# cachelab_report



# 一、实验结果图

```
lalala@ubuntu:~/ICS/lab/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        1219
Trans perf 61x67          10.0        10        1950
          Total points    53.0        53
```

# 二、Part A

代码如下：

```
#include "cachelab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <getopt.h>
#include <unistd.h>

typedef struct cache_index//cache的索引结构
{
    unsigned long addr;
    int active, hittime;//有效位以及最近一次访问的时间
}ci_t;

unsigned long xatol(char* s)//从文件中读取的字符串中读取出地址的函数
{
```

```
    unsigned long num = 0, i = 3;
    unsigned char c;
    if(s[0] != ' ')
        i = 2;
    for(; (c = s[i]) != ','; i++)
    {
        if('0' <= c && c <= '9')
            num = num * 16 + c - '0';
        else
            num = num * 16 + c - 'a' + 10;
    }
    return num;
}


int detail = 0, s = 0, E = 0, b = 0;//detail对应‘-v’参数，默认为0

int main(int argc, char* argv[])
{
    FILE* file;
    opterr = 0;
    int ch;
    while((ch = getopt(argc, argv, "vs:E:b:t:")) != -1)//读取命令行命令
    {
        switch(ch)
        {
            case 'v':detail = 1;break;
            case 's':s = atoi(optarg);break;
            case 'E':E = atoi(optarg);break;
            case 'b':b = atoi(optarg);break;
            case 't':file = fopen(optarg,"r");break;
        }
    }

    int S = pow(2, s);//cache组数
    unsigned long smask = (S - 1) << b, tmask = 0xffffffff << (s + b); //地址掩码
    int miss = 0, hit = 0, eviction = 0, count = 0;//4种计数，每进行一次L或S操作count
加1
    ci_t* CI = (ci_t*)malloc(sizeof(ci_t) * S * E);//为cache索引分配地址
    memset(CI, 0, S * E * sizeof(ci_t));//初始化
    char content[20];//储存文件一行内容的字符串
    unsigned long address;//文件里一行中的地址
    int line = 0;//行数，E行为一组

    while(fgets(content, 20, file))
    {
        int match = 0, op_count = 1;//match记录是否hit，op_count是‘L’‘S’‘M’操作需要
访问几次cache
        count++;//操作次数+1
        address = xatol(content);//从文件中读取地址
        line = (address & smask) >> b;//得到组号
        int start = line * E, end = (line + 1) * E;//这一组的开始行和结束行
        if(content[0] != ' ')//跳过‘I’操作
            continue;
        else if(content[1] == 'M')//‘M’操作分两步进行
            op_count = 2;
        if(detail)//带参数'-v'时，输出文件中该行信息
        {
```

```
            char* find = strchr(content, '\n');//将换行符去掉
            *find = '\0';
            printf("%s",content + 1);
        }
        for(int j = 0; j < op_count; j++)//操作需要步数的循环
        {
            for(int i = start; i < end; i++)//判断是否hit的循环，将这组的E行全部遍历
            {
                if(CI[i].active)//有效位是否为1
                {
                    if((CI[i].addr & tmask) == (address & tmask))//判断标记码
                    {
                        hit++;//匹配hit+1
                        match = 1;//match设置为1
                        CI[i].hittime = count;//记录下这次的访问时间，count越大表示访问的距上一次访问越近
                        if(detail)
                            printf(" hit");//带‘-v’输出信息
                    }
                }
            }
            if(!match)//判断是否miss
            {
                miss++;
                if(detail)//带‘-v’输出信息
                    printf(" miss");
                int full = 1, i = start;//full标志是否该组被占满了
                for(; i < end; i++)
                {
                    if(!CI[i].active)//如果存在有效位为0，表示没有占满，不用替换
                    {
                        full = 0;
                        break;
                    }
                }
                if(!full)//如果没有占满，直接加载
                {
                    CI[i].active = 1;
                    CI[i].addr = address;
                    CI[i].hittime = count;
                }
                else//占满则执行替换
                {
                    eviction++;
                    if(detail)//带‘-v’输出信息
                        printf(" eviction");
                    int min = CI[start].hittime, min_line = start;//最久远一次访问的时间和行数
                    for(i = start + 1; i < end; i++)//寻找最久远一次替换
                    {
                        if(CI[i].hittime < min)
                        {
                            min = CI[i].hittime;
                            min_line = i;
                        }
                    }
                    CI[min_line].active = 1;//替换
                    CI[min_line].addr = address;
```

```
                    CI[min_line].hittime = count;
                }
            }
        }
        if(detail)////带‘-v’输出信息
            printf("\n");
    }

    free(CI);
    fclose(file);

    printSummary(hit, miss, eviction);
    return 0;
}
```

# 三、Part B

代码如下

```
void trans_32(int M, int N, int A[N][M], int B[M][N])//32*32矩阵的转置函数
{
    int i, j, k, m, same;//前4个是循环控制变量，same标志是否遇到对角线元素
    for(k = 0; k < 4; k++)//行成4部分
    {
        for(m = 0; m < 4; m++)//列分成4部分，共有16个8*8分块
        {
            for(i = 8 * m; i < 8 * (m + 1); i++)//横
            {
                same = 0;
                for(j = 8 * k; j < 8 * (k + 1); j++)//纵
                {
                    if(i == j)//如果是对角线元素，等到其他元素都转置后，再转置
                    {
                        same = 1;
                        continue;
                    }
                    B[j][i] = A[i][j];
                }
                if(same == 1)//对角线元素的处理
                    B[i][i] = A[i][i];
            }
        }
    }
}

void trans_64(int M, int N, int A[N][M], int B[M][N])//64*64矩阵转置函数
{
    int i, j, k, m;//循环控制变量
    int tmp0, tmp1, tmp2, tmp3, tmp4, tmp5, tmp6, tmp7;//临时变量
    for(m = 0; m < 8; m++)//列分8个部分
    {
        for (k = 0; k < 8; k++)//行分8个部分，共有64个8*8分块
        {
            j = 8 * k;//关于A的列坐标
            for (i = 8 * m; i < 8 * m + 4; i++)//处理8*8分块的上半部分4*8分块
            {
```

```
                tmp0 = A[i][j];
                tmp1 = A[i][j + 1];
                tmp2 = A[i][j + 2];
                tmp3 = A[i][j + 3];
                tmp4 = A[i][j + 4];
                tmp5 = A[i][j + 5];
                tmp6 = A[i][j + 6];
                tmp7 = A[i][j + 7];//读取每行的8个变量
                B[j][i] = tmp0;
                B[j + 1][i] = tmp1;
                B[j + 2][i] = tmp2;
                B[j + 3][i] = tmp3;//赋值给对应的转置的B
                B[j][i + 4] = tmp4;
                B[j + 1][i + 4] = tmp5;
                B[j + 2][i + 4] = tmp6;
                B[j + 3][i + 4] = tmp7;//由于每行的cache能存8个int，前4个位置是正确的转
置位置，后4个位置暂时空，因此将内容暂存到这里
            }
            for (j = 8 * k; j < 8 * k + 4; j++)//处理8*8分块的左下部分
            {
                tmp0 = B[j][i];
                tmp1 = B[j][i + 1];
                tmp2 = B[j][i + 2];
                tmp3 = B[j][i + 3];//读取B中每行暂存的不正确的转置内容
                tmp4 = A[i][j];
                tmp5 = A[i + 1][j];
                tmp6 = A[i + 2][j];
                tmp7 = A[i + 3][j];//从A读取应该存在相应位置的内容
                B[j][i] = tmp4;
                B[j][i + 1] = tmp5;
                B[j][i + 2] = tmp6;
                B[j][i + 3] = tmp7;//将该行内容替换为正确的
                B[j + 4][i - 4] = tmp0;
                B[j + 4][i - 3] = tmp1;
                B[j + 4][i - 2] = tmp2;
                B[j + 4][i - 1] = tmp3;//将暂存内容放置到其正确位置
            }
            for (j = 8 * k + 4; j < 8 * k + 8; j++)//作理最后的右下角元素
            {
                tmp0 = A[i][j];
                tmp1 = A[i + 1][j];
                tmp2 = A[i + 2][j];
                tmp3 = A[i + 3][j];//从A中取出最后4个int
                B[j][i] = tmp0;
                B[j][i + 1] = tmp1;
                B[j][i + 2] = tmp2;
                B[j][i + 3] = tmp3;//传递给B中对应的位置
            }
        }
    }
}

void trans_61_67(int M, int N, int A[N][M], int B[M][N])//61*67矩阵转置函数
{
    int i, j ,k, m;//循环控制变量
    for(k = 0; k < 61; k += 17)//每17或小于17列分为一部分
    {
        for(m = 0; m < 67; m += 17)//每17或小于17行分为一部分，分成17*17分块
```

```
        {
            for(i = k; i < k + 17 && i < 67; i++)//以下是每分块进行转置操作
            {
                for(j = m; j < m + 17 && j < 61; j++)
                {
                    B[j][i] = A[i][j];
                }
            }
        }
    }
}

char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    if(M == 32 && N == 32)
        trans_32(M, N, A, B);
    else if(M == 64 && N == 64)
        trans_64(M, N, A, B);
    else
        trans_61_67(M ,N, A, B);
}
```