《*Pthreads* 并行程序 **热的传导**》

--测评报告

工学院　洪瑶　1801111621

目录

一、并行算法描述

输入参数为：p（线程数量）,n（2^n 大小的行数数组）,m（2^m 列数数组），k（热源点的个数，以及其位置，生成一个三元数组时使用），eps（迭代最大误差限）；

热传导的数学模型为：

一个二维空间中，若某一点的温度始终保持不变，则将该点称为一个热源。初始时，除热源外，其它地方的温度均为 0。热量将按照一定的阻尼系数从热源向周围扩散。将该空间离散化为 2^n * 2^m 的栅格后，除热源所在的格点外，其它格点的温度随时间变化的规则如下，其中二维数组 A(0)表示各栅格点上的初始温度值，二维数组 A(T)表示 T 时刻各栅格点上的温度值，sum{b0, b1, b2, ... bk } = b0+b1+b2+...+bk：

1、A(t)[i j] =A(0)[i j]，若 A(0)[i j]>0

2、A(t)[i j] =A(t-1)[i j]*0.1+sum{A(t-1)[x y]: abs(x-i)=1 && abs(y-j)=1 }*0.225，若 A(0)[i j]=0、0<i<2^n-1、0<j<2^m-1

3、A(t)[i j] =A(t-1)[i j]*0.325+(A(t-1)[i j-1]+A(t-1)[i j+1]+A(t-1)[x j])*0.225，若 A(0)[i j]=0、i mod (2^n-1) ==0、0<j<2^m-1、abs(x-i)=1

4、A(t)[i j] =A(t-1)[i j]*0.325+(A(t-1)[i-1 j]+A(t-1)[i+1 j]+A(t-1)[i y])*0.225，若 A(0)[i j]=0、j mod (2^m-1) ==0、0<i<2^n-1、abs(y-j)=1

5、A(t)[0 0] =A(t-1)[0 0]*0.1+(A(t-1)[1 0]+A(t-1)[0 1])*0.45，若 A(0)[0 0]=0

6、A(t)[0 2^m-1] =A(t-1)[0 2^m-1]*0.1+(A(t-1)[1 2^m-1]+A(t-1)[0 2^m-2])*0.45，若 A(0)[0 2^m-1]=0

7、A(t)[2^n-1 0] =A(t-1)[2^n-1 0]*0.1+(A(t-1)[2^n-2 0]+A(t-1)[2^n-1 1])*0.45，若 A(0)[2^n-1 0]=0

8、A(t)[2^n-1 2^m-1] =A(t-1)[2^n-1 2^m-1]*0.1+(A(t-1)[2^n-2 2^m-1]+A(t-1)[2^n-1 2^m-2])*0.45，若 A(0)[2^n-1 2^m-1]=0

假设该空间共有 k 个热源，每个热源的温度可以不同。每个格点上的温度值用一个单精度浮点数表示。

设计的并行算法为：

Step1：在单个线程中，按热传导规模的行数进行划分，除了第一行和最后一行，对每个线程进行平均分配计算资源，其中第一行和最

后一行进行按列分配计算资源，而四个角点以及热源位置，分配给 0 号线程进行计算。

Step2：进行迭代计算，引入 bool 变量 FLAG，只要有一次某个线程中有误差大于 eps 的时候，就令全局变量 FLAG 为真，继续迭代，否则，计算结束。

Step3:按行分配资源进寻找局部最小值，最后将 p 个局部最小值求一次全局温度最小值。

二、改变线程数量和迭代空间规模，所编写并行程序的评测结果

（1）、控制 P=8，改变 N.

N=4，M=4:

```
***************value evaluation*****************
array-size = 2^4 * 2^4  eps=0.000995
serial impl.   : lowerest_temperature=30.5000000000000
reference impl.: lowerest_temperature=30.5000000000000  difference_to_ref
erence=0.0000000000000
your impl.     : lowerest_temperature=30.5000000000000  difference_to_ref
erence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=1.1432e-03
reference parallel impl.: time_cost=1.3231e-02 speedup=0.086
your impl.              : time_cost=2.2115e-02 speedup=0.052
```

N=6,M=6:

```
***************value evaluation*****************
array-size = 2^6 * 2^6  eps=0.001898
serial impl.   : lowerest_temperature=30.5000000000000
reference impl.: lowerest_temperature=30.5000000000000  difference_to_re
ference=0.0000000000000
your impl.     : lowerest_temperature=30.5000000000000  difference_to_ref
erence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=2.0275e-02
reference parallel impl.: time_cost=4.1102e-02 speedup=0.493
your impl.              : time_cost=6.3374e-02 speedup=0.320
```

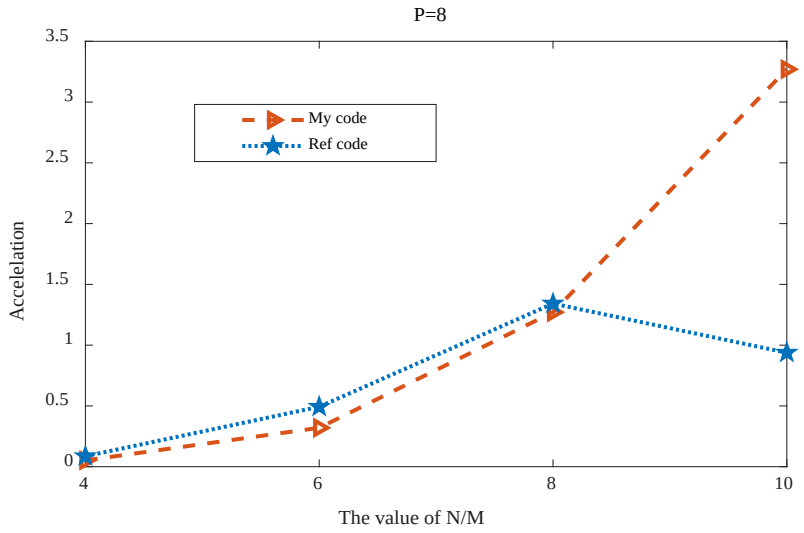
N=8,M=8:

```
***************value evaluation*****************
array-size = 2^8 * 2^8  eps=0.002754
serial impl.   : lowerest_temperature=30.5000000000000
reference impl.: lowerest_temperature=30.5000000000000  difference_to_re
ference=0.0000000000000
your impl.     : lowerest_temperature=30.5000000000000  difference_to_ref
erence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=8.1499e-01
reference parallel impl.: time_cost=6.0732e-01 speedup=1.342
your impl.              : time_cost=6.4117e-01 speedup=1.271
```

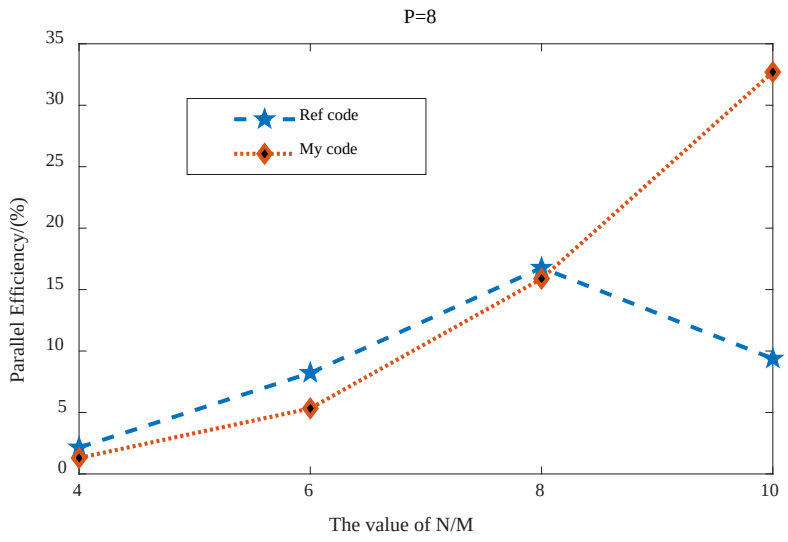
N=10, M=10:

```
array-size = 2^10 * 2^10  eps=0.002216
serial impl.   : lowerest_temperature=0.1089460104704
reference impl.: lowerest_temperature=0.1089460104704  difference_to_refe
rence=0.0000000000000
your impl.     : lowerest_temperature=0.1089460104704  difference_to_refe
rence=0.0000000000000
************performance evaluation*************
reference serial impl.  : time_cost=1.3740e+01
reference parallel impl.: time_cost=1.4645e+01 speedup=0.938
your impl.              : time_cost=4.2023e+00 speedup=3.270
```

加速比变化图：



并行效率变化图：



（2）、控制 N=10，M=10，改变 P

P=6：

```
***************value evaluation*****************
array-size = 2^10 * 2^10  eps=0.002176
serial impl.   : lowerest_temperature=0.1133995354176
reference impl.: lowerest_temperature=0.1133995354176  difference_to_refe
rence=0.0000000000000
your impl.     : lowerest_temperature=0.1133995354176  difference_to_refe
rence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=1.2373e+01
reference parallel impl.: time_cost=1.1400e+01 speedup=1.085
your impl.              : time_cost=3.9782e+00 speedup=3.110
```

P=8：

```
***************value evaluation*****************
array-size = 2^10 * 2^10  eps=0.002216
serial impl.   : lowerest_temperature=0.1089460104704
reference impl.: lowerest_temperature=0.1089460104704  difference_to_refe
rence=0.0000000000000
your impl.     : lowerest_temperature=0.1089460104704  difference_to_refer
ence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=1.3618e+01
reference parallel impl.: time_cost=1.4525e+01 speedup=0.938
your impl.              : time_cost=4.0465e+00 speedup=3.365
```

P=10：

```
***************value evaluation*****************
array-size = 2^10 * 2^10  eps=0.006515
serial impl.   : lowerest_temperature=0.0000067344563
reference impl.: lowerest_temperature=0.0000002280723  difference_to_refe
rence=0.0000065063841
your impl.     : lowerest_temperature=0.0000067344563  difference_to_refe
rence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=4.0088e+00
reference parallel impl.: time_cost=2.9307e+00 speedup=1.368
your impl.              : time_cost=1.3923e+00 speedup=2.879
```

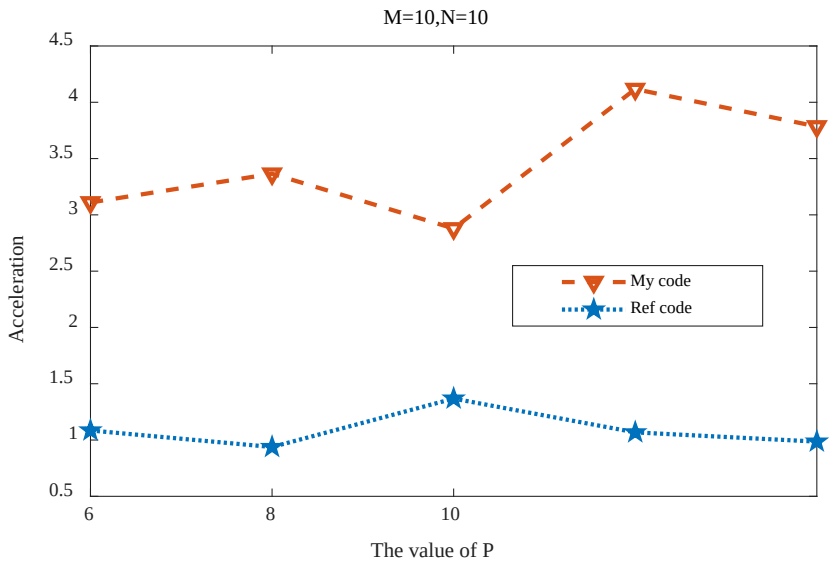
P=12：

```
***************value evaluation*****************
array-size = 2^10 * 2^10  eps=0.002451
serial impl.    : lowerest_temperature=0.0479986257851
reference impl.: lowerest_temperature=0.0479986257851  difference_to_refe
rence=0.0000000000000
your impl.      : lowerest_temperature=0.0481296852231  difference_to_refe
rence=0.0001310594380
************performance evaluation**************
reference serial impl.  : time_cost=1.1617e+01
reference parallel impl.: time_cost=1.0693e+01 speedup=1.086
your impl.              : time_cost=2.8216e+00 speedup=4.117
```

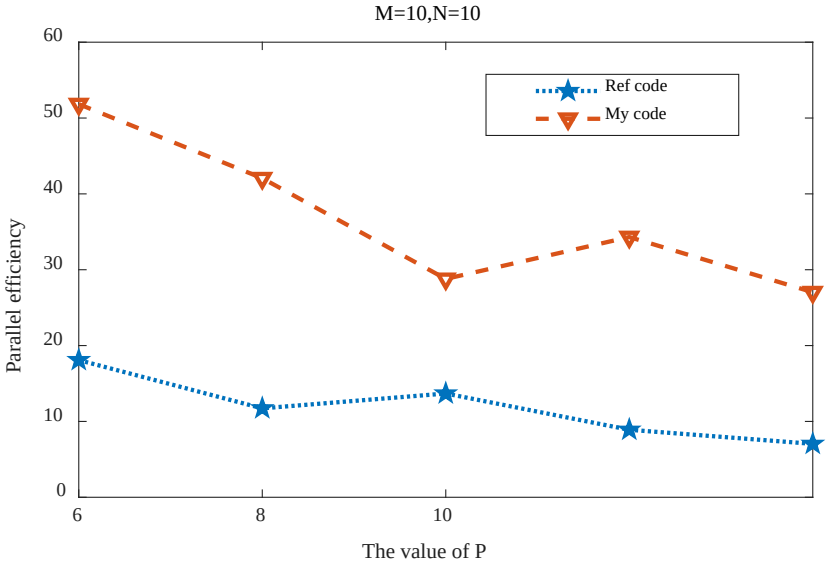
P=14：

```
***************value evaluation*****************
array-size = 2^10 * 2^10  eps=0.003047
serial impl.    : lowerest_temperature=0.0038444208913
reference impl.: lowerest_temperature=0.0038513254840  difference_to_ref
erence=0.0000069045927
your impl.      : lowerest_temperature=0.0038444208913  difference_to_refe
rence=0.0000000000000
************performance evaluation**************
reference serial impl.  : time_cost=6.4156e+00
reference parallel impl.: time_cost=6.5043e+00 speedup=0.986
your impl.              : time_cost=1.6948e+00 speedup=3.785
```

加速比变化图：



并行效率变化图：


M=10,N=10
Ref code
My code
Parallel efficiency
The value of P
60
50
40
30
20
10
0
6
8
10

三、评测结果分析

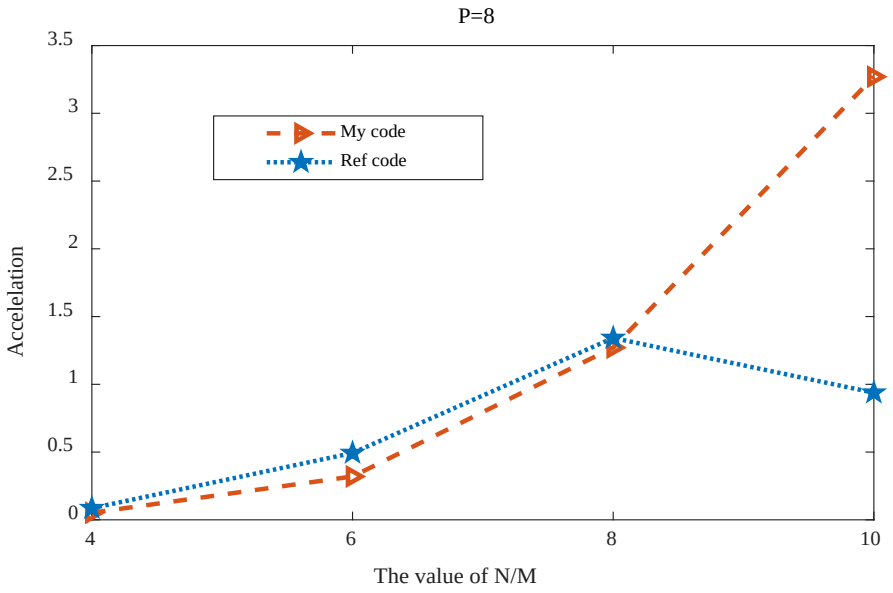


图 a 线程数量为 8，加速比随 M/N 变化

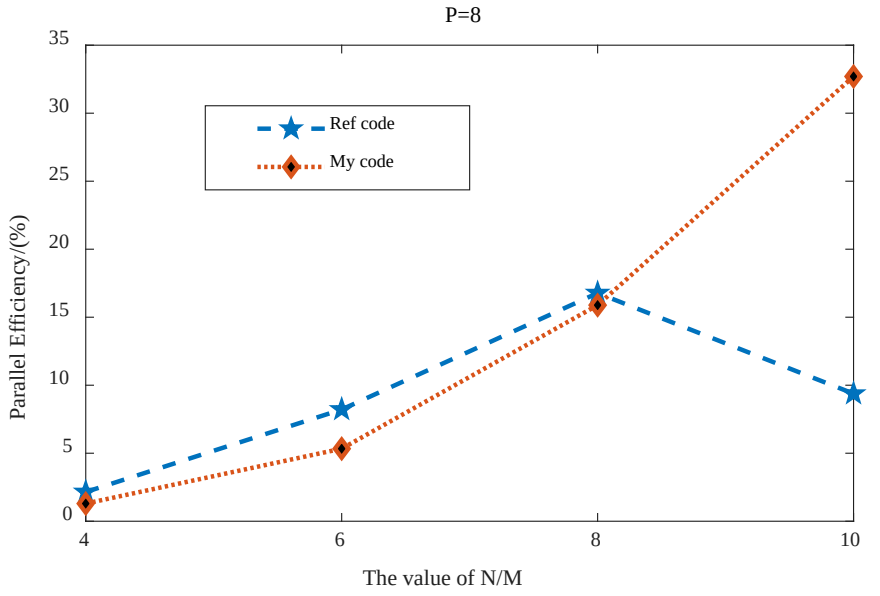


图 b 线程数量为 8，并行效率随 M/N 变化

图 a 为为 P=8 时加速比随着问题规模的变化图，在 N 比较小的时候，程序的并行加速比比较小，甚至小于 1，没有串行程序的计算速度快，随着 N 的增大，加速比大于 1，但是始终非常小，这是因为 P=8 时，计算规模较小时，并行的线程与线程间的通信时

间，等待时间，以及从全局变量存储的时间占比非常大，这导致计算加速比非常小。

在图 a/b 中，都能看出在计算规模达到 2^10 时，计算加速性得到体现，这说明在计算规模足够大时，在计算上花费的时间占比将会趋于越来越多。

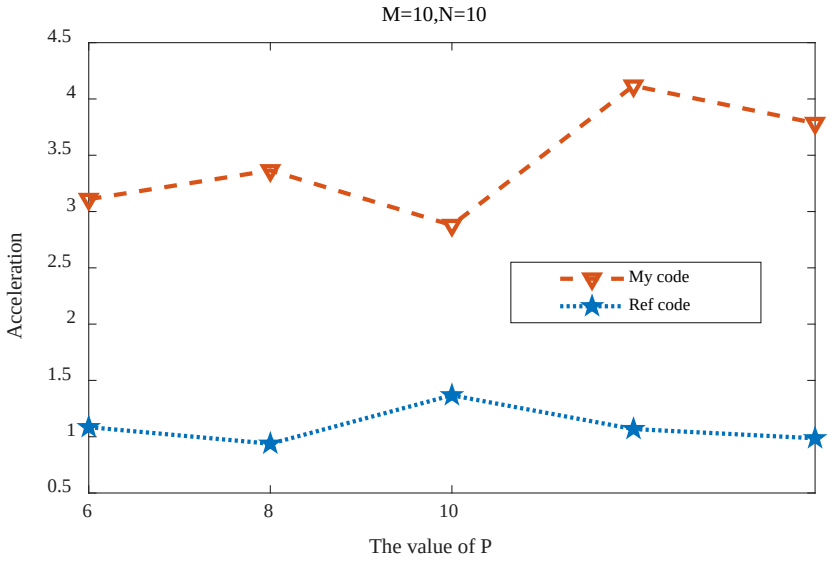


图 c M、N=10 时，加速比随线程数量变化图

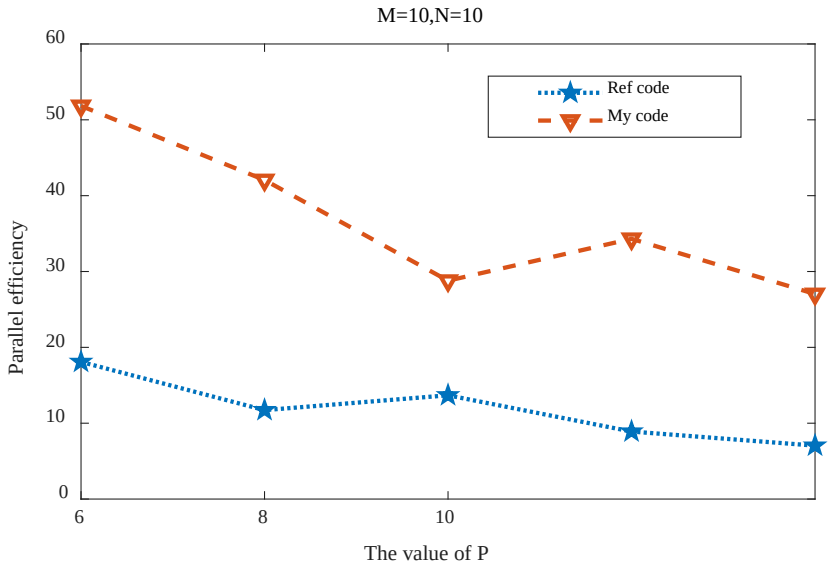


图 d M、N=10 时，并行效率随线程数量变化图

图 c 为控制 M,N 都为 10 的时候,加速比随着线程数量的变化情况，可以看到我的程序加速比是比参考程序大一些的，这是因为我的程序设计了较多变量，用于区别全局变量与线程变量，但是额外引入了新的变量，这导致在计算规模较小时，并行加速效果不如参考程序，但是在计算规模大的时候，加速效果明显优于参考程序。

图 c 显示加速比随着问题规模的变化趋势没有明显的线性关系，随着 P 的增加，加速比并没有稳固上升，这可能是因为线程的增加导致计算域减小，由于拥有者原则，其每次计算都需要其他区域的边缘数据，线程数量增加的时候，这部分边缘数据一致增加，一方面导致 R-R 冲突的增加，一方面违背了就近原则，这使得加速比没有随着线程数量的增加而增加。

图 d 从可以说明，并行效率并没有随着 P 的增加而呈现减小的趋势，这是由问题本身决定的，因此对于该问题，应当选择合适的 P 进行并行程序设计。

四、Pthread 程序

```
//
// Created by hongyao on 2018/11/1.
//
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include "fio.h"

//Global variable
```

```
int32_t *pinit;
int64_t n,m,size,k;
int thread_num,threadid;
pthread_barrier_t barrier;
float *pa, *pb ,*lowT;
float eps;
bool FLAG=true;
float loweresT = 0;
//*********************************//
//***************子****************//
//***************线****************//
//***************程****************//
//*********************************//
void  *worker(void *arg) {
    int64_t lb, ub, lc, uc, ld, ud, le, ue;
    int64_t pre, cur, nxt;
    int64_t i, j;
   // bool flag = true;
    float loweresT_p;
    int myID = __sync_fetch_and_add(&threadid, 1);
    int64_t loc_size = (n - 2) / thread_num;
    int64_t loc_size_lie = (m - 2) / thread_num;
    int64_t rest = (n - 2) % thread_num;
    int64_t rest_lie = (m - 2) % thread_num;
    int64_t loc_size_cpn = n / thread_num;
    int64_t loc_size_cpn_lie = m / thread_num;
    int64_t rest_cpn = (n) % thread_num;
    int64_t rest_cpn_lie = m % thread_num;
    //**********给线程分配计算资源*************
    if (myID < rest) {
        lb = loc_size * myID + myID;
        ub = lb + loc_size + 1;
    }
    else {
        lb = loc_size * myID + rest;
        ub = lb + loc_size;
    }

    if (myID < rest_lie) {
        lc = loc_size_lie * myID + myID;
        uc = lc + loc_size_lie + 1;
    }
    else {
    lc = loc_size_lie * myID + rest_lie;
```

```
    uc = lc + loc_size_lie;
    }

    if (myID < rest_cpn) {
        ld = loc_size_cpn * myID + myID;
        ud = ld + loc_size_cpn + 1;
    }
    else {
        ld = loc_size_cpn * myID + rest_cpn;
        ud = ld + loc_size_cpn;
    }

    if (myID < rest_cpn_lie) {
        le = loc_size_cpn_lie * myID + myID;
        ue = le + loc_size_cpn_lie + 1;
    }
    else {
        le = loc_size_cpn_lie * myID + rest_cpn_lie;
        ue = le + loc_size_cpn_lie;
    }
    //**********开始迭代**********
    while (FLAG) {
        nxt = m;
        for (j = lc + 1; j < uc + 1; j++) {
            pb[j] = 0.325 * pa[j] + 0.225 * (pa[nxt + j] + pa[j + 1]
+ pa[j - 1]);//第一行
        }
        cur = (lb + 1) * m;
        pre = lb * m;
        nxt = cur + m;
        for (i = lb + 1; i < ub + 1; i++) {
            pb[cur] = 0.325 * pa[cur] + 0.225 * (pa[pre] + pa[cur +
1] + pa[nxt]);
            for (j = 1; j < m - 1; j++)
                pb[cur + j] =
                        0.1 * pa[cur + j] + 0.225 * (pa[pre + j] +
pa[nxt + j] + pa[cur + j + 1] + pa[cur + j - 1]);
            pb[cur + j] = 0.325 * pa[cur + j] + 0.225 * (pa[pre + j]
+ pa[cur + j - 1] + pa[nxt + j]);
            pre = cur;
            cur = nxt;
            nxt += m;
        }
        cur = (n-1)*m;
```

```
        pre = cur-m;
        for (j = lc + 1; j < uc + 1; j++) {
            pb[cur + j] = 0.325 * pa[cur + j] + 0.225 * (pa[pre + j]
+ pa[cur + j + 1] + pa[cur + j - 1]);//最后一行
        }
        pthread_barrier_wait(&barrier);
        if (myID==0) {
            pb[0] = 0.1 * pa[0] + 0.45 * (pa[1] + pa[m]);//四个角点
            pb[m - 1] = 0.1 * pa[m - 1] + 0.45 * (pa[m - 2] + pa[m +
m - 1]);
            cur = (n - 1) * m;
            pb[cur] = 0.1 * pa[cur] + 0.45 * (pa[cur + 1] + pa[cur -
m]);
            pb[cur + m - 1] = 0.1 * pa[cur + m - 1] + 0.45 * (pa[cur
+ m - 2] + pa[cur - 1]);
            for (i = 0; i < k; i++)     {
                pb[pinit[i * 3] * m + pinit[i * 3 + 1]] = pa[pinit[i
* 3] * m + pinit[i * 3 + 1]];
            }                           //定热源
            FLAG = false;
        }
        pthread_barrier_wait(&barrier);
        //**********判断是否继续迭代*************
        cur = ld * m;
        for (i = ld; i < ud; i++) {
            for (j = 0; j < m; j++) {
                if (fabs(pb[cur + j] - pa[cur + j]) >= eps) {
                    FLAG = true;
                    break;
                }
            }
            if (FLAG) {
                break;
            }
            cur += m;
        }
       pthread_barrier_wait(&barrier);
        if (myID==0) {
          float *temp = pa;
             pa = pb;
             pb = temp;
        }
        pthread_barrier_wait(&barrier);
    }
```

```
    //**********开始寻找局部最低温**********
    cur = ld * m;
    loweresT_p = loweresT;
    for (i = ld; i < ud; i++) {
        for (j = 0; j < m; j++)
            if (loweresT_p > pa[cur + j]) {
                loweresT_p = pa[cur + j];
            }
        cur += m;
    }
    lowT[myID] = loweresT_p;
    return (void *) 0;
    }

//*********************************//
//**************主****************//
//**************线****************//
//**************程****************//
//*********************************//
int main(int argc, char** argv ) {
    int64_t i, j;
    thread_num = atoi(argv[1]); //p
    n = atoll(argv[2]);     //n
    m = atoll(argv[3]);     //m
    k = atoi(argv[4]);      //k
    eps = atof(argv[5]);      //eps

    size = ((int64_t) 1 << (n + m));
    pa = (float *) malloc(size * sizeof(float));
    pb = (float *) malloc(size * sizeof(float));
    lowT = (float *) malloc(thread_num * sizeof(float));
    n = 1L << n;
    m = 1L << m;
    pinit = (int32_t *) malloc(3 * k * sizeof(int32_t));  //热源位置
保存
    pthread_barrier_init(&barrier, NULL, thread_num);    //初始化栅樟
    threadid = 0;

    input_data(pinit, 3 * k * sizeof(int32_t));
    for (int i = 0; i < k; i++) {
        float *fp = (float *) &pinit[i * 3 + 2];
        pa[pinit[i * 3] * m + pinit[i * 3 + 1]] = *fp;
        if (loweresT < *fp) loweresT = *fp;
    }
```

```
    pthread_t *threads = new pthread_t[thread_num];
    for (int i = 0; i < thread_num; i++)
pthread_create(&(threads[i]), NULL, worker, &thread_num);
    for (int i = 0; i < thread_num; i++) pthread_join(threads[i],
NULL);
    for (j = 0; j < thread_num; j++)
    for (j = 0; j < thread_num; j++)
        if (loweresT > lowT[j]) {
            loweresT = lowT[j];
        }                          //找极值*/
    output_data(&loweresT, sizeof(float));
    pthread_barrier_destroy(&barrier);      //删除栅樟*/
    free(pa);
    free(pb);
    free(pinit);
    return EXIT_SUCCESS;
```