作业七　MPI 点对点通信实现 Nbody 程序

150*******　李师尧

利用 MPI，分别用阻塞式通信、非阻塞使通信重新编码实现作业 N-body 计算问题。在同一个多处理机系统上，使用不同的数据规模，对比 MPI 并行程序与 pthread 并行程序的加速比，分析其加速比差异的原因。

1　算法分析

采用如图 1 所示流水并行的方式，设粒子数为 N，每个进程分配一个 body 和 force 数组，长度为 N，每个进程负责一小部分的粒子（loca~locb，数目是 locbodynum）与其他粒子（loca+1~）之间的作用力的计算。每个进程计算作用力之后得到了其他粒子受力的一部分，通过 MPI 点对点通信，将这部分受力加到相应的粒子上（显然，这个过程也可以通过规约操作实现），然后进行时间演化。

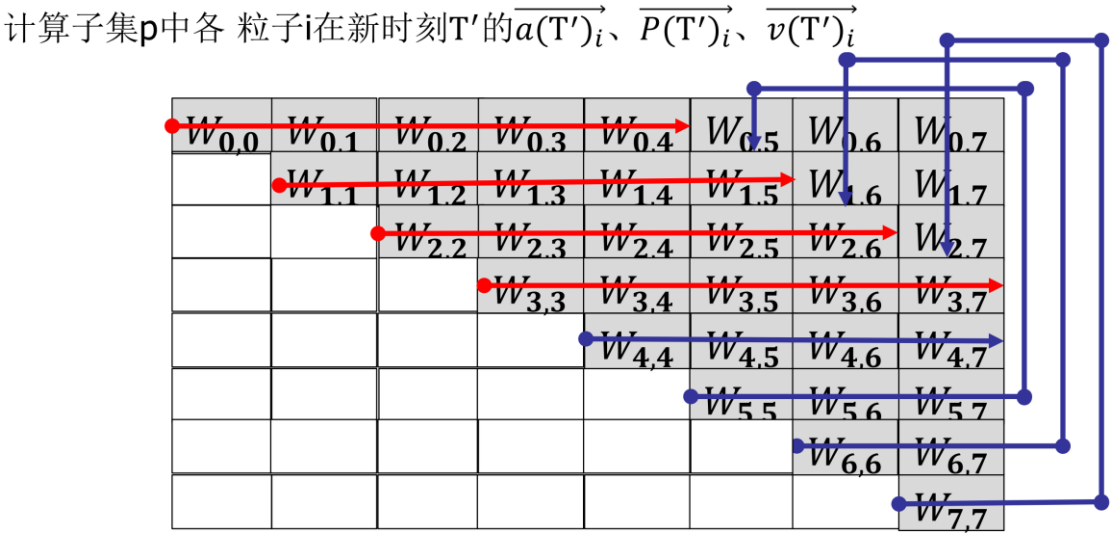

图 1　N body 问题的流水并行方式

2 结果与讨论

如图 2 所示，不同粒子数目下、不同时间步数的性能表现。空心点是 pthread 的结果，整体低于 MPI，MPI 阻塞式通信在小规模时优于非阻塞，但在大规模时，优势不在，根据趋势可知，但规模较大时，阻塞通信使得程序因通信开销大大增加而效率降低，非阻塞通信开始凸显优势。

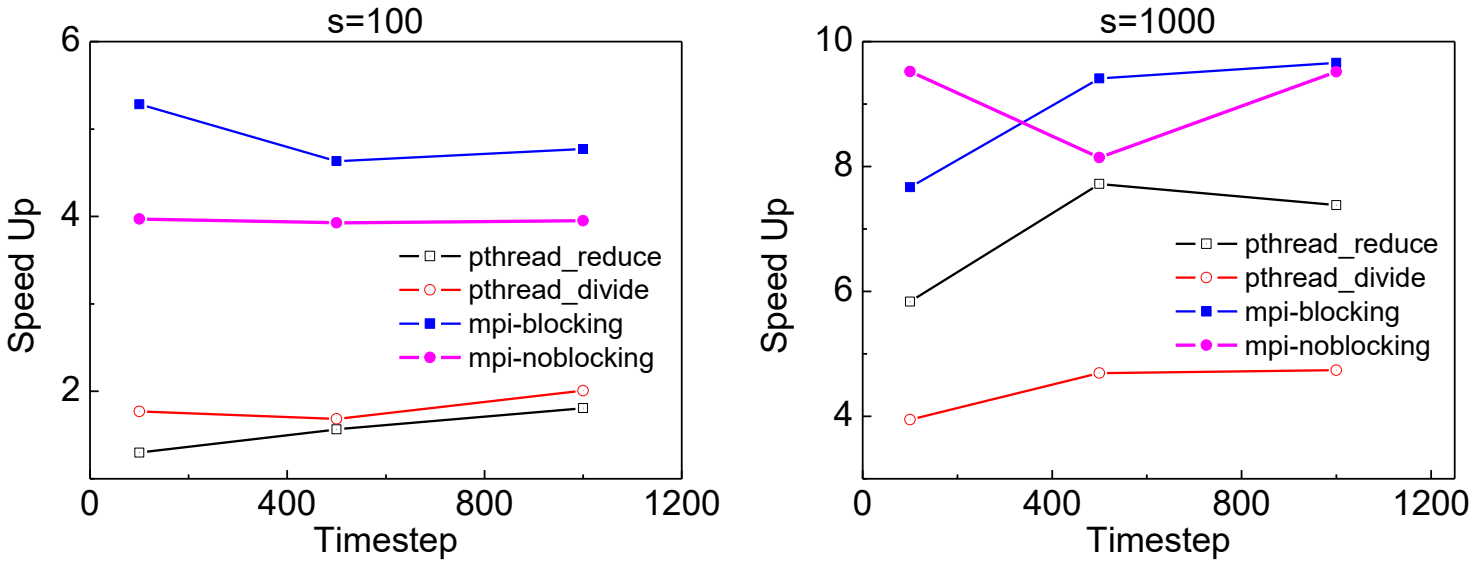

图 2 两种粒子数目、不同时间步数的加速比比较。空心点是 pthread 的实现，实心点事 MPI 的表现。

```c
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include<mpi.h>
#define NANO           1000000000
#define Max_Thread_Num 256
#define REAL           double

int BodyNum=0;
int TimeSteps=0;
REAL *body;
REAL *force;
MPI_Status status;
int myid, size;
int totalThread;
int bcastint[10];

pthread_cond_t   cond;
pthread_mutex_t  mtx;

double serial();
double mpi_nbody_blocking();
double mpi_nbody_noblocking();

double pthread_reduce();
void *reduce_worker(void *arg);
int  freeWorker;

double pthread_divide();
void *divide_worker(void *arg);
// struct STATUS {
        // int lbound;
        // int ubound;
        // int *task;
        // REAL *force;
        // pthread_mutex_t  mtx;
// }
*status;


int main(int argc, char** argv ) {
    int i;
    REAL ser_time, red_time,
    div_time,mpi_time_blocking,mpi_time_noblocking;
```

```
46      char *pStr;
47
48
49      MPI_Init( &argc, &argv );
50      MPI_Comm_rank( MPI_COMM_WORLD, &myid );
51      MPI_Comm_size( MPI_COMM_WORLD, &size );
52      if (myid==0) {
53          for ( i=1; i<argc; i++ ) {
54            pStr=strstr(argv[i], "-s=");
55            if ( pStr!=NULL) sscanf(pStr, "-s=%d", &BodyNum);
56            pStr=strstr(argv[i], "-t=");
57            if ( pStr!=NULL) sscanf(pStr, "-t=%d", &TimeSteps);
58
59          }
60          if ( BodyNum*TimeSteps==0) {
61            printf("usage: -s=number-of-bodies
              -t=number-of-steps\n");
62            return 0;
63          }
64          bcastint[0]=BodyNum;
65          bcastint[1]=TimeSteps;
66      }
67      MPI_Bcast(bcastint,2,MPI_INT,0,MPI_COMM_WORLD);
68      BodyNum=bcastint[0];
69      TimeSteps=bcastint[1];
70
71
72
73      if (myid==0) {
74          ser_time = serial();
75          printf("serial: %f\n", ser_time);
76      }
77      MPI_Barrier(MPI_COMM_WORLD);
78      mpi_time_blocking=mpi_nbody_blocking();
79      if (myid==0) printf("mpi-blocking: %f
        speedup=%f\n",mpi_time_blocking,ser_time/mpi_time_blocking);
80      mpi_time_noblocking=mpi_nbody_noblocking();
81      if (myid==0) printf("mpi-noblocking: %f
        speedup=%f\n",mpi_time_noblocking,ser_time/mpi_time_noblockin
        g);
82      free(body);
83      free(force);
84      MPI_Finalize();
85
86  }
87
88  double serial() {
89          REAL fac, fx, fy, fz;
90          REAL dx, dy, dz, sq, dist;
```

```
91          int t, i, j, bi,bj,fi,fj;
92          struct timespec ts,te;
93          double result;
94
95           body = (REAL*)malloc(4*BodyNum*sizeof(REAL));
96          /*  Initialize mass and positions in array p to make a
            test case   */
97          for ( i=0; i<BodyNum; i++)    {
98             body[4*i] = 10.05 + i;
99             body[4*i+1] = 30.0*i;
100            body[4*i+2] = 20.0*i;
101            body[4*i+3] = 10.0*i;
102         }
103
104         clock_gettime(CLOCK_REALTIME, &ts);
105
106         force = (REAL*)malloc(3*BodyNum*sizeof(REAL));
107         for ( i=0; i<3*BodyNum; i++)  force[i] = 0;
108
109         t = 0;
110         while ( t<TimeSteps){
111             /*  Loop over points calculating force between each
                pair.*/
112             for ( i=0; i<BodyNum; i++ ) {
113                bi = 4*i;
114                fi = 3*i;
115                for ( j=i+1; j<BodyNum; j++ )  {
116                   bj = 4*j;
117                   fj = 3*j;
118                   /*Calculate force between particle i and j
                   according to Newton's Law*/
119                   dx = body[bi+1] - body[bj+1];
120                   dy = body[bi+2] - body[bj+2];
121                   dz = body[bi+3] - body[bj+3];
122                   sq = dx*dx + dy*dy + dz*dz;
123                   dist = sqrt(sq);
124                   fac = body[bi] * body[bj] / ( dist * sq );
125                   fx = fac * dx;
126                   fy = fac * dy;
127                   fz = fac * dz;
128                   /*Add in force and opposite force to particle
                   i and j */
129                   force[fi] -= fx;
130                   force[fi+1] -= fy;
131                   force[fi+2] -= fz;
132                   force[fj] += fx;
133                   force[fj+1] += fy;
134                   force[fj+2] += fz;
135                }
```

```
136             }
137             for ( i=0; i<BodyNum; i++ ){
138                bi = 4*i;
139                fi = 3*i;
140                body[bi+1] = body[bi+1] + force[fi] / body[bi];
141                force[fi] = 0;
142                body[bi+2] = body[bi+2] + force[fi+1] / body[bi];
143                force[fi+1] = 0;
144                body[bi+3] = body[bi+3] + force[fi+2] / body[bi];
145                force[fi+2] = 0;
146             }
147             t++;
148          }
149          free(force);
150
151          clock_gettime(CLOCK_REALTIME, &te);
152          result = te.tv_sec - ts.tv_sec +
          (double)(te.tv_nsec-ts.tv_nsec)/NANO;
153
154          FILE *fResult=fopen("result_ser_nbody.txt", "w");
155          char str[50];
156          for (i=0; i<BodyNum; i++)   {
157         sprintf(str, "(%10.4f %10.4f %10.4f %10.4f)\n",
         body[4*i], body[4*i+1], body[4*i+2], body[4*i+3]);
158         fwrite(str, sizeof(char), strlen(str), fResult);
159          }
160          fclose(fResult);
161          return result;
162   }
163
164   double mpi_nbody_blocking() {
165          REAL fac, fx, fy, fz;
166          REAL dx, dy, dz, sq, dist;
167          int t, i, j, bi,bj,fi,fj;
168          struct timespec ts,te;
169          double result;
170           int locbodynum,loca,locb,locnum;
171           REAL *bodysend,*forcesend;
172           if (myid<size-1) {
173               locbodynum=int(BodyNum/size);
174               loca=0+myid*locbodynum;
175               locb=loca+locbodynum-1;
176           }
177           else{
178               locbodynum=BodyNum-int(BodyNum/size)*(size-1);
179               loca=0+(size-1)*int(BodyNum/size);
180               locb=BodyNum-1;
181           }
182           //printf("%d: %d %d %d \n",myid,loca, locb, locbodynum);
```

```
183          body=(REAL*)malloc(4*BodyNum*sizeof(REAL));
184          force=(REAL*)malloc(3*BodyNum*sizeof(REAL));
185        /*  Initialize mass and positions in array p to make a
           test case   */
186        for ( i=0; i<BodyNum; i++)    {
187           bi=4*i;
188           body[bi] = 10.05 + i;
189           body[bi+1] = 30.0*i;
190           body[bi+2] = 20.0*i;
191           body[bi+3] = 10.0*i;
192        }
193
194        clock_gettime(CLOCK_REALTIME, &ts);
195
196        for ( i=0; i<3*locbodynum; i++)  force[loca+i] = 0;
197
198        t = 0;
199        int j1;
200        while ( t<TimeSteps){
201            /*  Loop over points calculating force between each
               pair.*/
202            for ( i=0; i<locbodynum; i++ ) {
203               bi = loca+4*i;
204               fi = loca+3*i;
205               if (2*i<=BodyNum)
206                   locnum=int(BodyNum/2);
207               else
208                   locnum=BodyNum-int(BodyNum/2);
209               for ( j1=0; j1<locnum; j1++ )  {
210                  j=(j1+i+1)%BodyNum;
211                  bj = 4*j;
212                  fj = 3*j;
213                  /*Calculate force between particle i and j
                     according to Newton's Law*/
214                  dx = body[bi+1] - body[bj+1];
215                  dy = body[bi+2] - body[bj+2];
216                  dz = body[bi+3] - body[bj+3];
217                  sq = dx*dx + dy*dy + dz*dz;
218                  dist = sqrt(sq);
219                  fac = body[bi] * body[bj] / ( dist * sq );
220                  fx = fac * dx;
221                  fy = fac * dy;
222                  fz = fac * dz;
223                  /*Add in force and opposite force to particle
                     i and j */
224                  force[fi] -= fx;
225                  force[fi+1] -= fy;
226                  force[fi+2] -= fz;
227                  force[fj] += fx;
```

```
228                     force[fj+1] += fy;
229                     force[fj+2] += fz;
230                 }
231             }
232
233             for (i=0; i<size; i++){
234                 if (i==myid) continue;
235
                    MPI_Send(force,3*BodyNum,MPI_DOUBLE,i,myid,MPI_CO
                    MM_WORLD);
236             }
237             for (i=0; i<size; i++) {
238                 if (i==myid) continue;
239                 REAL *temp;
240                 temp=(REAL*)malloc(3*BodyNum*sizeof(REAL));
241
                    MPI_Recv(temp,3*BodyNum,MPI_DOUBLE,i,i,MPI_COMM_W
                    ORLD,&status);
242                 for (j=loca; j<=locb; j++) {
243                     fj=3*j;
244                     force[fj]+=temp[fj];
245                     force[fj+1]+=temp[fj+1];
246                     force[fj+2]+=temp[fj+2];
247                 }
248
249                 delete temp;
250             }
251
252             // REAL *temp;
253             // temp=(REAL*)malloc(3*size*BodyNum*sizeof(REAL));
254             // for (i=0; i<size; i++) {
255                 // if (i==myid) continue;
256                 //
                    MPI_Recv(temp+3*i*BodyNum,3*BodyNum,MPI_DOUBLE,i,
                    i,MPI_COMM_WORLD,&status);
257             // }
258             // for (i=0; i<size; i++) {
259                 // if (i==myid) continue;
260                 // for (j=loca; j<=locb; j++) {
261                     // fj=3*j;
262                     // fi=fj+size*i;
263                     // force[fj]+=temp[fi];
264                     // force[fj+1]+=temp[fi+1];
265                     // force[fj+2]+=temp[fi+2];
266                 // }
267             // }
268             // delete temp;
269
270             for ( i=0; i<locbodynum; i++ ){
```

```
271                 bi = 4*i;
272                 fi = 3*i;
273                 body[bi+1] = body[bi+1] + force[fi] / body[bi];
274                 force[fi] = 0;
275                 body[bi+2] = body[bi+2] + force[fi+1] / body[bi];
276                 force[fi+1] = 0;
277                 body[bi+3] = body[bi+3] + force[fi+2] / body[bi];
278                 force[fi+2] = 0;
279             }
280
281             for (i=0; i<size; i++){
282                  if (i==myid) continue;
283
                     MPI_Send(body+4*loca,4*locbodynum,MPI_DOUBLE,i,my
                     id,MPI_COMM_WORLD);
284              }
285              for (i=0; i<size; i++) {
286                  if (i==myid) continue;
287
                     MPI_Recv(body+4*locbodynum*i,4*locbodynum,MPI_DOU
                     BLE,i,i,MPI_COMM_WORLD,&status);
288              }
289
290
291             t++;
292         }
293         free(force);
294
295         clock_gettime(CLOCK_REALTIME, &te);
296         result = te.tv_sec - ts.tv_sec +
            (double)(te.tv_nsec-ts.tv_nsec)/NANO;
297
298         FILE *fResult=fopen("result_mpi_nbody.txt", "w");
299         char str[50];
300         for (i=0; i<BodyNum; i++)   {
301            sprintf(str, "(%10.4f %10.4f %10.4f %10.4f)\n",
               body[4*i], body[4*i+1], body[4*i+2], body[4*i+3]);
302            fwrite(str, sizeof(char), strlen(str), fResult);
303         }
304         fclose(fResult);
305         return result;
306   }
307
308   double mpi_nbody_noblocking() {
309         REAL fac, fx, fy, fz;
310         REAL dx, dy, dz, sq, dist;
311         int t, i, j, bi,bj,fi,fj;
312         struct timespec ts,te;
313         double result;
```

```
314         MPI_Request *request;
315         MPI_Status  *state;
316          int locbodynum,loca,locb,locnum;
317          REAL *bodysend,*forcesend;
318          if (myid<size-1) {
319              locbodynum=int(BodyNum/size);
320              loca=0+myid*locbodynum;
321              locb=loca+locbodynum-1;
322          }
323          else{
324              locbodynum=BodyNum-int(BodyNum/size)*(size-1);
325              loca=0+(size-1)*int(BodyNum/size);
326              locb=BodyNum-1;
327          }
328          //printf("%d: %d %d %d \n",myid,loca, locb, locbodynum);
329          body=(REAL*)malloc(4*BodyNum*sizeof(REAL));
330          force=(REAL*)malloc(3*BodyNum*sizeof(REAL));
331         /*  Initialize mass and positions in array p to make a
            test case   */
332         for ( i=0; i<BodyNum; i++)    {
333            bi=4*i;
334            body[bi] = 10.05 + i;
335            body[bi+1] = 30.0*i;
336            body[bi+2] = 20.0*i;
337            body[bi+3] = 10.0*i;
338         }
339
340         clock_gettime(CLOCK_REALTIME, &ts);
341
342         for ( i=0; i<3*locbodynum; i++)  force[loca+i] = 0;
343
344         t = 0;
345         int j1;
346         int req;
347         request=new MPI_Request[size*2];
348         state=new MPI_Status[size*2];
349         while ( t<TimeSteps){
350             /*  Loop over points calculating force between each
                pair.*/
351             for ( i=0; i<locbodynum; i++ ) {
352                bi = loca+4*i;
353                fi = loca+3*i;
354                if (2*i<=BodyNum)
355                    locnum=int(BodyNum/2);
356                else
357                    locnum=BodyNum-int(BodyNum/2);
358                for ( j1=0; j1<locnum; j1++ )  {
359                   j=(j1+i+1)%BodyNum;
360                   bj = 4*j;
```

```
361                     fj = 3*j;
362                     /*Calculate force between particle i and j
                        according to Newton's Law*/
363                     dx = body[bi+1] - body[bj+1];
364                     dy = body[bi+2] - body[bj+2];
365                     dz = body[bi+3] - body[bj+3];
366                     sq = dx*dx + dy*dy + dz*dz;
367                     dist = sqrt(sq);
368                     fac = body[bi] * body[bj] / ( dist * sq );
369                     fx = fac * dx;
370                     fy = fac * dy;
371                     fz = fac * dz;
372                     /*Add in force and opposite force to particle
                        i and j */
373                     force[fi] -= fx;
374                     force[fi+1] -= fy;
375                     force[fi+2] -= fz;
376                     force[fj] += fx;
377                     force[fj+1] += fy;
378                     force[fj+2] += fz;
379                   }
380                 }
381
382                 for (i=0; i<size; i++){
383                     if (i==myid) continue;
384
                        MPI_Isend(force,3*BodyNum,MPI_DOUBLE,i,myid,MPI_C
                        OMM_WORLD,&request[i]);
385                 }
386                 REAL *temp;
387                 temp=(REAL*)malloc(3*size*BodyNum*sizeof(REAL));
388                 req=0;
389                 for (i=0; i<size; i++) {
390                     if (i==myid) continue;
391
                        MPI_Irecv(temp+3*i*BodyNum,3*BodyNum,MPI_DOUBLE,i
                        ,i,MPI_COMM_WORLD,&request[size+req]);
392                     req++;
393                 }
394                 MPI_Waitall(size-1,request+size,state+size);
395                 //MPI_Barrier(MPI_COMM_WORLD);
396                 for (i=0; i<size; i++) {
397                     if (i==myid) continue;
398                     for (j=loca; j<=locb; j++) {
399                         fj=3*j;
400                         fi=fj+size*i;
401                         force[fj]+=temp[fi];
402                         force[fj+1]+=temp[fi+1];
403                         force[fj+2]+=temp[fi+2];
```

```
404                 }
405              }
406              delete temp;
407
408
409
410             for ( i=0; i<locbodynum; i++ ){
411                bi = 4*i;
412                fi = 3*i;
413                body[bi+1] = body[bi+1] + force[fi] / body[bi];
414                force[fi] = 0;
415                body[bi+2] = body[bi+2] + force[fi+1] / body[bi];
416                force[fi+1] = 0;
417                body[bi+3] = body[bi+3] + force[fi+2] / body[bi];
418                force[fi+2] = 0;
419             }
420
421             for (i=0; i<size; i++){
422                  if (i==myid) continue;
423
                     MPI_Isend(body+4*loca,4*locbodynum,MPI_DOUBLE,i,m
                     yid,MPI_COMM_WORLD,&request[i]);
424              }
425              req=0;
426              for (i=0; i<size; i++) {
427                  if (i==myid) continue;
428
                     MPI_Irecv(body+4*locbodynum*i,4*locbodynum,MPI_DO
                     UBLE,i,i,MPI_COMM_WORLD,&request[size+req]);
429                  req++;
430              }
431              MPI_Waitall(size-1,request+size,state+size);
432              //MPI_Barrier(MPI_COMM_WORLD);
433
434             t++;
435          }
436
437
438
439          clock_gettime(CLOCK_REALTIME, &te);
440          result = te.tv_sec - ts.tv_sec +
             (double)(te.tv_nsec-ts.tv_nsec)/NANO;
441
442          FILE *fResult=fopen("result_mpi_nbody.txt", "w");
443          char str[50];
444          for (i=0; i<BodyNum; i++)   {
445             sprintf(str, "(%10.4f %10.4f %10.4f %10.4f)\n",
             body[4*i], body[4*i+1], body[4*i+2], body[4*i+3]);
446             fwrite(str, sizeof(char), strlen(str), fResult);
```

```
447         }
448         fclose(fResult);
449         return result;
450 }
```