32
Running ./test-trans -M 64 -N 64
Running ./test-trans -M 61 -N 67

Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           8.0         8        1171
Trans perf 61x67          10.0        10        1970
          Total points    53.0        53
polariscyy@ubuntu:~/Desktop/CSAPP/all/csapplab/cachelab/cachelab-handout$
```

partA

- 首先要知道m = address size = 64 bits，也就是16个16进制位。
- 命令行中会输入参数s，E，b，需要使用getopt来获得参数数值，然后通过t = m - (s + b)计算标记位tag长度
- 通过地址计算出定位cache的参数

```
void setCur(){
    cur_t = address >> (b + s);
    cur_s = (address>>b) & ((0xffffffff)>>(32-s));
}
```

- 如何模拟cache?
  - cache分为S个块，每个块有E个cache_line，每个cache_line有1个有效位，t个标记位，B个字节数据。本题中不考虑存储的数据，因此不需要B
  - 出现conflict时，使用LRU选择被evict的cache_line，因此每个cache_line中需要有一个记录使用的数据。考虑使用时间戳，每次访问某块时，将其中所有cache_line的时间戳+1，再将刚访问的cache_line时间戳置0
  - cache_line如下，cache为一个二维数组，init为数据初始化

```
typedef struct{
    int stamp; //used to record the recently visited cache
    bool is_valid; //valid bit
    int tag; //t
}cache_line;

cache_line** cache;

void init(){//init cache, S, t
    S = 1 << s;
    t = m - s - b;
    cache = (cache_line**)malloc(S * sizeof(cache_line*));
    for(i = 0; i < S; ++i)
        cache[i] = (cache_line*)malloc(E * sizeof(cache_line));
    for(i = 0; i < S; ++i){
        for(j = 0; j < E; ++j){
            cache[i][j].is_valid = false;
            cache[i][j].tag = -1;
            cache[i][j].stamp = 0;
        }
    }
}
```

- 访问某块时，首先将所有块的stamp加1，并记录下最大的stamp对应的下标its_index。之后先判断是否hit，若hit，修改其stamp为0。否则miss，判断是否存在空cache_line，若存在，加载置该cache_line。否则evict，该块已满，将its_index对应的cache_line替换。
- 代码如下(Load和Save在此题中没有区别)

```
void Load(){//Well... That's equivalent to Save
    int big_stamp = -1, its_index = -1;
    for(i = 0; i < E; ++i){
        if(cache[cur_s][i].stamp > big_stamp){
            big_stamp = cache[cur_s][i].stamp;
            its_index = i;
        }
        cache[cur_s][i].stamp++; //update
    }

    for(i = 0; i < E; ++i){
```

```
        if(cache[cur_s][i].is_valid && cache[cur_s][i].tag == cur_t){
            cache[cur_s][i].stamp = 0;
            hit++;
            return;
        }
    }

    miss++;
    if(cache[cur_s][its_index].is_valid)
        evict++;

    cache[cur_s][its_index].is_valid = true;
    cache[cur_s][its_index].stamp = 0;
    cache[cur_s][its_index].tag = cur_t;
}
```

- 完整代码如下

```
#include "cachelab.h"
#include <stdio.h>
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#define true 1
#define false 0
typedef int bool;

int address;
int size; //input
int i, j, k; //count control
int hit, miss, evict; //output
int s, E, b, t; //arguments
int m = 64; //size of address is 64-bits
int S; //2 ^ s
char opt; // return value of getopt
char op; // [operation] address,size

typedef struct{
    int stamp; //used to record the recently visited cache
    bool is_valid; //valid bit
    int tag; //t
}cache_line;

cache_line** cache;

void init(){//init cache, S, t, B
    S = 1 << s;
    t = m - s - b;
    cache = (cache_line**)malloc(S * sizeof(cache_line*));
    for(i = 0; i < S; ++i)
        cache[i] = (cache_line*)malloc(E * sizeof(cache_line));
    for(i = 0; i < S; ++i){
        for(j = 0; j < E; ++j){
            cache[i][j].is_valid = false;
```

```
            cache[i][j].tag = -1;
            cache[i][j].stamp = 0;
        }
    }
}

int cur_t, cur_s;

void setCur(){
    cur_t = address >> (b + s);
    cur_s = (address>>b) & ((0xffffffff)>>(32-s));
}

void Load(){//Well... That's equivalent to Save
    int big_stamp = -1, its_index = -1;
    for(i = 0; i < E; ++i){
        if(cache[cur_s][i].stamp > big_stamp){
            big_stamp = cache[cur_s][i].stamp;
            its_index = i;
        }
        cache[cur_s][i].stamp++; //update
    }

    for(i = 0; i < E; ++i){
        if(cache[cur_s][i].is_valid && cache[cur_s][i].tag == cur_t){
            cache[cur_s][i].stamp = 0;
            hit++;
            return;
        }
    }

    miss++;
    if(cache[cur_s][its_index].is_valid)
        evict++;

    cache[cur_s][its_index].is_valid = true;
    cache[cur_s][its_index].stamp = 0;
    cache[cur_s][its_index].tag = cur_t;
}

int main(int argc, char* argv[])
{
    char tracefile[100] = "\0";

    while((opt = getopt(argc, argv, "hvs:E:b:t:")) != -1){
        switch(opt){
            case 's':
                    s = atoi(optarg);
                    break;
            case 'E':
                    E = atoi(optarg);
                    break;
            case 'b':
                    b = atoi(optarg);
                    break;
            case 't':
                    strcpy(tracefile, optarg);
        }
```

```
    }

    init();
    FILE*fp = fopen(tracefile, "r");

    while(fscanf(fp, " %c %x,%d", &op, &address, &size) > 0)
    {
        printf(" %c %x,%d", op, address, size);
        setCur();
        switch(op)
        {
            case 'L':
                Load();
                break;
            case 'M':
                Load();
                hit++;
                break;
            case 'S':
                Load();
                break;
        }
        printf("\n");
    }
    for(i = 0; i < S; ++i)
        free(cache[i]);
    free(cache);
    printSummary(hit,miss,evict);
    return 0;
}
```

Part B

- 这个部分要求对给定大小的矩阵A进行转置，转置后的矩阵为B。要求使用的变量不超过12个（包含调用函数时的参数，准确来说是栈中的变量不超过12个），且miss次数在限制次数内。
- 很显然，直接进行转置是不行的。我们要进行分块，然后转置。题中cache的大小是1024bytes，每个block是32btytes。
- 32×32
    - 对于32×32矩阵，cache可以一次存下8行。进行8×8分块后，miss次数就已经达到要求。
    - 理论最小值是256。没有达到最小值的原因是主对角线块出现conflict，可能还有函数调用的消耗。
- 64×64矩阵
    - 对于64×64矩阵，cache只可以一次存下4行。尝试进行4×4分块，miss数为1677
    - 尝试先将8×4的数据放入矩阵B的相应位置，然后在B矩阵内部进行转置，减少在A矩阵中read时产生的miss数。最终结果为1171
    - 理论最小值是1024，和32×32一样，是由于主对角线块和函数调用时产生的miss。
- 主对角线优化
    - 我没有写，但是我觉得可以优化。
    - 由于主对角线上转置时，write和read的数据映射到同样的block，会造成很多conflict miss。
    - 可以将主对角线上的循环展开，减少miss数量，但可以预想到代码会非常verbose（-v），我不想写，所以我放弃了。
- 61×67矩阵

- 没有找到规律，对矩阵进行各种大小的分块。当17×17分块时，miss数为1970已经可以达到满分要求了。原理我也不明白。

实验总结

- 这次实验，我深入理解了cache的原理
  - 实现了一个简单的cache simulator，模拟了cache存取和LRU机制。
  - 实现了矩阵转置程序trans.c，深入理解了地址与block的映射和miss产生的情况
  - 深入了解了cache的原理
  - 增强了代码能力，学会了读取命令行和使用man查看函数攻略。