《*Pthreads* 并行程序 **2D5P**》

--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）;

数组拼接的数学模型为：

通过 2D5P 模板计算，将二维单精度浮点数型数组 A 变换为数组 B，数组 A 的规模为 2^n * 2^m。n 和 m 均不超过 15。采用的模板计算规则如下：

1、B[i j] = A[i j]，若 i=0、或者 i=2^n-1、或者 j=0、或者 j=2^m-1。

2、B[i j] = (A[i j]+A[i-1 j]+A[i j-1]+A[i+1 j]+A[i j+1])/5，若 0<I<2^n-1、且 0<j<2^m-1。

设计的并行算法为：

Step1： 输入 n，m,令 n=2^n,m=2^m,利用互斥锁,令某一个线程将第一行与最后一行的 pa 元素全部拷贝进 pb 中，剩余 n-2 行的元素需要进行模板计算。

Step2： 由于 P 远小于 2^n,所以分配每个线程(n-2)/p 行的 pa 进行 2D5P 扫描模板计算。

Step3: 扫描变换过程中，若为首个数字或者行中最后一个数组，那么直接将 pa 赋值给 pb,若不是，则进行 2D5P 变换再给 pb。

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=3，M=3:

```
***************value evaluation*****************
array-size = 2^3 * 2^3
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=3.7442e-04
reference parallel impl.: time_cost=3.0081e-04 speedup=1.245
your impl.              : time_cost=3.4598e-04 speedup=1.082
```

N=6,M=6:

```
***************value evaluation*****************
array-size = 2^6 * 2^6
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=4.7160e-04
reference parallel impl.: time_cost=3.5028e-04 speedup=1.346
your impl.              : time_cost=3.5791e-04 speedup=1.318
```

N=9,M=9:

```
***************value evaluation*****************
array-size = 2^9 * 2^9
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=4.8360e-03
reference parallel impl.: time_cost=1.8279e-03 speedup=2.646
your impl.              : time_cost=1.6167e-03 speedup=2.991
```

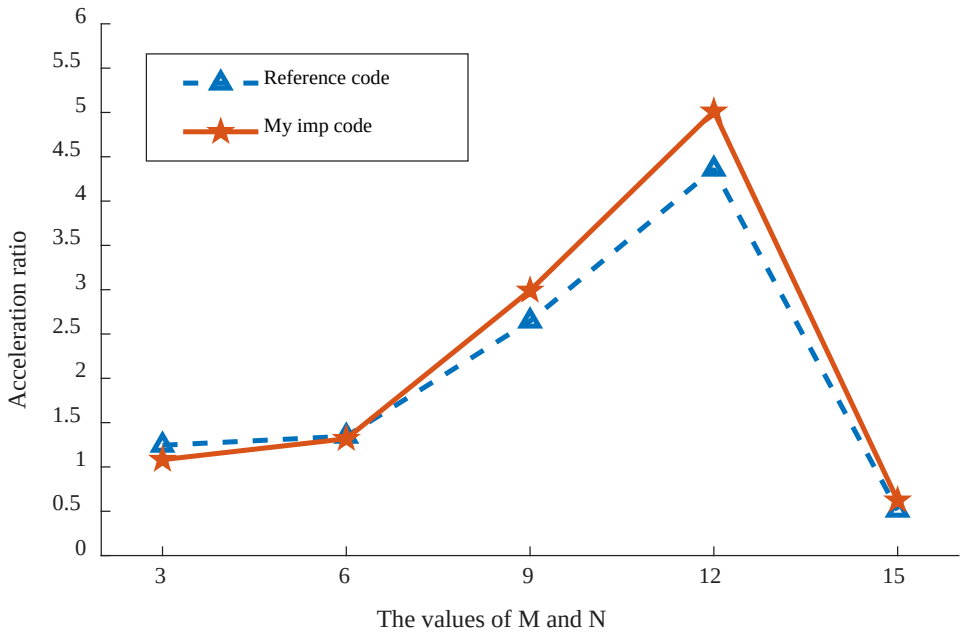
N=12，M=12:

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4831e-01
reference parallel impl.: time_cost=5.6952e-02 speedup=4.360
your impl.              : time_cost=4.9541e-02 speedup=5.012
```

N=15, M=15:

```
***************value evaluation*****************
array-size = 2^15 * 2^15
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=9.4164e+01
reference parallel impl.: time_cost=1.8385e+02 speedup=0.512
your impl.              : time_cost=1.5107e+02 speedup=0.623
```

加速比变化图：



（2）、控制 N=12，M=12，改变 P

P=2：

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4803e-01
reference parallel impl.: time_cost=6.3661e-02 speedup=3.896
your impl.              : time_cost=5.5525e-02 speedup=4.467
```

P=4：

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.5688e-01
reference parallel impl.: time_cost=5.5239e-02 speedup=4.650
your impl.              : time_cost=4.8477e-02 speedup=5.299
```

P=6：

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4768e-01
reference parallel impl.: time_cost=5.7405e-02 speedup=4.315
your impl.              : time_cost=5.3555e-02 speedup=4.625
```

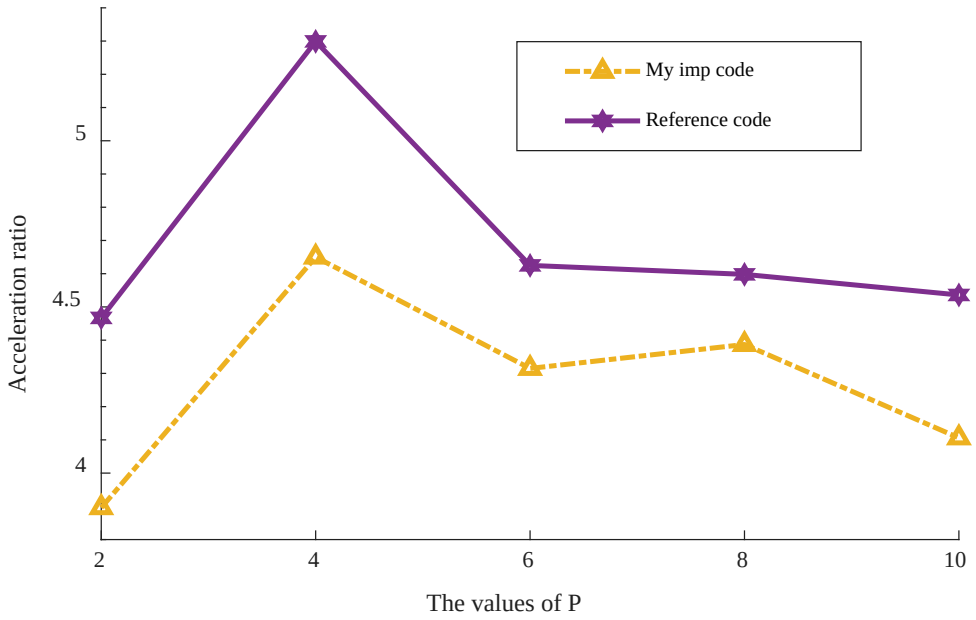
P=8：

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4595e-01
reference parallel impl.: time_cost=5.6059e-02 speedup=4.387
your impl.              : time_cost=5.3485e-02 speedup=4.598
```

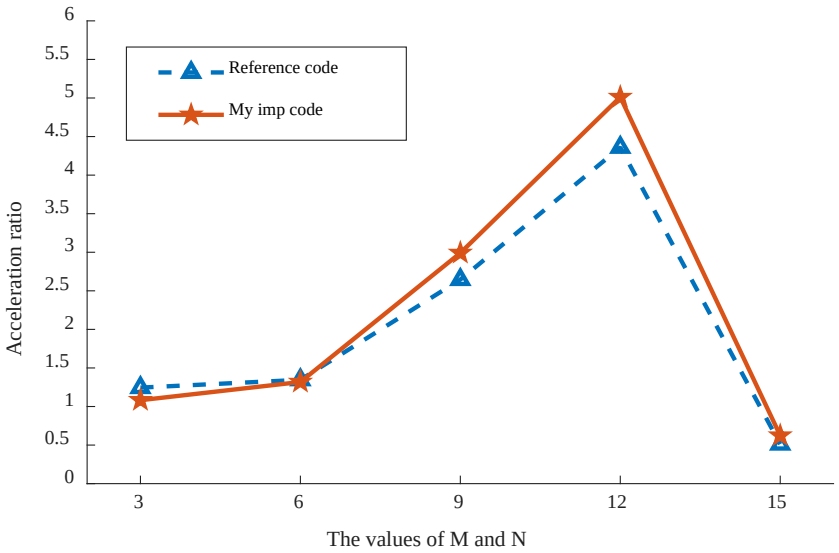
P=10：

```
***************value evaluation*****************
array-size = 2^12 * 2^12
congratulations, result of your impl. is correct
************performance evaluation**************
reference serial impl.  : time_cost=2.4581e-01
reference parallel impl.: time_cost=5.9870e-02 speedup=4.106
your impl.              : time_cost=5.4196e-02 speedup=4.536
```

加速比变化图：



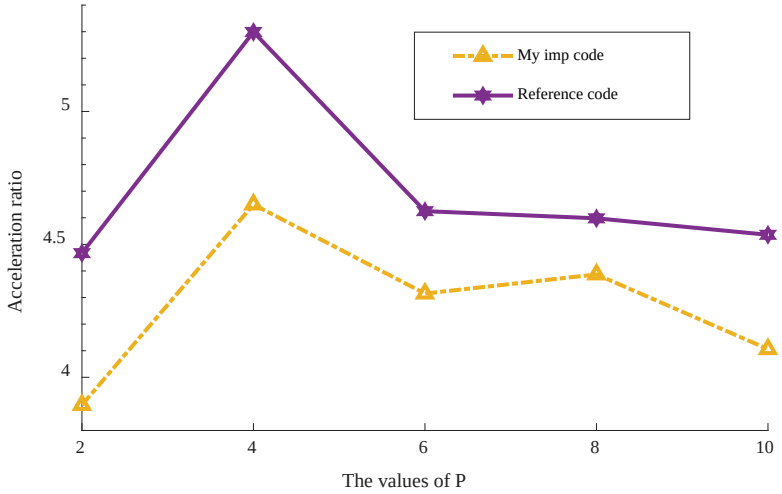
三、评测结果分析



（1）上图为 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，随着 N 的增大，加速比也越来越大，但是在 N=15 时，加速比变得异常小，这可能是因为计算规模过大，导致系统资源难以连续导致的。

（2）在 N 比较小的时候，我的程序的加速比比参考程序的加速比小，而 N 比较大的时候，加速比超过参考程序的，这是因为的我的程序将第一行第一列使用复制函数在第一个线程中独立运行了，而参考程序将这部分平摊给程序进行计算了，这使得我的程序进行判断的成分减少，因此在 N 比较大的时候加速比超过参考程序。

（3）由于 P 固定，故除了 N=15 外，并行效率一致在增加，这说明并行程序伸缩性良好。



（1）上图为控制 M，N 都为 12 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，这是因为我的程序使用 BARRIER 将第一行和最后一行的数据进行直接复制，没有进行判断，这使得我的程序加速比比参考程序的大。

（2）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，加速比在大雨 4 的时候反而减小，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，其每次模板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。

（3）从斜率可以看出，并行效率并没有随着 P 的增加而增加，这说明，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
int64_t n,m,size,size_center;
int thread_num,threadid;
pthread_barrier_t barrier;
float *pa, *pb;

//*********************************//
//**************子*****************//
//**************线*****************//
//**************程*****************//
//*********************************//
void  *worker(void *arg) {
    int64_t i,j,lb,ub;
    int myID = __sync_fetch_and_add(&threadid,1);
    int64_t loc_size = (n-2)/thread_num;
    int64_t rest = (n-2)%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 (pthread_barrier_wait(&barrier) ==
PTHREAD_BARRIER_SERIAL_THREAD) {
        memcpy(pb, pa, sizeof(float) * m);
        memcpy(&pb[m*(n-1)], &pa[m*(n-1)], sizeof(float) * m);
    }

    for (i=lb+1;i<ub+1;i++){
```

```
        pb[m*i]=pa[m*i];
        pb[m*(i+1)-1]=pa[m*(i+1)-1];
        for (j=i*m+1;j<i*m+m-1;j++){
            pb[j] = (pa[j] + pa[j - 1] + pa[j + 1] + pa[j-m]
+pa[j+m]) / 5.0;
        }
    }
}

//*********************************//
//**************主****************//
//**************线****************//
//**************程****************//
//*********************************//
int main(int argc, char** argv ) {
    thread_num = atoi(argv[1]);
    n = atoll(argv[2]);
    m = atoll(argv[3]);
    size = ((int64_t) 1 << (n + m));
    size_center = size-2*m-2*n+4;
    threadid=0;
    n = 1L<<n;
    m = 1L<<m;
    if (posix_memalign((void **) &pa, getpagesize(), size *
sizeof(float))) {
        perror("posix_memalign");
        return EXIT_SUCCESS;
    }
    if (posix_memalign((void **) &pb, getpagesize(), size *
sizeof(float))) {
        perror("posix_memalign");
        return EXIT_SUCCESS;
    }
    pthread_barrier_init(&barrier,NULL,thread_num);    //设置栅障
    input_data(pa, size * sizeof(float));
    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);
    output_data(pb, size*sizeof(float));
    pthread_barrier_destroy(&barrier);      //删除栅障*/
    free(pa);
    free(pb);
```

```
    return EXIT_SUCCESS;
}
```