中山大学计算机学院本科生实验报告

课程名称：　超级计算机原理与操作　　　　任课教师：吴迪、黄聃

年级	2019 级	专业（方向）	计算机科学与技术（超算）
学号	19335074	姓名	黄玟瑜
开始日期	2021 年 4 月 29 日星期四	完成日期	2021 年 5 月 7 日星期五

一、 实验题目

根据 7-Development.pdf 课件在 nbody 问题或者 tsp 问题中二选一进行实现，要求实现一个串行版本和 MPI，OpenMP，pthread 中的任意两种版本。

nbody 问题

输入文件为 nbody.txt，输入每一行为一个 body，每列分别是质量，x 轴位置，y 轴位置，z 轴位置，x 轴速度，y 轴速度，z 轴速度

输出文件格式与输入一致，为 20 轮迭代后的结果

相关参数设置为 dT=0.005，G=1，迭代次数为 20

tsp 问题（任选一个数据集）

大规模输入文件为 tsp.txt，每一行为每个城市的坐标

小规模输入文件为 tsp2.txt，文件内是城市之间的距离矩阵

补充说明：可以自行调整数据规模，例如删减至 10 个城市，提交时需要将输入文件也一起打包

输出文件为经过城市的顺序（从 0 开始编号）

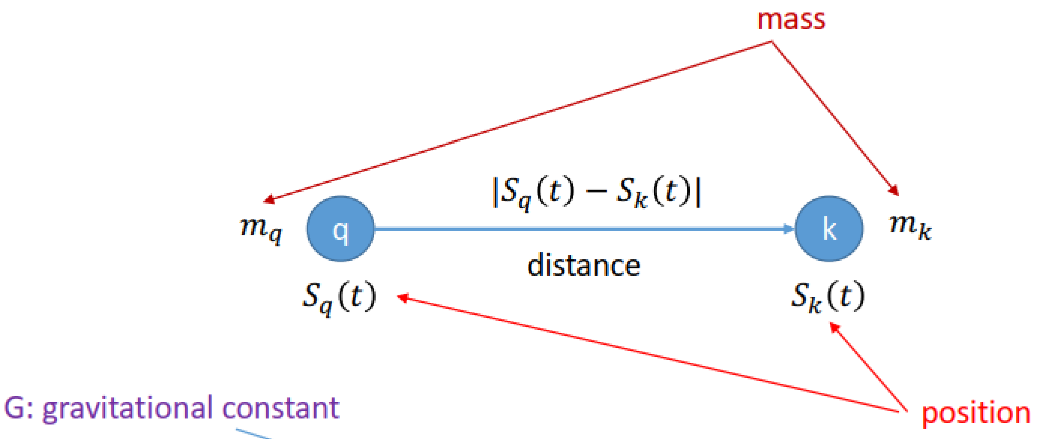
本次实验将求解 nbody 问题，实现一个串行版本和 MPI、OpenMP、pthread 三种版本的解法。

二、 实验内容

nbody 问题

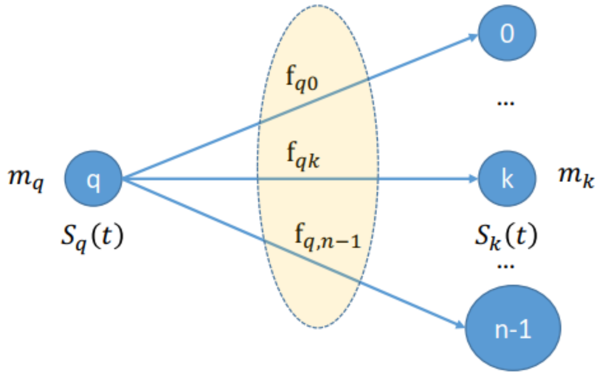
- 给定一系列物体的初始位置坐标、初始速度，求出在物体间引力作用下一段时间后它们的位置坐标、速度。
- 一个 nbody 问题的求解器通过模拟粒子的行为求解 nbody 问题。
  输入：物体数量 n、物体的质量 masses、物体的初始位置、物体的初始速度、经过时间 dT
  输出：dT 后物体的位置、速度

由牛顿的万有引力定律可以计算出两个物体之间的引力，对任意一个物体，它和其余 n-1 个物体间的引力之和即它所受到的合力，据此由牛顿第二定律可得到该物体的加速度，由加速度可得到它下一时刻的速度，由物体的起始位置和速度可得它下一时刻的速度。



$$\mathbf{f}_{qk}(t) = -\frac{\boxed{G} m_q m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3}\left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right] \qquad (6.1)$$

物体间作用力的计算公式



$$\mathbf{F}_q(t) = \sum_{\substack{k=0 \\ k\neq q}}^{n-1} \mathbf{f}_{qk} = -Gm_q \sum_{\substack{k=0 \\ k\neq q}}^{n-1} \frac{m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3}\left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right] \qquad (6.2)$$

原子受到的作用力之和

串行版本

伪代码如下：

```
//////////////////////////////////////////////////////////////////////////////
// Declare variables
// SCALE = 1024
// DIM = 3
double masses[SCALE];
double pos[SCALE][DIM];
double vel[SCALE][DIM];
double force_qk[DIM];
double forces[SCALE][DIM];

procedure serial solver

    Get input data
    get time start
    for each timestep do
        // Initialize forces to 0
        for each particle q do
            forces[q] = 0
        end

        // Compute total forces on q
        for each particle q do
            for each particle k>q do
                Compute force_qk(force between q and k)
                forces[q] += force_qk
                forces[k] -= force_qk
            end
        end

        // Compute position and velocity of q
        for each particle q do
            Compute vel[q]
            Compute pos[q]
        end
    end
    get time stop

    Print positions and velocities
    Print time stop-start

end procedure
```

解释：

- get time 部分通过 GET_TIME 函数获取当前时间并返回到变量中。

```
#define GET_TIME(now) { \
   struct timeval t; \
   gettimeofday(&t, NULL); \
   now = t.tv_sec + t.tv_usec/1000000.0; \
}
```

- get input 读取输入文件 nbody.txt 的信息并存储在 masses、pos、vel 中。

```
FILE* data= fopen("nbody.txt", "rb+");
```

```
    if(data==NULL)
        fprintf(stderr, "Can not open data file.\n");

    for(i=0; i<SCALE; i++){
        fscanf(data, "%lf %lf %lf %lf %lf %lf %lf", &masses[i], &pos[i][X], &pos[i][
Y], &pos[i][Z], &vel[i][X], &vel[i][Y], &vel[i][Z]);
    }
    fclose(data);
```

- 计算物体所受合力的具体过程如下：

```
        for(q=0; q<SCALE; q++){
            for(k=q+1; k<SCALE; k++){
                x_diff = pos[q][X] - pos[k][X];
                y_diff = pos[q][Y] - pos[k][Y];
                z_diff = pos[q][Z] - pos[k][Z];
                dist = sqrt(x_diff*x_diff + y_diff*y_diff + z_diff*z_diff);
                dist_cubed = dist*dist*dist;

                force_qk[X] = G*masses[q]*masses[k]/dist_cubed*x_diff;
                force_qk[Y] = G*masses[q]*masses[k]/dist_cubed*y_diff;
                force_qk[Z] = G*masses[q]*masses[k]/dist_cubed*z_diff;

                forces[q][X] -= force_qk[X];
                forces[q][Y] -= force_qk[Y];
                forces[q][Z] -= force_qk[Z];
                forces[k][X] += force_qk[X];
                forces[k][Y] += force_qk[Y];
                forces[k][Z] += force_qk[Z];
            }
        }
```

- 计算物体的位置、速度的具体过程如下：

```
        for(q=0; q<SCALE; q++){
            vel[q][X] += dT/masses[q]*forces[q][X];
            vel[q][Y] += dT/masses[q]*forces[q][Y];
            vel[q][Z] += dT/masses[q]*forces[q][Z];

            pos[q][X] += vel[q][X]*dT;
            pos[q][Y] += vel[q][Y]*dT;
            pos[q][Z] += vel[q][Z]*dT;
        }
```

OpenMP 版本

伪代码如下：

```
////////////////////////////////////////////////////////////////////////////
// Declare  shared variables
// SCALE = 1024
// DIM = 3
double masses[SCALE];
double pos[SCALE][DIM];
double vel[SCALE][DIM];

double forces[SCALE][DIM];

procedure openmp solver
    Get thread_count
    double loc_forces[thread_count][SCALE][DIM];

    Get input data

    get time start
    for each timestep do
        // Initialize loc_forces to 0
        # pragma omp for
        for each particle q do
            for each thread do
                loc_forces[thread][q] = 0
            end
        end

        // Compute local forces on q
        # pragma omp for
        for each particle q do
            for each particle k>q do
                Compute force_qk(force between q and k)
                loc_forces[q] += force_qk
                loc_forces[k] -= force_qk
            end
        end

        // Gather all the local forces
        # pragma omp for
        for each particle q do
            forces[q] = 0
            for each thread do
                forces[q] += loc_forces[thread][q]
            end
        end

        // Compute position and velocity of q
        # pragma omp for
        for each particle q do
```

```
            Compute vel[q]
            Compute pos[q]
        end
    end
    get time stop

    Print positions and velocities
    Print time stop-start

end procedure
```

解释：

- 为了防止竞争，用 loc_forces 存储各个线程的计算结果，最后再汇总到 forces 中。为了减少不必要的空间浪费，根据输入的线程数申请 loc_forces 的空间。

```
if (argc != 2) Usage(argv[0]);
Get_args(argv, &thread_count);   //获取线程数
double loc_forces[thread_count][SCALE][DIM];
```

- 获取输入的过程和并行版本相同。

- 初始化 loc_forces 的过程如下：

```
      # pragma omp for
         for(q=0; q<SCALE; q++){
            for(thread=0; thread<thread_count; thread++)
               loc_forces[thread][q][X] = loc_forces[thread][q][Y] = loc_forces[th
read][q][Z] = 0;
         }
```

- 对于计算 loc_forces 的过程中使用到的变量 q, k, x_diff, y_diff, z_diff, dist, dist_cubed 等，若将它们用作共享变量中将会导致竞争冲突，使得计算结果出错，因此需要把它们的声明放在各个线程的内部。

```
      # pragma omp parallel num_threads(thread_count)
      {
         int q, k;
         int my_rank = omp_get_thread_num();
         double x_diff, y_diff, z_diff, dist, dist_cubed;
         double force_qk[DIM];

         # pragma omp for
         for(q=0; q<SCALE; q++){
            for(k=q+1; k<SCALE; k++){
               x_diff = pos[q][X] - pos[k][X];
               y_diff = pos[q][Y] - pos[k][Y];
               z_diff = pos[q][Z] - pos[k][Z];
               dist = sqrt(x_diff*x_diff + y_diff*y_diff + z_diff*z_diff);
               dist_cubed = dist*dist*dist;

               force_qk[X] = G*masses[q]*masses[k]/dist_cubed*x_diff;
               force_qk[Y] = G*masses[q]*masses[k]/dist_cubed*y_diff;
               force_qk[Z] = G*masses[q]*masses[k]/dist_cubed*z_diff;

               loc_forces[my_rank][q][X] -= force_qk[X];
               loc_forces[my_rank][q][Y] -= force_qk[Y];
               loc_forces[my_rank][q][Z] -= force_qk[Z];
               loc_forces[my_rank][k][X] += force_qk[X];
               loc_forces[my_rank][k][Y] += force_qk[Y];
               loc_forces[my_rank][k][Z] += force_qk[Z];
            }
         }
      }
```

- 对每个物体 q，将各个线程计算出的对 q 的作用力汇总：

```
# pragma omp for
for(q=0; q<SCALE; q++){
   forces[q][X] = forces[q][Y] = forces[q][Z] = 0;
   for(thread=0; thread<thread_count; thread++){
      forces[q][X] += loc_forces[thread][q][X];
      forces[q][Y] += loc_forces[thread][q][Y];
      forces[q][Z] += loc_forces[thread][q][Z];
   }
}
```

- 计算物体的位置、速度的具体过程如下：

```
# pragma omp for
for(q=0; q<SCALE; q++){
   vel[q][X] += dT/masses[q]*forces[q][X];
   vel[q][Y] += dT/masses[q]*forces[q][Y];
   vel[q][Z] += dT/masses[q]*forces[q][Z];

   pos[q][X] += vel[q][X]*dT;
   pos[q][Y] += vel[q][Y]*dT;
   pos[q][Z] += vel[q][Z]*dT;
}
```

Pthread 版本

伪代码如下：

```
////////////////////////////////////////////////////////////////
// Declare  shared variables
// SCALE = 1024
// DIM = 3
```

```
int thread_count;

double masses[SCALE];
double pos[SCALE][DIM];
double vel[SCALE][DIM];

double force_qk[DIM];

double forces[SCALE][DIM];
double loc_forces[MAX_THREADS][SCALE][DIM];

procedure pthread solver
    Get thread_count
    Malloc space of thread_handles

    Get input data

    get time start
    for each timestep do
        // Initialize forces to 0
        for each thread do
            pthread_create thread_initForces
        end

        for each thread do
            pthread_join
        end

        // Compute local forces on q
        for each thread do
            pthread_create thread_compForce
        end

        for each thread do
            pthread_join
        end

        // Gather all the local forces
        for each thread do
            pthread_create thread_reducForce
        end

        for each thread do
            pthread_join
        end
        // Compute position and velocity of q
        for each thread do
            pthread_create thread_compPosAndVel
        end

        for each thread do
            pthread_join
        end
    end
    get time stop

    Print positions and velocities
    Print time stop-start

end procedure

function thread_initForces
input: my_rank, loc_forces
output: set loc_forces[my_rank] to 0

for each particle q do
    loc_forces[my_rank][q] = 0
end

end function

function thread_compForce
input: my_rank, masses, pos
output: loc_forces

my_n = SCALE/thread_count
my_first_q = my_rank*my_n
my_last_q = my_first_q+my_n

for each particle q in [my_first_q, my_last_q] do
    for each particle k>q do
        Compute force_qk (force between q and k)
        loc_forces[q] += force_qk
        loc_forces[k] -= force_qk
    end
end
end function

function thread_reducForce
input: my_rank, loc_forces
output: forces
```

```
my_n = SCALE/thread_count
my_first_q = my_rank*my_n
my_last_q = my_first_q+my_n

for each particle q in [my_first_q, my_last_q] do
    forces[q] = 0
    for each thread do
        forces[q] += loc_forces[thread][q]
    end
end

end function

function thread_compPosAndVel
input: my_rank, forces, pos, vel
output: updated pos and vel

my_n = SCALE/thread_count
my_first_q = my_rank*my_n
my_last_q = my_first_q+my_n

for each particle q in [my_first_q, my_last_q] do
    Compute vel[q]
    Compute pos[q]
end
end function
```

解释：

- 各个线程函数声明如下：

```
void* thread_initForces(void* args_p);
void* thread_compForce(void* args_p);
void* thread_reducForce(void* args_p);
void* thread_compPosAndVel(void* args_p);
```

  对于每一个线程，它们执行时的输入除了 my_rank 都是共享的，因此只需要传递参数 thread（对应线程的 my_rank）即可。

- 对线程函数 thread_initForces，任务分配方式是每个线程处理自己被分配的 loc_forces 的部分。

- 对线程函数 thread_compForce、thread_reducForce、thread_compPosAndVel，任务分配方式是将物体按块划分到各个线程去处理，每个线程处理自己范围内的物体（[my_first_q, my_last_q]）。

- 每次调用线程函数后都要使用 pthread_join 设置路障，因为各个步骤间有先后关系，必须等待所

有线程都得出运算结果才能进行下一步的运算。

- 其余部分和 OpenMP 版本大致相同。

MPI 版本

由于进程间不共享内存空间，因此 MPI 版本的解法和 OpenMP、Pthread 大不相同。

每个进程都需要保存物体的质量 masses。

每个进程处理 loc_n = SCALE/comm_sz 个物体，使用 loc_pos 保存进程所处理的物体的位置，使用 loc_vel 保存进程所处理的物体的速度，使用 loc_forces 存储进程所处理的物体受到的合力，使用 tmp_pos 和 tmp_forces 暂时存储目标物体的质量和受到力。

为了高效地使用 MPI 的库函数，使用一维数组来存储物体的位置、速度信息，在前述解法中都是使用二维数组来存储多维度的位置、速度信息，例如 double pos[SCALE][DIM]，物体 q 在 X 轴的坐标为 pos[q][X]，在 MPI 版本中将使用一维数组来存储，例如
double* pos; pos = malloc(SCALE*DIM*sizeof(double));物体 q 在 X 轴的坐标为
pos[q*DIM+X]，使用这种方法的存储的好处是可以使用 MPI_Send、MPI_Recv
、MPI_Scatter 等 MPI 库函数，因为这些函数只支持一维数组的操作。

进程间呈环形传递信息。

各个进程一开始时只持有自己所处理的物体的位置信息，在计算完进程内的物体间作用力后，将自己所持有的物体的位置信息及其所受的力传递给它的邻居进程，同时从它的另一个邻居进程获取它所持有的物体的位置信息及其所受的力，再计算自己所处理的物体和接收到的物体的物体间作用力，再将接收的物体的位置信息及其所受的力传递给它的邻居进程……最后对每个进程，它所处理的物体和其他进程所处理的物体之间的力都被计算了，此时 loc_forces 存储的便是自己所处理的物体所受的合力。

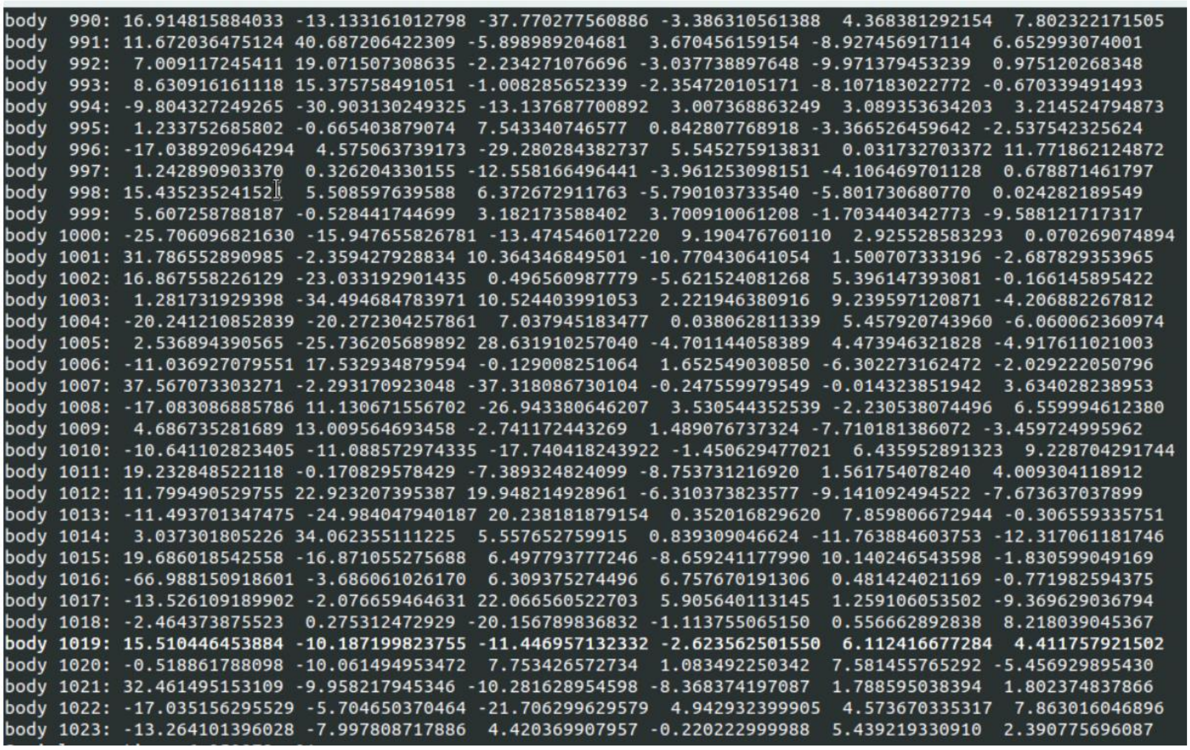
伪代码如下：

```
//////////////////////////////////////////////////////////
procedure mpi solver
Get comm_sz
Get my_rank

source = (my_rank+1)%comm_sz
dest = (my_rank-1+comm_sz)%comm_sz

Get input data
MPI_Scatter pos to every process as loc_pos
MPI_Scatter vel to every process as loc_vel
MPI_Bcast masses
get time start
for each timestep do
    copy loc_pos into tmp_pos
    loc_forces = tmp_forces =0

    // Compute forces due to interactions among local particles
    for each local particle q do
        for each local particle k>q do
            Compute force_qk (force between q and k)
            loc_forces += force_qk
            tmp_forces -= force_qk
        end
    end
    for phase = 1 to comm_sz-1 do
        Send tmp_pos to dest
        Send tmp_forces to dest
        Recv tmp_pos from source
        Recv tmp_forces from source

        owner = (my_rank+phase)%comm_sz

        Compute forces due to interactions among my particles and owner’s particles
    end
    // Recv my particles' pos and forces
    Send tmp_pos to dest
    Send tmp_forces to dest
    Recv tmp_pos from source
    Recv tmp_forces from source

    // Add tmp_forces to loc_forces
    for each local particle q do
        loc_forces += tmp_forces
    end

    for each local particle do
        Compute loc_vel[q]
        Compute loc_pos[q]
    end
end
get time stop

if my_rank = 0 do
    Malloc space of pos, vel
end
MPI_Gather loc_pos to pos
MPI_Gather loc_vel to vel
if my_rank = 0 do
    Print pos, vel
    Print time stop-start
end
end procedure
```

三、 实验结果

串行版本

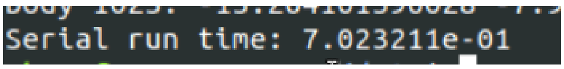
运行结果如下：



```
body  990: 16.914815884033 -13.133161012798 -37.770277560886 -3.386310561388  4.368381292154  7.802322171505
body  991: 11.672036475124 40.687206422309 -5.898989204681  3.670456159154 -8.927456917114  6.652993074001
body  992:  7.009117245411 19.071507308635 -2.234271076696 -3.037738897648 -9.971379453239  0.975120268348
body  993:  8.630916161118 15.375758491051 -1.008285652339 -2.354720105171 -8.107183022772 -0.670339491493
body  994: -9.804327249265 -30.903130249325 -13.137687700892  3.007368863249  3.089353634203  3.214524794873
body  995:  1.233752685802 -0.665403879074  7.543340746577  0.842807768918 -3.366526459642 -2.537542325624
body  996: -17.038920964294  4.575063739173 -29.280284382737  5.545275913831  0.031732703372 11.771862124872
body  997:  1.242890903370  0.326204330155 -12.558166496441 -3.961253098151 -4.106469701128  0.678871461797
body  998: 15.43523524152  5.508597639588  6.372672911763 -5.790103733540 -5.801730680770  0.024282189549
body  999:  5.607258788187 -0.528441744699  3.182173588402  3.700910061208 -1.703440342773 -9.588121717317
body 1000: -25.706096821630 -15.947655826781 -13.474546017220  9.190476760110  2.925528583293  0.070269074894
body 1001: 31.786552890985 -2.359427928834 10.364346849501 -10.770430641054  1.500707333196 -2.687829353965
body 1002: 16.867558226129 -23.033192901435  0.496560987779 -5.621524081268  5.396147393081 -0.166145895422
body 1003:  1.281731929398 -34.494684783971 10.524403991053  2.221946380916  9.239597120871 -4.206882267812
body 1004: -20.241210852839 -20.272304257861  7.037945183477  0.038062811339  5.457920743960 -6.060062360974
body 1005:  2.536894390565 -25.736205689892 28.631910257040 -4.701144058389  4.473946321828 -4.917611021003
body 1006: -11.036927079551 17.532934879594 -0.129008251064  1.652549030850 -6.302273162472 -2.029222050796
body 1007: 37.567073303271 -2.293170923048 -37.318086730104 -0.247559979549 -0.014323851942  3.634028238953
body 1008: -17.083086885786 11.130671556702 -26.943380646207  3.530544352539 -2.230538074496  6.559994612380
body 1009:  4.686735281689 13.009564693458 -2.741172443269  1.489076737324 -7.710181386072 -3.459724995962
body 1010: -10.641102823405 -11.088572974335 -17.740418243922 -1.450629477021  6.435952891323  9.228704291744
body 1011: 19.232848522118 -0.170829578429 -7.389324824099 -8.753731216920  1.561754078240  4.009304118912
body 1012: 11.799490529755 22.923207395387 19.948214928961 -6.310373823577 -9.141092494522 -7.673637037899
body 1013: -11.493701347475 -24.984047940187 20.238181879154  0.352016829620  7.859806672944 -0.306559335751
body 1014:  3.037301805226 34.062355111225  5.557652759915  0.839309046624 -11.763884603753 -12.317061181746
body 1015: 19.686018542558 -16.871055275688  6.497793777246 -8.659241177990 10.140246543598 -1.830599049169
body 1016: -66.988150918601 -3.686061026170  6.309375274496  6.757670191306  0.481424021169 -0.771982594375
body 1017: -13.526109189902 -2.076659464631 22.066560522703  5.905640113145  1.259106053502 -9.369629036794
body 1018: -2.464373875523  0.275312472929 -20.156789836832 -1.113755065150  0.556662892838  8.218039045367
body 1019: 15.510446453884 -10.187199823755 -11.446957132332 -2.623562501550  6.112416677284  4.411757921502
body 1020: -0.518861788098 -10.061494953472  7.753426572734  1.083492250342  7.581455765292 -5.456929895430
body 1021: 32.461495153109 -9.958217945346 -10.281628954598 -8.368374197087  1.788595038394  1.802374837866
body 1022: -17.035156295529 -5.704650370464 -21.706299629579  4.942932399905  4.573670335317  7.863016046896
body 1023: -13.264101396028 -7.997808717886  4.420369907957 -0.220222999988  5.439219330910  2.390775696087
```

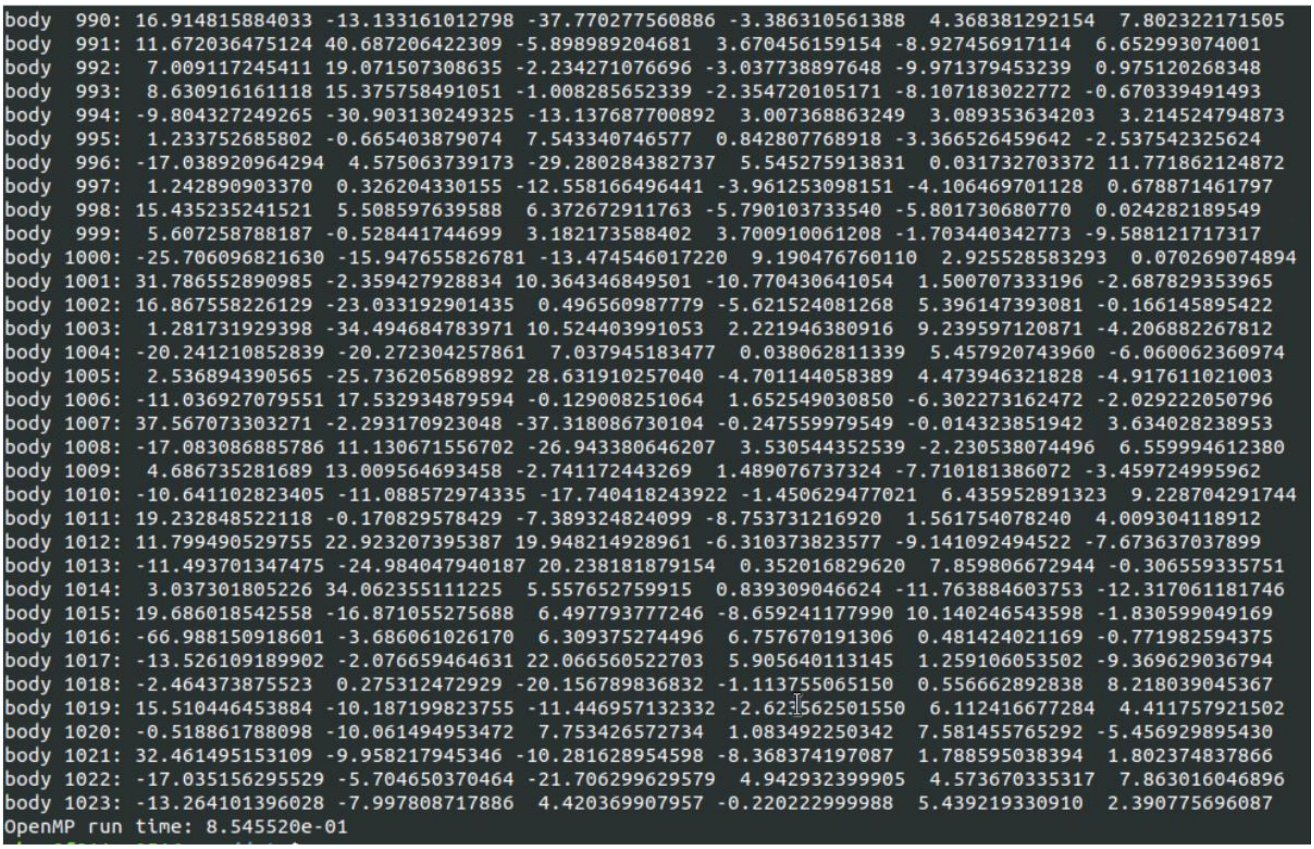
经分析，与参考输出存在的误差<10^{-7}。

运行 7 次（输出结果相同，运行时间不同），每次的运行时间和平均运行时间如下：

序号	运行时间	平均运行时间
1	Serial run time: 6.950879e-01	6.987495×10^{-1}
2	Serial run time: 6.995649e-01	
3	Serial run time: 6.954870e-01	
4	Serial run time: 6.993802e-01	
5	 Serial run time: 7.023211e-01	
6	Serial run time: 6.982172e-01	
7	Serial run time: 7.011881e-01	

OpenMP 版本

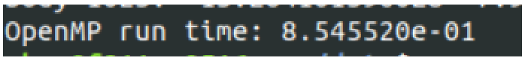
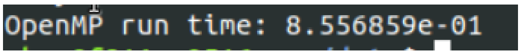
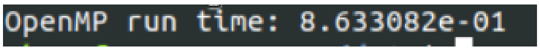
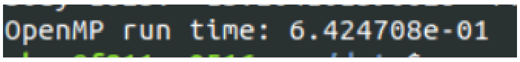
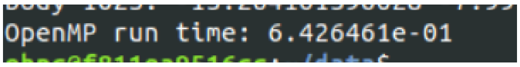
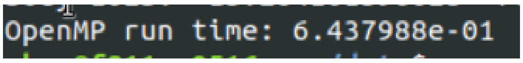
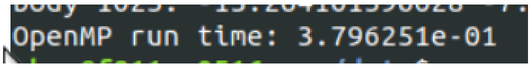
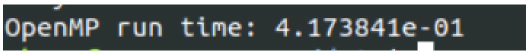
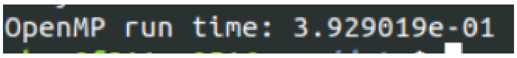
运行结果如下：

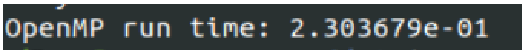
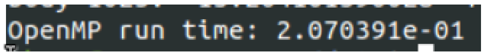
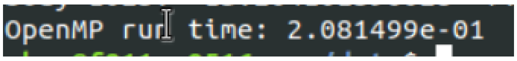
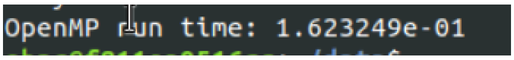
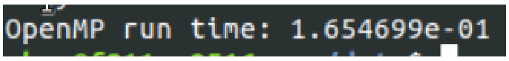
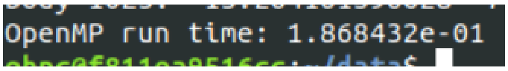
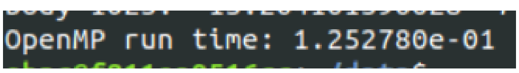
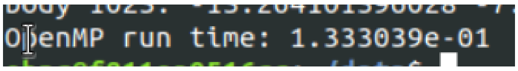
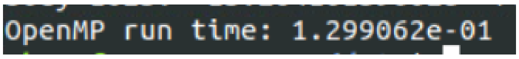


```
body  990: 16.914815884033 -13.133161012798 -37.770277560886 -3.386310561388  4.368381292154  7.802322171505
body  991: 11.672036475124 40.687206422309 -5.898989204681  3.670456159154 -8.927456917114  6.652993074001
body  992:  7.009117245411 19.071507308635 -2.234271076696 -3.037738897648 -9.971379453239  0.975120268348
body  993:  8.630916161118 15.375758491051 -1.008285652339 -2.354720105171 -8.107183022772 -0.670339491493
body  994: -9.804327249265 -30.903130249325 -13.137687700892  3.007368863249  3.089353634203  3.214524794873
body  995:  1.233752685802 -0.665403879074  7.543340746577  0.842807768918 -3.366526459642 -2.537542325624
body  996: -17.038920964294  4.575063739173 -29.280284382737  5.545275913831  0.031732703372 11.771862124872
body  997:  1.242890903370  0.326204330155 -12.558166496441 -3.961253098151 -4.106469701128  0.678871461797
body  998: 15.435235241521  5.508597639588  6.372672911763 -5.790103733540 -5.801730680770  0.024282189549
body  999:  5.607258788187 -0.528441744699  3.182173588402  3.700910061208 -1.703440342773 -9.588121717317
body 1000: -25.706096821630 -15.947655826781 -13.474546017220  9.190476760110  2.925528583293  0.070269074894
body 1001: 31.786552890985 -2.359427928834 10.364346849501 -10.770430641054  1.500707333196 -2.687829353965
body 1002: 16.867558226129 -23.033192901435  0.496560987779 -5.621524081268  5.396147393081 -0.166145895422
body 1003:  1.281731929398 -34.494684783971 10.524403991053  2.221946380916  9.239597120871 -4.206882267812
body 1004: -20.241210852839 -20.272304257861  7.037945183477  0.038062811339  5.457920743960 -6.060062360974
body 1005:  2.536894390565 -25.736205689892 28.631910257040 -4.701144058389  4.473946321828 -4.917611021003
body 1006: -11.036927079551 17.532934879594 -0.129008251064  1.652549030850 -6.302273162472 -2.029222050796
body 1007: 37.567073303271 -2.293170923048 -37.318086730104 -0.247559979549 -0.014323851942  3.634028238953
body 1008: -17.083086885786 11.130671556702 -26.943380646207  3.530544352539 -2.230538074496  6.559994612380
body 1009:  4.686735281689 13.009564693458 -2.741172443269  1.489076737324 -7.710181386072 -3.459724995962
body 1010: -10.641102823405 -11.088572974335 -17.740418243922 -1.450629477021  6.435952891323  9.228704291744
body 1011: 19.232848522118 -0.170829578429 -7.389324824099 -8.753731216920  1.561754078240  4.009304118912
body 1012: 11.799490529755 22.923207395387 19.948214928961 -6.310373823577 -9.141092494522 -7.673637037899
body 1013: -11.493701347475 -24.984047940187 20.238181879154  0.352016829620  7.859806672944 -0.306559335751
body 1014:  3.037301805226 34.062355111225  5.557652759915  0.839309046624 -11.763884603753 -12.317061181746
body 1015: 19.686018542558 -16.871055275688  6.497793777246 -8.659241177990 10.140246543598 -1.830599049169
body 1016: -66.988150918601 -3.686061026170  6.309375274496  6.757670191306  0.481424021169 -0.771982594375
body 1017: -13.526109189902 -2.076659464631 22.066560522703  5.905640113145  1.259106053502 -9.369629036794
body 1018: -2.464373875523  0.275312472929 -20.156789836832 -1.113755065150  0.556662892838  8.218039045367
body 1019: 15.510446453884 -10.187199823755 -11.446957132332 -2.623562501550  6.112416677284  4.411757921502
body 1020: -0.518861788098 -10.061494953472  7.753426572734  1.083492250342  7.581455765292 -5.456929895430
body 1021: 32.461495153109 -9.958217945346 -10.281628954598 -8.368374197087  1.788595038394  1.802374837866
body 1022: -17.035156295529 -5.704650370464 -21.706299629579  4.942932399905  4.573670335317  7.863016046896
body 1023: -13.264101396028 -7.997808717886  4.420369907957 -0.220222999988  5.439219330910  2.390775696087
OpenMP run time: 8.545520e-01
```

经分析，与参考输出存在的误差<10^{-7}，与串行版本的输出完全相同。

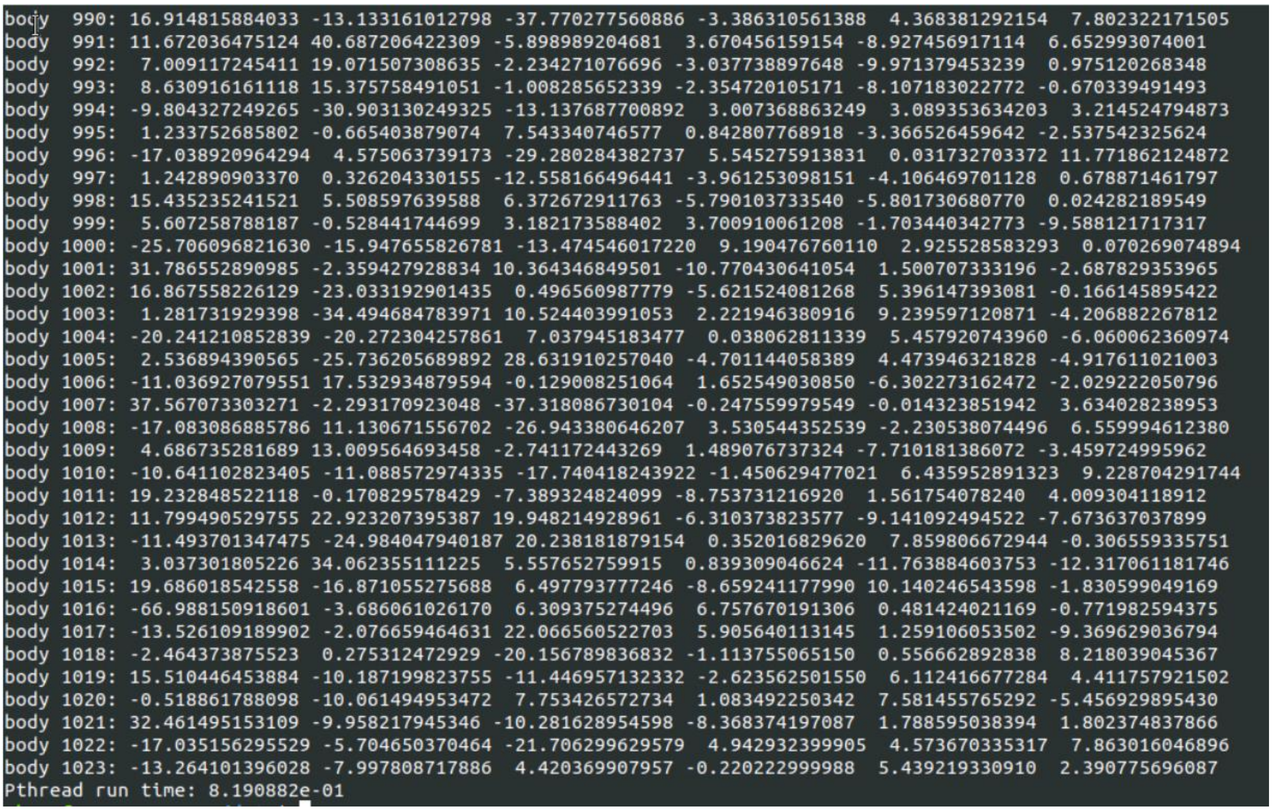
分别以 1、2、4、8、16、32 个线程运行 OpenMP 版本的求解器，每种情况运行 3 次并计算平均运行时间，如下所示：

线程数	运行时间	平均运行时间
1	 OpenMP run time: 8.545520e-01	8.578487×10^{-1}
	 OpenMP run time: 8.556859e-01	
	 OpenMP run time: 8.633082e-01	
2	 OpenMP run time: 6.424708e-01	6.429719×10^{-1}
	 OpenMP run time: 6.426461e-01	
	 OpenMP run time: 6.437988e-01	
4	 OpenMP run time: 3.796251e-01	3.966370×10^{-1}
	 OpenMP run time: 4.173841e-01	
	 OpenMP run time: 3.929019e-01	

8	OpenMP run time: 2.303679e-01	2.151856×10^{-1}
	OpenMP run time: 2.070391e-01	
	OpenMP run time: 2.081499e-01	
16	OpenMP run time: 1.623249e-01	1.715460×10^{-1}
	OpenMP run time: 1.654699e-01	
	OpenMP run time: 1.868432e-01	
32	OpenMP run time: 1.252780e-01	1.294960×10^{-1}
	OpenMP run time: 1.333039e-01	
	OpenMP run time: 1.299062e-01	

Pthread 版本

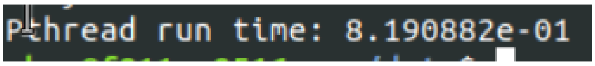
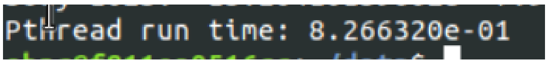
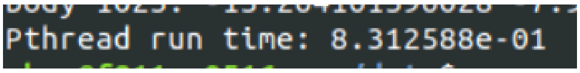
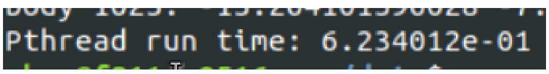
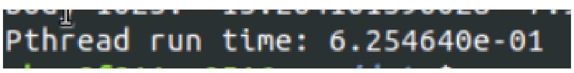
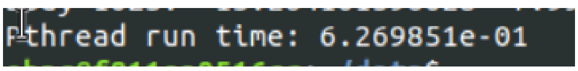
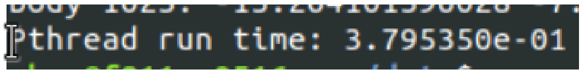
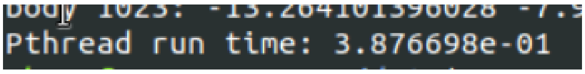
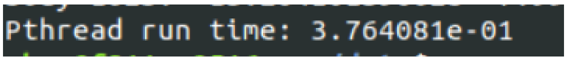
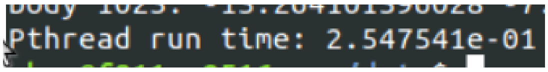
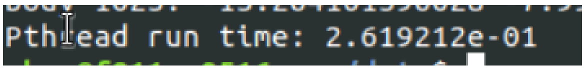
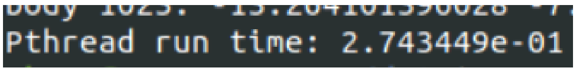
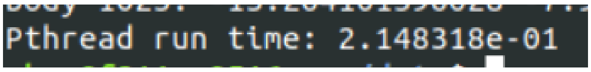
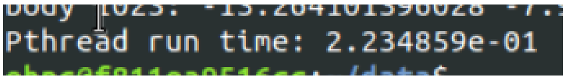
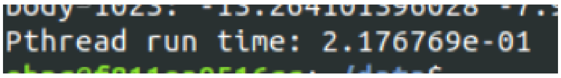
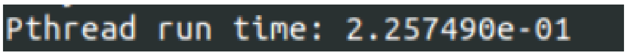
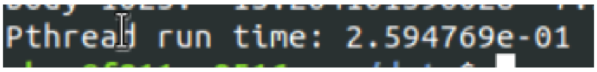
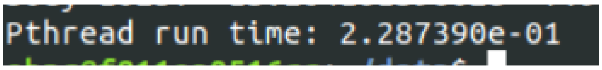
运行结果如下：



```
body  990: 16.914815884033 -13.133161012798 -37.770277560886 -3.386310561388  4.368381292154  7.802322171505
body  991: 11.672036475124 40.687206422309 -5.898989204681  3.670456159154 -8.927456917114  6.652993074001
body  992:  7.009117245411 19.071507308635 -2.234271076696 -3.037738897648 -9.971379453239  0.975120268348
body  993:  8.630916161118 15.375758491051 -1.008285652339 -2.354720105171 -8.107183022772 -0.670339491493
body  994: -9.804327249265 -30.903130249325 -13.137687700892  3.007368863249  3.089353634203  3.214524794873
body  995:  1.233752685802 -0.665403879074  7.543340746577  0.842807768918 -3.366526459642 -2.537542325624
body  996: -17.038920964294  4.575063739173 -29.280284382737  5.545275913831  0.031732703372 11.771862124872
body  997:  1.242890903370  0.326204330155 -12.558166496441 -3.961253098151 -4.106469701128  0.678871461797
body  998: 15.435235241521  5.508597639588  6.372672911763 -5.790103733540 -5.801730680770  0.024282189549
body  999:  5.607258788187 -0.528441744699  3.182173588402  3.700910061208 -1.703440342773 -9.588121717317
body 1000: -25.706096821630 -15.947655826781 -13.474546017220  9.190476760110  2.925528583293  0.070269074894
body 1001: 31.786552890985 -2.359427928834 10.364346849501 -10.770430641054  1.500707333196 -2.687829353965
body 1002: 16.867558226129 -23.033192901435  0.496560987779 -5.621524081268  5.396147393081 -0.166145895422
body 1003:  1.281731929398 -34.494684783971 10.524403991053  2.221946380916  9.239597120871 -4.206882267812
body 1004: -20.241210852839 -20.272304257861  7.037945183477  0.038062811339  5.457920743960 -6.060062360974
body 1005:  2.536894390565 -25.736205689892 28.631910257040 -4.701144058389  4.473946321828 -4.917611021003
body 1006: -11.036927079551 17.532934879594 -0.129008251064  1.652549030850 -6.302273162472 -2.029222050796
body 1007: 37.567073303271 -2.293170923048 -37.318086730104 -0.247559979549 -0.014323851942  3.634028238953
body 1008: -17.083086885786 11.130671556702 -26.943380646207  3.530544352539 -2.230538074496  6.559994612380
body 1009:  4.686735281689 13.009564693458 -2.741172443269  1.489076737324 -7.710181386072 -3.459724995962
body 1010: -10.641102823405 -11.088572974335 -17.740418243922 -1.450629477021  6.435952891323  9.228704291744
body 1011: 19.232848522118 -0.170829578429 -7.389324824099 -8.753731216920  1.561754078240  4.009304118912
body 1012: 11.799490529755 22.923207395387 19.948214928961 -6.310373823577 -9.141092494522 -7.673637037899
body 1013: -11.493701347475 -24.984047940187 20.238181879154  0.352016829620  7.859806672944 -0.306559335751
body 1014:  3.037301805226 34.062355111225  5.557652759915  0.839309046624 -11.763884603753 -12.317061181746
body 1015: 19.686018542558 -16.871055275688  6.497793777246 -8.659241177990 10.140246543598 -1.830599049169
body 1016: -66.988150918601 -3.686061026170  6.309375274496  6.757670191306  0.481424021169 -0.771982594375
body 1017: -13.526109189902 -2.076659464631 22.066560522703  5.905640113145  1.259106053502 -9.369629036794
body 1018: -2.464373875523  0.275312472929 -20.156789836832 -1.113755065150  0.556662892838  8.218039045367
body 1019: 15.510446453884 -10.187199823755 -11.446957132332 -2.623562501550  6.112416677284  4.411757921502
body 1020: -0.518861788098 -10.061494953472  7.753426572734  1.083492250342  7.581455765292 -5.456929895430
body 1021: 32.461495153109 -9.958217945346 -10.281628954598 -8.368374197087  1.788595038394  1.802374837866
body 1022: -17.035156295529 -5.704650370464 -21.706299629579  4.942932399905  4.573670335317  7.863016046896
body 1023: -13.264101396028 -7.997808717886  4.420369907957 -0.220222999988  5.439219330910  2.390775696087
Pthread run time: 8.190882e-01
```

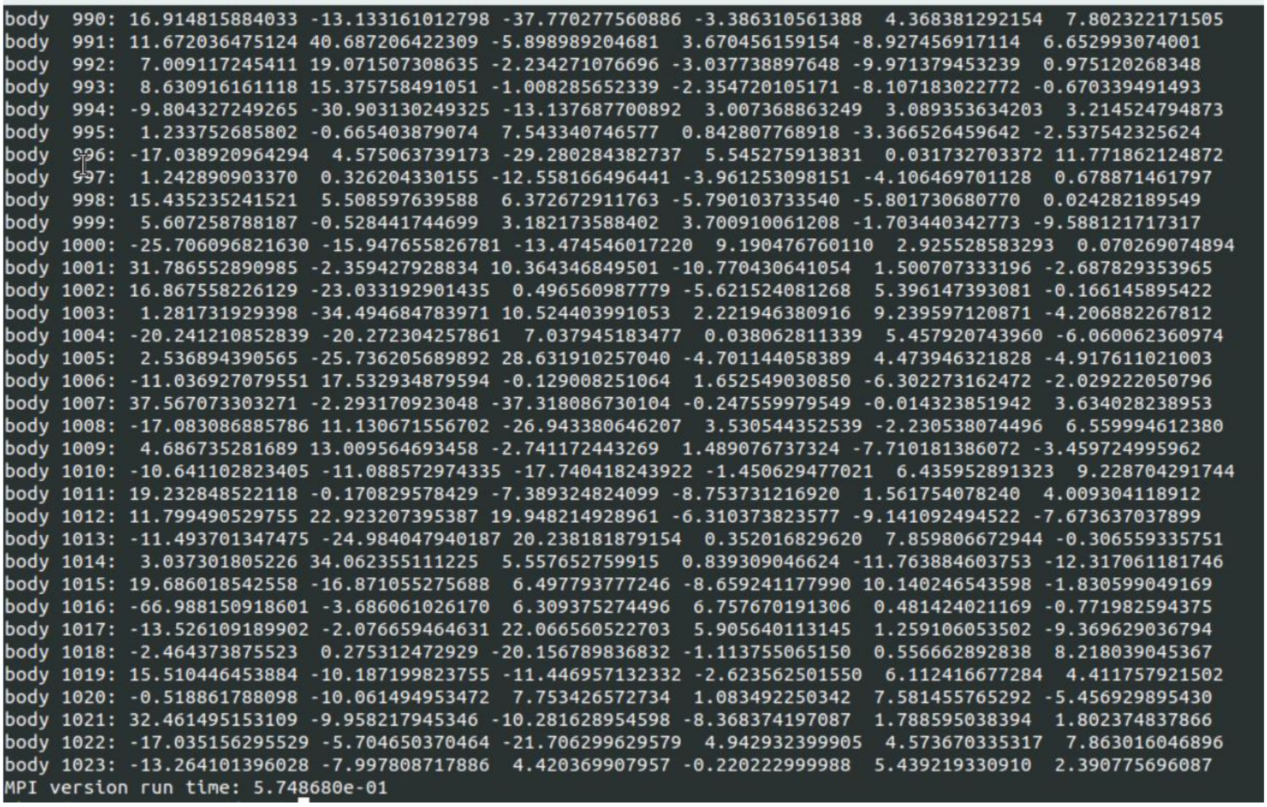
经分析，与参考输出存在的误差<10^{-7}，与串行版本的输出完全相同。

分别以 1、2、4、8、16、32 个线程运行 Pthread 版本的求解器，每种情况运行 3 次并计算平均运行时间，如下所示：

线程数	运行时间	平均运行时间
1	 Pthread run time: 8.190882e-01	8.256597×10^{-1}
	 Pthread run time: 8.266320e-01	
	 Pthread run time: 8.312588e-01	
2	 Pthread run time: 6.234012e-01	6.252744×10^{-1}
	 Pthread run time: 6.254640e-01	
	 Pthread run time: 6.269851e-01	
4	 Pthread run time: 3.795350e-01	3.812043×10^{-1}
	 Pthread run time: 3.876698e-01	
	 Pthread run time: 3.764081e-01	
8	 Pthread run time: 2.547541e-01	2.636734×10^{-1}
	 Pthread run time: 2.619212e-01	
	 Pthread run time: 2.743449e-01	
16	 Pthread run time: 2.148318e-01	2.1866487×10^{-1}
	 Pthread run time: 2.234859e-01	
	 Pthread run time: 2.176769e-01	
32	 Pthread run time: 2.257490e-01	2.379883×10^{-1}
	 Pthread run time: 2.594769e-01	
	 Pthread run time: 2.287390e-01	

MPI 版本

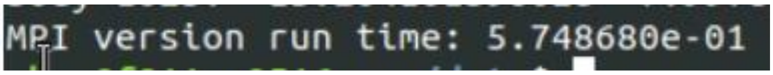
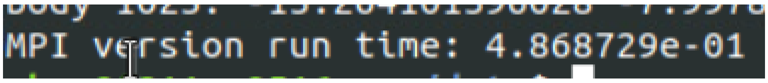
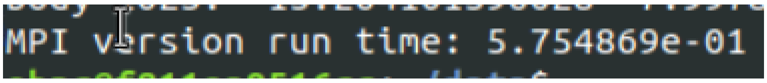
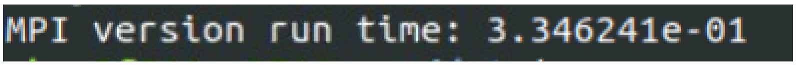
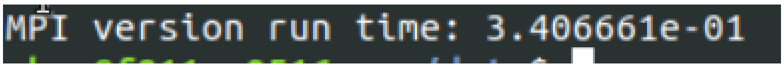
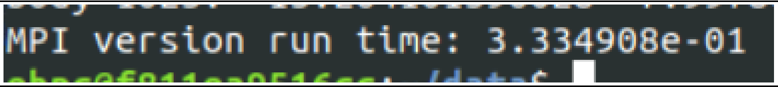
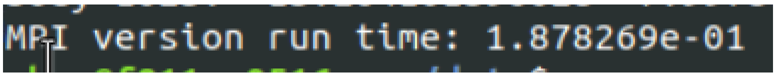
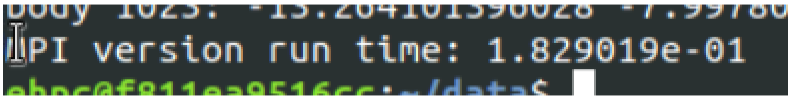
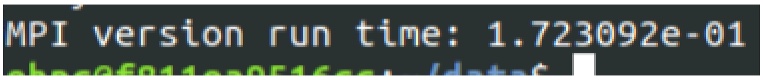
运行结果如下：

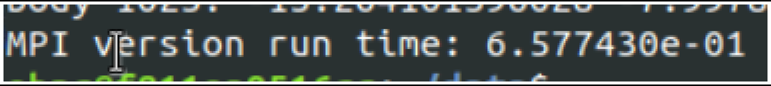
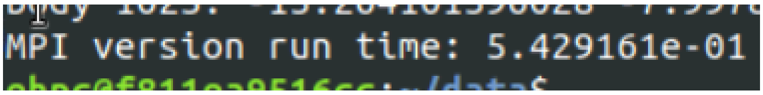
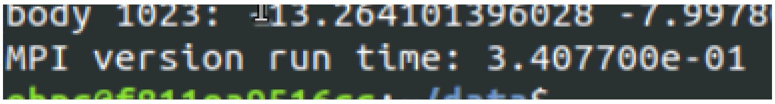


```
body  990: 16.914815884033 -13.133161012798 -37.770277560886 -3.386310561388  4.368381292154  7.802322171505
body  991: 11.672036475124 40.687206422309 -5.898989204681  3.670456159154 -8.927456917114  6.652993074001
body  992:  7.009117245411 19.071507308635 -2.234271076696 -3.037738897648 -9.971379453239  0.975120268348
body  993:  8.630916161118 15.375758491051 -1.008285652339 -2.354720105171 -8.107183022772 -0.670339491493
body  994: -9.804327249265 -30.903130249325 -13.137687700892  3.007368863249  3.089353634203  3.214524794873
body  995:  1.233752685802 -0.665403879074  7.543340746577  0.842807768918 -3.366526459642 -2.537542325624
body  996: -17.038920964294  4.575063739173 -29.280284382737  5.545275913831  0.031732703372 11.771862124872
body  997:  1.242890903370  0.326204330155 -12.558166496441 -3.961253098151 -4.106469701128  0.678871461797
body  998: 15.435235241521  5.508597639588  6.372672911763 -5.790103733540 -5.801730680770  0.024282189549
body  999:  5.607258788187 -0.528441744699  3.182173588402  3.700910061208 -1.703440342773 -9.588121717317
body 1000: -25.706096821630 -15.947655826781 -13.474546017220  9.190476760110  2.925528583293  0.070269074894
body 1001: 31.786552890985 -2.359427928834 10.364346849501 -10.770430641054  1.500707333196 -2.687829353965
body 1002: 16.867558226129 -23.033192901435  0.496560987779 -5.621524081268  5.396147393081 -0.166145895422
body 1003:  1.281731929398 -34.494684783971 10.524403991053  2.221946380916  9.239597120871 -4.206882267812
body 1004: -20.241210852839 -20.272304257861  7.037945183477  0.038062811339  5.457920743960 -6.060062360974
body 1005:  2.536894390565 -25.736205689892 28.631910257040 -4.701144058389  4.473946321828 -4.917611021003
body 1006: -11.036927079551 17.532934879594 -0.129008251064  1.652549030850 -6.302273162472 -2.029222050796
body 1007: 37.567073303271 -2.293170923048 -37.318086730104 -0.247559979549 -0.014323851942  3.634028238953
body 1008: -17.083086885786 11.130671556702 -26.943380646207  3.530544352539 -2.230538074496  6.559994612380
body 1009:  4.686735281689 13.009564693458 -2.741172443269  1.489076737324 -7.710181386072 -3.459724995962
body 1010: -10.641102823405 -11.088572974335 -17.740418243922 -1.450629477021  6.435952891323  9.228704291744
body 1011: 19.232848522118 -0.170829578429 -7.389324824099 -8.753731216920  1.561754078240  4.009304118912
body 1012: 11.799490529755 22.923207395387 19.948214928961 -6.310373823577 -9.141092494522 -7.673637037899
body 1013: -11.493701347475 -24.984047940187 20.238181879154  0.352016829620  7.859806672944 -0.306559335751
body 1014:  3.037301805226 34.062355111225  5.557652759915  0.839309046624 -11.763884603753 -12.317061181746
body 1015: 19.686018542558 -16.871055275688  6.497793777246 -8.659241177990 10.140246543598 -1.830599049169
body 1016: -66.988150918601 -3.686061026170  6.309375274496  6.757670191306  0.481424021169 -0.771982594375
body 1017: -13.526109189902 -2.076659464631 22.066560522703  5.905640113145  1.259106053502 -9.369629036794
body 1018: -2.464373875523  0.275312472929 -20.156789836832 -1.113755065150  0.556662892838  8.218039045367
body 1019: 15.510446453884 -10.187199823755 -11.446957132332 -2.623562501550  6.112416677284  4.411757921502
body 1020: -0.518861788098 -10.061494953472  7.753426572734  1.083492250342  7.581455765292 -5.456929895430
body 1021: 32.461495153109 -9.958217945346 -10.281628954598 -8.368374197087  1.788595038394  1.802374837866
body 1022: -17.035156295529 -5.704650370464 -21.706299629579  4.942932399905  4.573670335317  7.863016046896
body 1023: -13.264101396028 -7.997808717886  4.420369907957 -0.220222999988  5.439219330910  2.390775696087
MPI version run time: 5.748680e-01
```

经分析，与参考输出存在的误差<10^{-7}，与串行版本的输出完全相同。

分别以 2、4、8、16、32 个线程运行 MPI 版本的求解器，每种情况运行 3 次并计算平均运行时间，如下所示：

线程数	运行时间	平均运行时间
2	 MPI version run time: 5.748680e-01  MPI version run time: 4.868729e-01  MPI version run time: 5.754869e-01	5.457426×10^{-1}
4	 MPI version run time: 3.346241e-01  MPI version run time: 3.406661e-01  MPI version run time: 3.334908e-01	3.362603×10^{-1}
8	 MPI version run time: 1.878269e-01  MPI version run time: 1.829019e-01  MPI version run time: 1.723092e-01	1.810127×10^{-1}

16	 MPI version run time: 6.577430e-01	5.137097×10^{-1}
	 MPI version run time: 5.429161e-01	
	 body 1023: -13.264101396028 -7.9978 MPI version run time: 3.407700e-01	

上述结果的运行时间汇总如下：

线程/进程数	串行版本	OpenMP	Pthread	MPI
1	6.987495×10^{-1}	8.578487×10^{-1}	8.256597×10^{-1}	/
2	/	6.429719×10^{-1}	6.252744×10^{-1}	5.457426×10^{-1}
4	/	3.966370×10^{-1}	3.812043×10^{-1}	3.362603×10^{-1}
8	/	2.151856×10^{-1}	2.636734×10^{-1}	1.810127×10^{-1}
16	/	1.715460×10^{-1}	2.186649×10^{-1}	5.137097×10^{-1}
32	/	1.294960×10^{-1}	2.379883×10^{-1}	/

对各个并行程序，若只以 1 个线程/进程运行，运行时间将会比普通的串行程序长，OpenMP 版本的运行时间随着线程数增加而减短，Pthread 版本的运行时间随着线程数增加而减短，但当线程数从 16 增加到 32 时运行时间反而增加， MPI 版本的运行时间随着线程数增加而减短，但当进程数大于 8 时运行时间反而增加，且是十分显著的增加。

只以 1 个线程/进程运行并行程序，并行程序比串行程序多了申请空间、判断语句等，使得运行时间比串行程序要慢，在本次实验中，OpenMP 版本是在确定线程数之后才申请 loc_forces 的空间，而 Pthread 直接以最大线程数申请空间，且 Pthread 版本需要一次次的创建线程函数再调度运行，这些原因可能造成 Pthread 版本比 OpenMP 版本要慢，此外，与 Pthread 相比，OpenMP 更多的部分是有编译器来执行的，编译器所使用的算法可能比我用 Pthread 写的更高效，这也是 OpenMP 更快的一个可能原因。MPI 之间的数据共享需要通过消息传递，因为 MPI 同步的程序属于不同的进程，甚至不同的主机上的不同进程，相反由于 OpenMP 和 Pthread 共享内存，由于数据不共享，每个进程都需要存储大量的相同数据（比如 masses），我认为这是导致 MPI 版本速度受限的一大原因，当进程数为 16 时，运行时间大大增加，估计是因为分配较大的内存空间拖慢速度，此外，进程间的通信速度也是影响 MPI 版本的求解器的运行时间的一大原因。