《*Pthreads* 并行程序 **N-BODY**》

--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）；

N-BODY 的数学模型为：

在一个三维空间，有 N 个粒子。已知各粒子的质量、初始位置，按下列规则模拟这些粒子的运动过程。

- 在每一时刻，任意两个粒子 i 和粒子 j 间的作用力$\overrightarrow{f(T)_{\iota,J}} = G\frac{m_i \times m_j}{\left|\overrightarrow{P_J}^T - \overrightarrow{P_\iota}^T\right|^3} \times (\overrightarrow{P_J}^T - \overrightarrow{P_\iota}^T)$
- m_i和m_j分别为粒子 i 和粒子 j 的质量
- $\left|\overrightarrow{P_J}^T - \overrightarrow{P_\iota}^T\right|$表示 T 时刻第 i 颗粒子和第 j 颗粒子的绝对距离
- $\overrightarrow{P_J}^T - \overrightarrow{P_\iota}^T$表示 T 时刻第 i 颗粒子和第 j 颗粒子的矢量距离
- G 是一个常量
- 假设一个粒子的质量为 m，它在 t 时刻的位置为(x(t) y(t) z(t))，受到其他粒子的作用力为$\vec{F}$，则它在 t+1 时刻的位置(x(t+1) y(t+1) z(t+1)) = (x(t) y(t) z(t)) + $\vec{F}$/m

令$\overrightarrow{F(\iota,T)}$表示第 i#粒子在 T 时刻的受力，时间步 T 上的计算可以划分为三部分，

- 计算粒子对之间的作用力，共 n(n-1)/2 个并行的任务，tpair(i j)负责计算$\overrightarrow{f(T)_{\iota,J}}$
- 计算每个粒子的受力，共 n(n-1)个并发的任务 tf(i　j)：

  若i<j：$\overrightarrow{F(\iota,T)}$+=$\overrightarrow{f(T)_{\iota,J}}$

  若 i> j：　$\overrightarrow{F(J,T)}$ −=$\overrightarrow{f(T)_{\iota,J}}$

- 共 n 个并行的任务，tu(i)执行(x_i(T)　y_i(T)　z_i(T))+=$\overrightarrow{F(\iota,T)}/m_i$

设计的并行算法为：

假设有 np 颗处理器，将粒子划分成 np 份，每一份 size 个粒子

- 任务 W(x y)

  执行 tpair(i j)计算粒子 i 和粒子 j 之间的作用力$\overrightarrow{f(T)_{i,j}}$

  x×size≤i<(x+1)×size, y×size≤j<(y+1)×size

  $\overrightarrow{F(i,T)}$+=$\overrightarrow{f(T)_{i,j}}$、$\overrightarrow{lF(j,T)}$-=$\overrightarrow{f(T)_{i,j}}$

  当前任务的所有粒子对计算后，再执行$\overrightarrow{F(j,T)}$+=$\overrightarrow{lF(j,T)}$

  y×size≤j<(y+1)×size

- 时间步 T 上的算法

  在第 k#处理器上，执行任务 W(k k)

  循环执行 np/2 个超级计算步，在第 s 个超级计算步，第 k#处理器执行任务 W(x y)

  若 s+k<np：x=k, y=s+k

  若 s+k≥np：x=s+k-np, y=k

处理器 p 上执行的任务，其中 k 是处理器的数量

- 执行的任务$W_{i,j}$

  $W_{p,p}$

  $W_{p,j}$ ： j-p≤[(k+1)/2]

  $W_{i,p}$ ： p-i＞[(k+1)/2]

- 计算子集 p 中各 粒子 i 在新时刻T′的$\overrightarrow{a(T')_i}$、$\overrightarrow{P(T')_i}$、$\overrightarrow{v(T')_i}$

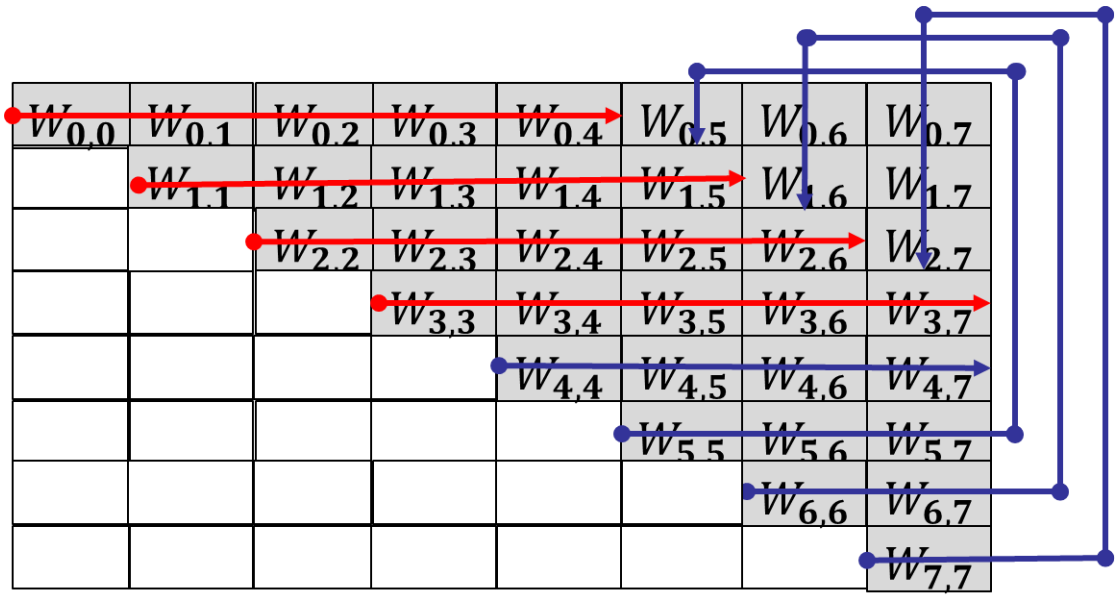


Figure1. 流水并行示意图

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

在 **easyHPC** 评测平台进行评测：

（1）、控制 P=4，改变 N.

N=5，T=3：

```
***************value evaluation*****************
array-size = 2^5 and steps = 2^3
reference impl.: difference_1=1.793603e-14 difference_2=1.847411e-13
reference impl.: difference_1=1.793603e-14 difference_2=1.847411e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k])         if         fabs((rValue[k]-
value[k])/rValue[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by
reference impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=4.1086e-02
reference parallel impl.: time_cost=2.0076e-03 speedup=20.465
your impl.              : time_cost=1.9699e-03 speedup=20.857
```

N=6, T=3：

```
***************value evaluation*****************
array-size = 2^6 and steps = 2^3
reference impl.: difference_1=1.517664e-14 difference_2=1.563194e-13
reference impl.: difference_1=1.517664e-14 difference_2=1.563194e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=4.3104e-02
reference parallel impl.: time_cost=2.4519e-03 speedup=17.580
your impl.              : time_cost=2.4379e-03 speedup=17.681
```

N=7, T=3:

```
***************value evaluation*****************
array-size = 2^7 and steps = 2^3
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=2.7694e-02
reference parallel impl.: time_cost=2.2572e-03 speedup=12.269
your impl.              : time_cost=3.1019e-03 speedup=8.928
```

N=8, T=3:

```
***************value evaluation*****************
array-size = 2^8 and steps = 2^3
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=2.7678e-01
reference parallel impl.: time_cost=3.4986e-03 speedup=79.112
your impl.              : time_cost=6.0473e-03 speedup=45.770
```

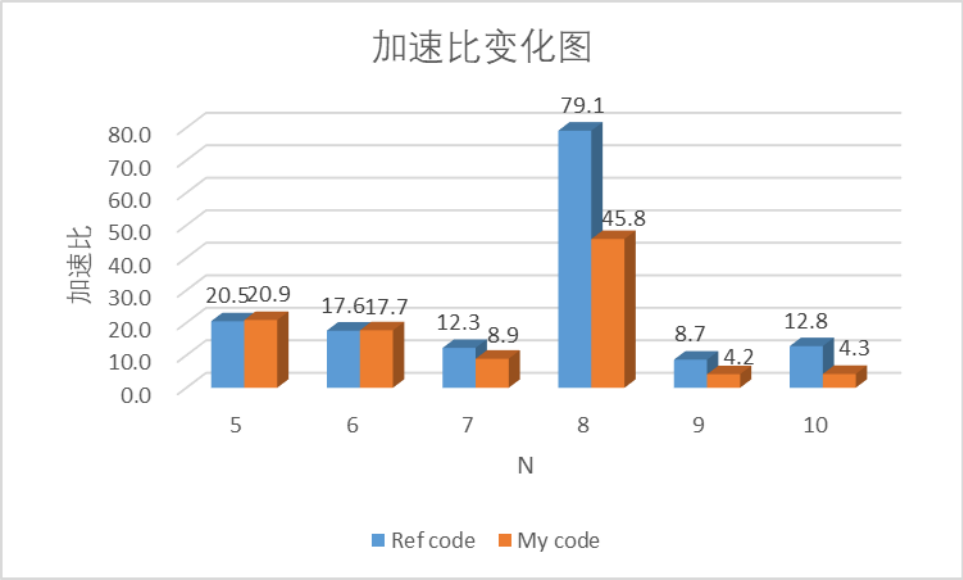
N=9, T=3:

```
***************value evaluation*****************
array-size = 2^9 and steps = 2^3
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=5.3469e-02
reference parallel impl.: time_cost=6.1818e-03 speedup=8.650
your impl.              : time_cost=1.2827e-02 speedup=4.169
```

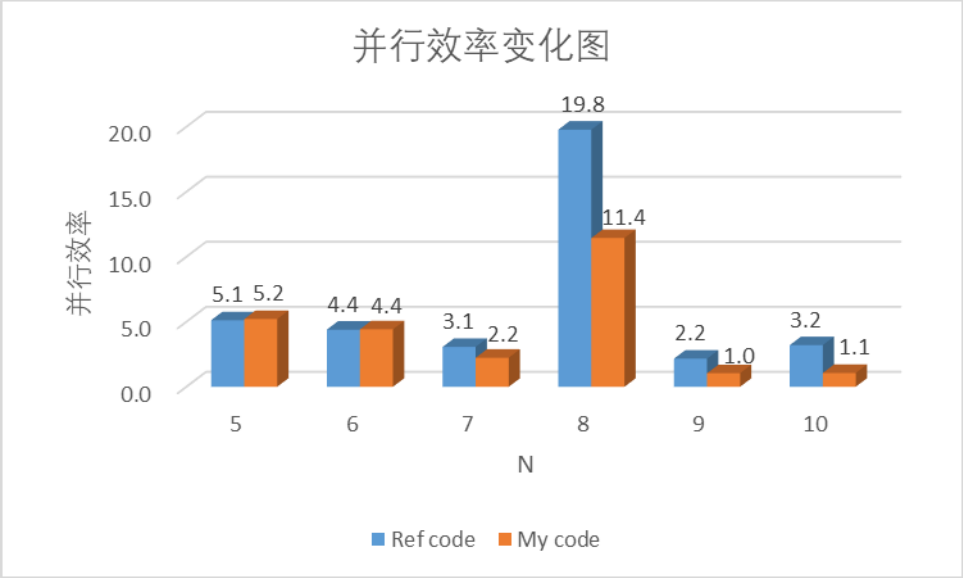
N=10, M=10:

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=2.2744e-01
reference parallel impl.: time_cost=1.7740e-02 speedup=12.821
your impl.              : time_cost=5.3475e-02 speedup=4.253
```

加速比变化图（P=4）：



并行效率变化图：



（2）、控制 N=10，T=3，改变 P

P=4：

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
reference impl.: difference_1=1.241725e-14 difference_2=1.278977e-13
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValue
[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=2.2821e-01
reference parallel impl.: time_cost=1.7406e-02 speedup=13.111
your impl.              : time_cost=4.1216e-02 speedup=5.537
```

P=16：

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=4.139084e-15 difference_2=4.263256e-14
reference impl.: difference_1=4.139084e-15 difference_2=4.263256e-14
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValue
[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=2.9228e-01
reference parallel impl.: time_cost=1.6439e-02 speedup=17.779
your impl.              : time_cost=2.0172e-02 speedup=14.490
```

P=64:

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=1.379695e-15 difference_2=1.421085e-14
reference impl.: difference_1=1.379695e-15 difference_2=1.421085e-14
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValue
[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=1.1337e-01
reference parallel impl.: time_cost=1.4472e-01 speedup=0.783
your impl.              : time_cost=2.1607e-02 speedup=5.247
```

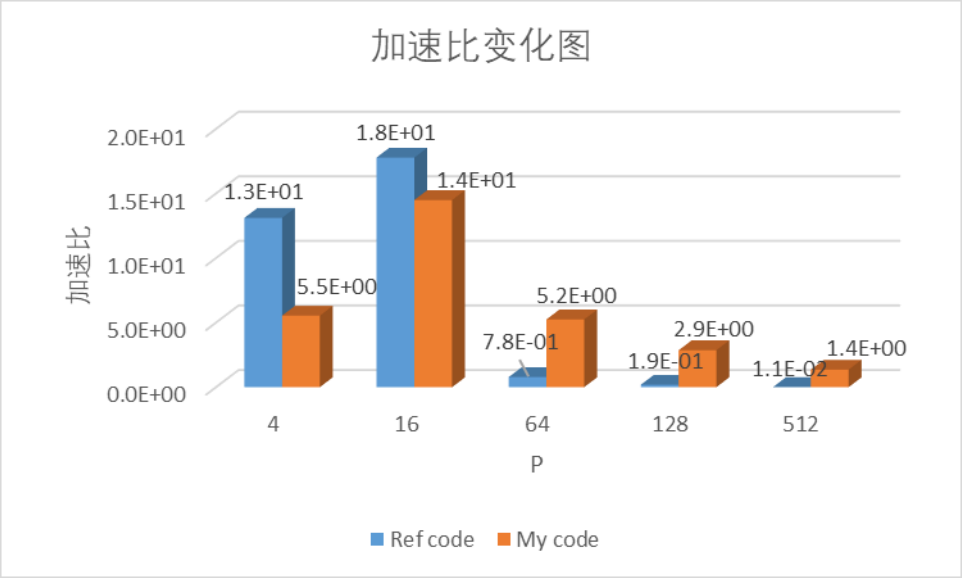
P=128:

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=1.379695e-15 difference_2=1.421085e-14
reference impl.: difference_1=1.379695e-15 difference_2=1.421085e-14
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValue
[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation**************
reference serial impl.  : time_cost=9.8824e-02
reference parallel impl.: time_cost=5.2499e-01 speedup=0.188
your impl.              : time_cost=3.4529e-02 speedup=2.862
```

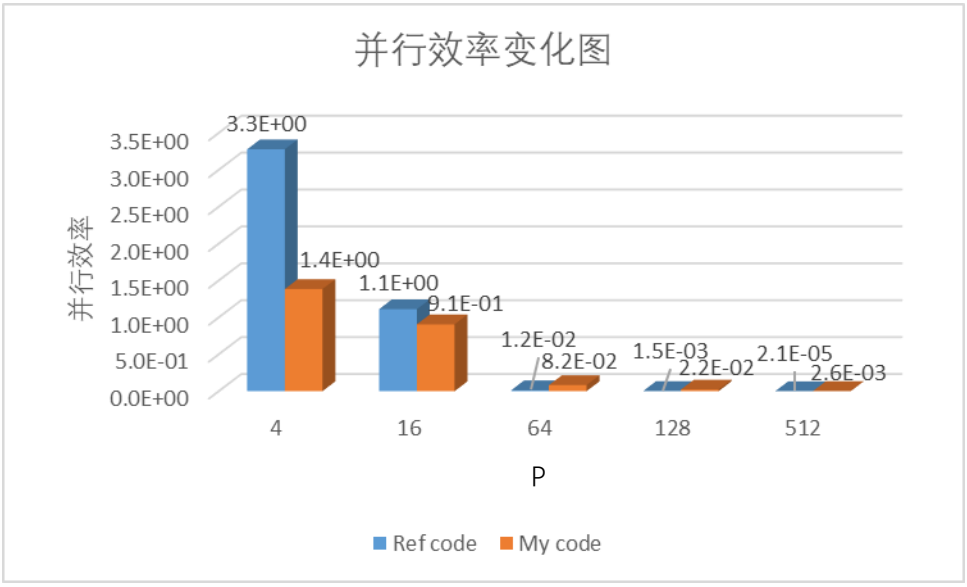
P=512：

```
***************value evaluation*****************
array-size = 2^10 and steps = 2^3
reference impl.: difference_1=0.000000e+00 difference_2=0.000000e+00
reference impl.: difference_1=0.000000e+00 difference_2=0.000000e+00
your impl.     : difference_1=0.000000e+00 difference_2=0.000000e+00

difference_1=max{fabs((rValue[i]-value[i])/rValue[i])}
difference_2=fabs(rValue[k]-value[k]) if fabs((rValue[k]-value[k])/rValu
e[i])=max{fabs((rValue[i]-value[i])/rValue[i])}
rValue----value resulted by serial impl.; value----value resulted by refe
rence impl. or your impl.
************performance evaluation*************
reference serial impl.  : time_cost=9.6570e-02
reference parallel impl.: time_cost=8.9829e+00 speedup=0.011
your impl.              : time_cost=7.1413e-02 speedup=1.352
```

加速比变化图(N=10)：



并行效率变化图(N=10)：


并行效率变化图
并行效率
3.5E+00
3.0E+00
2.5E+00
2.0E+00
1.5E+00
1.0E+00
5.0E-01
0.0E+00
3.3E+00
1.4E+00
1.1E+00
9.1E-01
1.2E-02
8.2E-02
1.5E-03
2.2E-02
2.1E-05
2.6E-03
4
16
64
128
512
P
Ref code
My code

三、评测结果分析

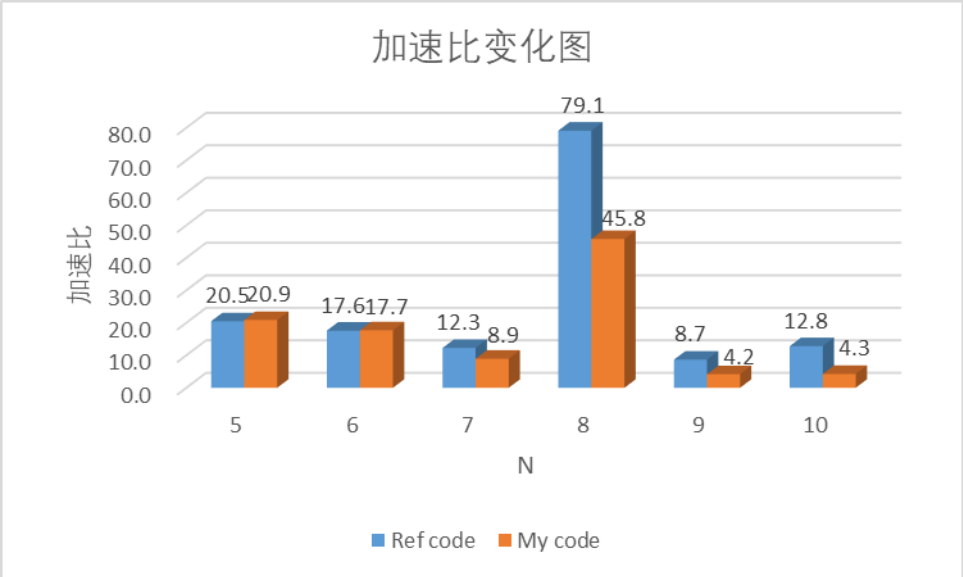


图 a 线程数量为 4，加速比随 N 变化

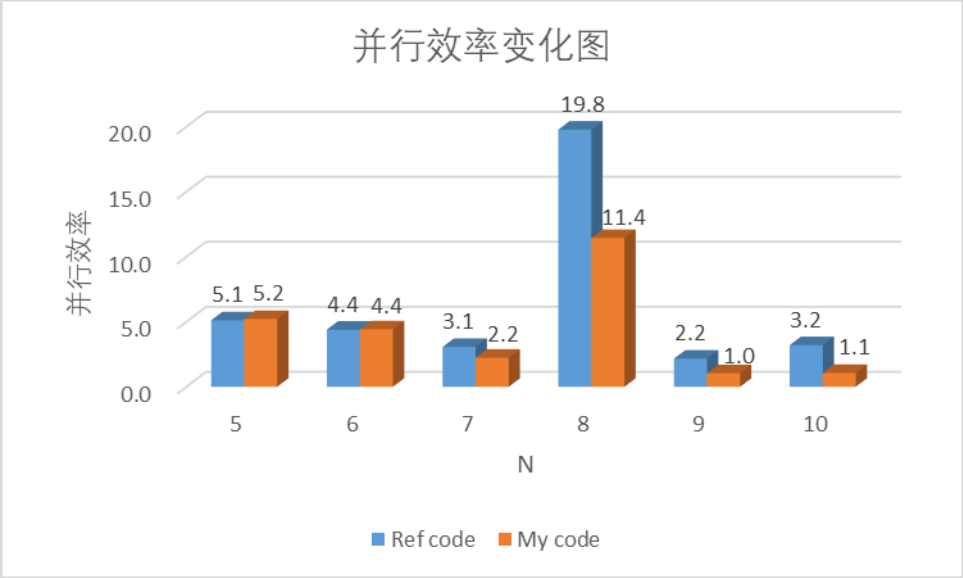


图 b 线程数量为 4，并行效率随 N 变化

图 a 为为 P=4 时加速比随着问题规模的变化图，在 easyHPC 平台上可以测出加速比都远远超出 p=4，这可能是因为加速时间计算在数据载入之后进行计算，否则，本问题的计算量在并行时并没有采用分治并行策略减少运算量，加速比不可能超过线程的数量。同理，图 b 也时如此。

在图 a/b 中，可以清晰地看见，除了在 N=8 时，出现很大程度的加速外，加速比/并行效率都随着问题规模的增大而减小，这是因为本问题的算法中，数据量越大，数据划分的连续性就越差，程序也需要耗费更多的时间用于同步和通信。

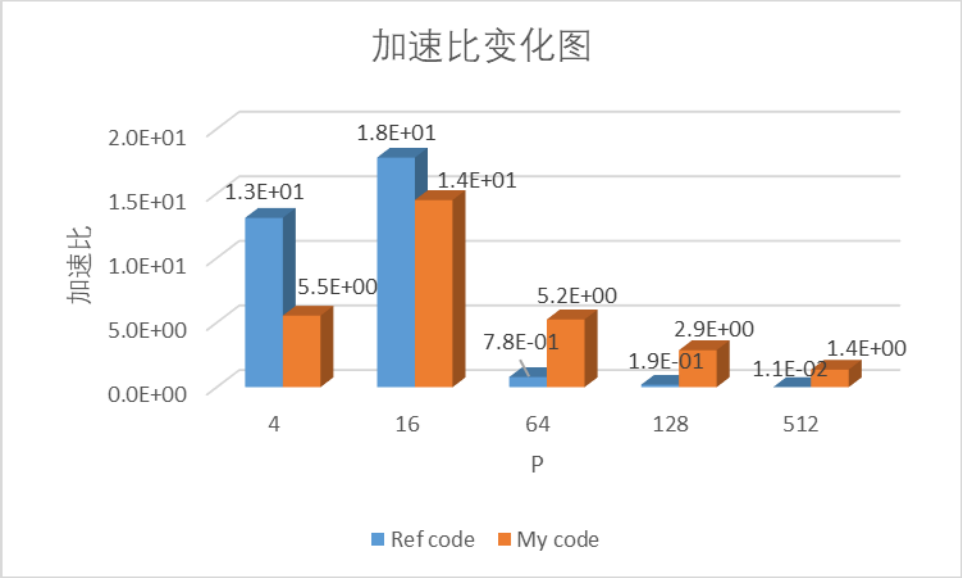


图 c M、N=10 时，加速比随线程数量变化图

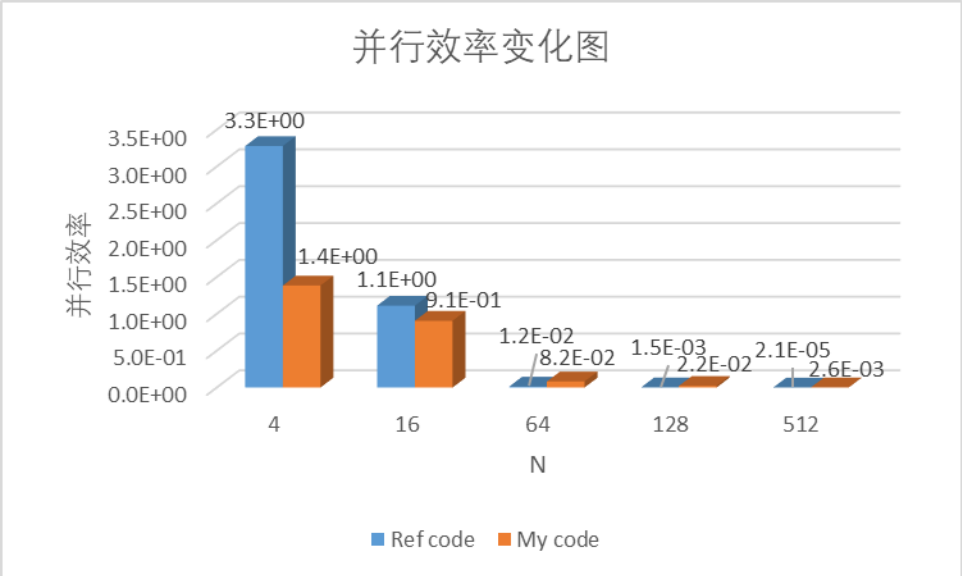


图 d M、N=10 时，并行效率随线程数量变化图

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，并行程序的加速比随着线程数量的增加，一开始出现了加速比增加的情况，之后加速比和并行效率都发生了骤减，显然是由于通信过程/资源分配过程耗费时间过长，并且同步耗费时间也很长。

四、Pthread 程序

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <vector>
#include <math.h>

#include "fio.h"

int64_t size;
int thread_num,threadid;
pthread_barrier_t barrier;
double *pbd, *pfc, *pfl ;
int64_t *lb,*ub;
int32_t T, t = 0;
int num;

void  *worker(void *arg) {
    int64_t i, j, w, cir;
    int myID = __sync_fetch_and_add(&threadid, 1);
    int64_t loc_size = (size) / thread_num;
    int64_t rest = (size) % thread_num;

    if (myID < rest) {
        lb[myID] = loc_size * myID + myID;
        ub[myID] = lb[myID] + loc_size + 1;
    } else {
        lb[myID] = loc_size * myID + rest;
        ub[myID] = lb[myID] + loc_size;
    }

    if (myID < num && thread_num % 2 == 0) {
```

```
        cir = num + 1;//w  Cycles
    } else {
        cir = num;
    }

    pthread_barrier_wait(&barrier);

    while (t < T) {
        //printf("t[%d]=%d\t",myID,t);
        for (w = myID; w < myID + cir; w++) {
            if (w < thread_num) {

//printf("lb[%d]=%ld\tub[%d]=%ld\n",myID,lb[myID],myID,ub[myID]);
                for (i = lb[myID]; i < ub[myID]; i++) {
                    int64_t bi = (i << 2);
                    int64_t fi = bi - i;

                    for (j = lb[w]; j < ub[w]; j++) {
                        int64_t bj = (j << 2);
                        int64_t fj = bj - j;
                        //printf("bj[%d]=%d",myID,bj);
                        double dx = pbd[bi + 1] - pbd[bj + 1];
                        double dy = pbd[bi + 2] - pbd[bj + 2];
                        double dz = pbd[bi + 3] - pbd[bj + 3];
                        double sq = dx * dx + dy * dy + dz * dz;
                        double dist = sqrt(sq);
                        double fac = G * pbd[bi] * pbd[bj] / (dist *
sq);
                        double fx = fac * dx;
                        double fy = fac * dy;
                        double fz = fac * dz;

                        pfc[fi] -= fx;
                        pfc[fi + 1] -= fy;
                        pfc[fi + 2] -= fz;
                        pfl[fj] += fx;
                        pfl[fj + 1] += fy;
                        pfl[fj + 2] += fz;

                    }
                }
            }
            if (w >= thread_num) {
```

```
            for (i = lb[w - thread_num]; i < ub[w - thread_num];
i++) {
                int64_t bi = (i << 2);
                int64_t fi = bi - i;
                for (j = lb[myID]; j < ub[myID]; j++) {
                    int64_t bj = (j << 2);
                    int64_t fj = bj - j;

                    double dx = pbd[bi + 1] - pbd[bj + 1];
                    double dy = pbd[bi + 2] - pbd[bj + 2];
                    double dz = pbd[bi + 3] - pbd[bj + 3];
                    double sq = dx * dx + dy * dy + dz * dz;
                    double dist = sqrt(sq);
                    double fac = G * pbd[bi] * pbd[bj] / (dist *
sq);
                    double fx = fac * dx;
                    double fy = fac * dy;
                    double fz = fac * dz;

                    pfc[fi] -= fx;
                    pfc[fi + 1] -= fy;
                    pfc[fi + 2] -= fz;
                    pfl[fj] += fx;
                    pfl[fj + 1] += fy;
                    pfl[fj + 2] += fz;
                }
            }
        }

        /*if (w - myID < num) {
            //printf("this is ID:%d,and front barrier\n", myID);
            pthread_barrier_wait(&barrier);

        }*/
    }
    //pthread_barrier_wait(&barrier);

         for (i = lb[myID]; i < ub[myID]; i++) {
            int64_t bi = (i << 2);
            int64_t fi = bi - i;
            pbd[bi + 1] = pbd[bi + 1] + (pfc[fi]+pfl[fi]) /
pbd[bi];
```

```
                pfc[fi] = 0;
                pfl[fi] = 0;
                pbd[bi + 2] = pbd[bi + 2] + (pfc[fi + 1]+pfl[fi + 1])
/ pbd[bi];
                pfc[fi + 1] = 0;
                pfl[fi + 1] = 0;
                pbd[bi + 3] = pbd[bi + 3] + (pfc[fi + 2]+pfl[fi + 2])
/ pbd[bi];
                pfc[fi + 2] = 0;
                pfl[fi + 2] = 0;
            }
        if(myID==0) {
            t++;
        }
        pthread_barrier_wait(&barrier);
    }
}




int main(int argc, char** argv ) {
    int64_t i, j;

    thread_num = atoi(argv[1]);
    size = atoll(argv[2]);
    T = atoi(argv[3]);
    // printf("T=%d",T);
    size = ((int64_t)1<<size);
    pbd = (double*)malloc(4*size*sizeof(double));

    lb = (int64_t*)malloc(thread_num*sizeof(int64_t));
    ub = (int64_t*)malloc(thread_num*sizeof(int64_t));

    pthread_barrier_init(&barrier, NULL, thread_num);
    threadid = 0;

    pfc = (double*)malloc(3*size*sizeof(double));
    memset(pfc, 0, 3*size*sizeof(double));

    pfl = (double*)malloc(3*size*sizeof(double));
    memset(pfl, 0, 3*size*sizeof(double));
    num = (thread_num + 1) / 2;
```

```
    input_data(pbd, 4*size*sizeof(double));


    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(pbd, 4*size*sizeof(double));
    free(pbd);
    free(pfc);
    pthread_barrier_destroy(&barrier);
    return EXIT_SUCCESS;
}
```