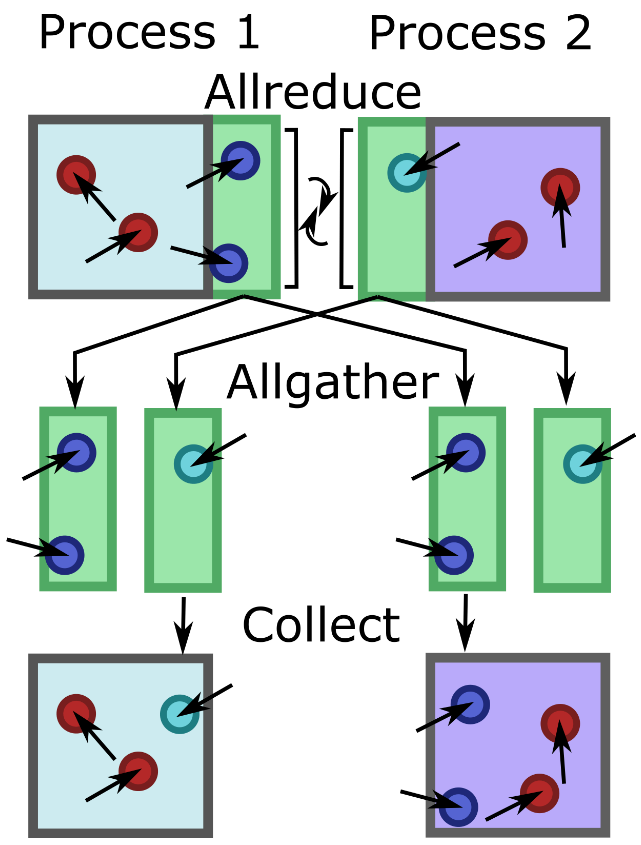
HPSC Lab 03

Ashley Pera, Varun Hoskere, Prajwal Kiran Naik

Parallel Implementation

We present a code for the simulation of the 2D N-particle problem. Newtonian iteration is performed to update the position and velocities of each particle. In order to parallelize this code, space is divided into rectangular blocks. Each block is stored on a single process, and when particles exit the edge of a block, they are transferred to the appropriate neighboring process according to the following scheme:

1. If the particle is outside the block, add it to a list of IDs of outgoing particles along with the destination process.
2. Remove it from the list of active particles on the current process.
3. Find the maximum number of outgoing particles using an allreduce.
4. Create memory on each processor to hold particle data for that many particles.
5. Fill these with the appropriate particle data.
6. Allgather these outgoing particle lists, so each process has all the particles which crossed a cell boundary.
7. Collect all the particles in this gathered list with the destination process.
8. Add these particles to active simulation in the block.



Self-Evaluation

Initially, the plot we saw had particles flowing smoothly across all cells, except those assigned to the top right corner processor (iPE = 1, jPE = 1). The particles completely disappeared when they moved into that processor's cells. To debug this, we ran it with cells only along the x-x-direction and saw no issues. We ran it vice-versa too, with cells only along the y-direction. This didn’t help us solve our issue either.

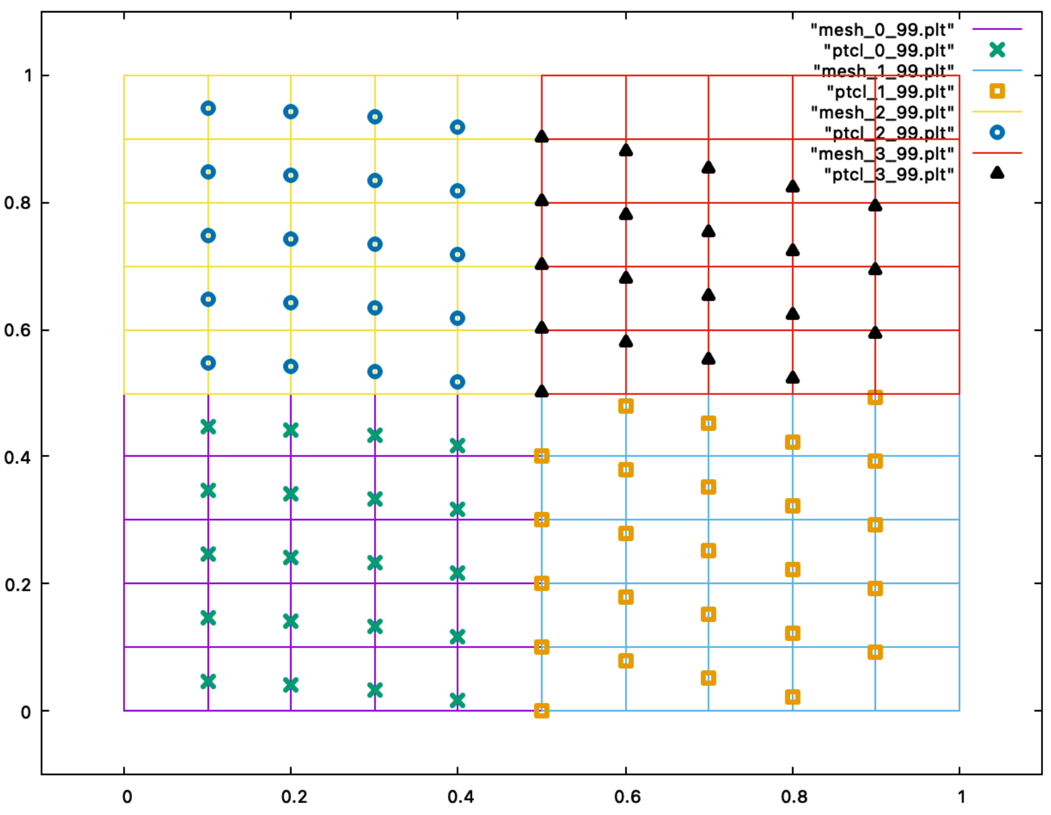
After logical deduction, we concurred that the issue had to do with the calculation of the destination processor - we just weren’t sure what the issue was. After consulting with Dr. Runnels, we were able to pinpoint the bug. We initially used nRealx for the new processor computation. nRealx is associated with the number of cells in the mesh, and not the number of processors. After changing that to nPEx, which is the number of processors in the x-direction, our plot straightened out.

In the plot we can see that the N-particle system flows smoothly across the display. Every so often, some particles move from one set of colored cells to the other, which corresponds to particles being sent from one processor to another. Since these particles are under the influence of gravity, they also fall vertically throughout the plot.

Appendix A : Code

```
void ParticleExchange( VI &ptcl_send_list , VI &ptcl_send_PE , particles &PTCL)
 {
   MPI_Status  status;
   MPI_Request request;

   // (1) Get the max number particles to be sent by any particular processor, and make sure 
all processors  know that number.

   int numToSend = ptcl_send_list.size();      int maxToSend;

   MPI_Iallreduce( &numToSend , &maxToSend , 1 , MPI_INT, MPI_MAX , MPI_COMM_WORLD, &request); 
MPI_Wait(&request,&status);

   // (2) Allocate contributions to the upcoming Gather operation.  Here, "C" for 
"Contribution" to be Gathered

   int    *Cptcl_PE;  Cptcl_PE = new int    [  maxToSend ];  // Particles' destination PEs
   double *Cptcl_x ;  Cptcl_x  = new double [  maxToSend ];
   double *Cptcl_y ;  Cptcl_y  = new double [  maxToSend ];
   double *Cptcl_vx;  Cptcl_vx = new double [  maxToSend ];
   double *Cptcl_vy;  Cptcl_vy = new double [  maxToSend ];

   // (3) Populate contributions on all processors for the upcoming Gather operation

   for ( int i = 0 ; i < maxToSend ; ++i ) { Cptcl_PE[i] = -1; Cptcl_x [i] = 0.; Cptcl_y [i] = 
0.; Cptcl_vx[i] = 0.; Cptcl_vy[i] = 0.; }


   // (4) Populate with all the particles on this PE.  Note that some/most processors will 
have left-over space in the C* arrays.

   for ( int i = 0 ; i < ptcl_send_list.size() ; ++i )
     {
       int id       = ptcl_send_list[ i ];
       Cptcl_PE[i] = ptcl_send_PE   [ i ];
       Cptcl_x [i] = PTCL.x         [ id ];
       Cptcl_y [i] = PTCL.y         [ id ];
       Cptcl_vx[i] = PTCL.vx        [ id ];
       Cptcl_vy[i] = PTCL.vy        [ id ];
     }

   // (5) Allocate and initialize the arrays for upcoming Gather operation to PE0.  The 
sizeOfGather takes
   //     into account the number of processors, like this figure:
   //
   //     |<------------------------------ sizeOfGather ------------------------------->|
   //     |                                                                             |
   //     |                                                                             |
   //     |<- maxToSend    ->|<- maxToSend    ->|<- maxToSend    ->|<- maxToSend    ->|
   //     +-------------------+-------------------+-------------------+-------------------+
   //             PE0                PE1                PE2                PE3


   int sizeOfGather = maxToSend * numPE;

   int    *Gptcl_PE;  Gptcl_PE = new int    [ sizeOfGather ];
   double *Gptcl_x ;  Gptcl_x  = new double [ sizeOfGather ];
   double *Gptcl_y ;  Gptcl_y  = new double [ sizeOfGather ];
   double *Gptcl_vx;  Gptcl_vx = new double [ sizeOfGather ];
   double *Gptcl_vy;  Gptcl_vy = new double [ sizeOfGather ];

   for ( int i = 0 ; i < sizeOfGather ; ++i ) { Gptcl_PE[i] = -1; Gptcl_x [i] = 0.; Gptcl_y 
[i] = 0.; Gptcl_vx[i] = 0.; Gptcl_vy[i] = 0.;  }


   // (6)  Gather "Contributions" ("C" arrays) from all PEs onto all PEs into these bigger 
arrays so all PE will know what particles
   //      need to go where.
   MPI_Barrier(MPI_COMM_WORLD);

   MPI_Iallgather( Cptcl_PE, maxToSend, MPI_INT, Gptcl_PE, maxToSend, MPI_INT, MPI_COMM_WORLD,
&request);  MPI_Wait(&request,&status);
   MPI_Iallgather( Cptcl_x, maxToSend, MPI_DOUBLE, Gptcl_x, maxToSend, MPI_DOUBLE,
MPI_COMM_WORLD, &request);  MPI_Wait(&request,&status);
   MPI_Iallgather( Cptcl_y, maxToSend, MPI_DOUBLE, Gptcl_y, maxToSend, MPI_DOUBLE,
MPI_COMM_WORLD,  &request);  MPI_Wait(&request,&status);
   MPI_Iallgather( Cptcl_vx, maxToSend, MPI_DOUBLE, Gptcl_vx, maxToSend, MPI_DOUBLE,
MPI_COMM_WORLD,  &request);  MPI_Wait(&request,&status);
   MPI_Iallgather( Cptcl_vy, maxToSend, MPI_DOUBLE, Gptcl_vy, maxToSend, MPI_DOUBLE,
MPI_COMM_WORLD,  &request);  MPI_Wait(&request,&status);


   MPI_Barrier(MPI_COMM_WORLD);

   // (7) Put in vector form so they can be added to PTCL.  These arrays are 1-based.

   int Np = 0;  for ( int i = 0 ; i < sizeOfGather ; ++i ) if ( Gptcl_PE[i] == myPE ) ++Np;

   VD  std_add_x  ;  std_add_x.resize  ( Np+1 );
   VD  std_add_y  ;  std_add_y.resize  ( Np+1 );
   VD  std_add_vx ;  std_add_vx.resize ( Np+1 );
   VD  std_add_vy ;  std_add_vy.resize ( Np+1 );

   int count = 1;
   for ( int i = 0 ; i < sizeOfGather ; ++i )
     if ( Gptcl_PE[i] == myPE )
     {
       std_add_x [ count ] = Gptcl_x[i];
       std_add_y [ count ] = Gptcl_y [i];
       std_add_vx[ count ] = Gptcl_vx[i];
       std_add_vy[ count ] = Gptcl_vy[i];
       count++;
     }

   PTCL.add(std_add_x, std_add_y, std_add_vx, std_add_vy);

   // (8) Free up memory

   if (maxToSend    > 0 ) { delete[] Cptcl_PE;  delete[] Cptcl_x ;  delete[] Cptcl_y ;
delete[] Cptcl_vx ; delete[] Cptcl_vy;  }
   if (sizeOfGather > 0 ) { delete[] Gptcl_PE;  delete[] Gptcl_x ;  delete[] Gptcl_y ;
delete[] Gptcl_vx ; delete[] Gptcl_vy;  }

 }
```

Appendix B: Resulting plot


"mesh_0_99.plt"
"ptcl_0_99.plt"
"mesh_1_99.plt"
"ptcl_1_99.plt"
"mesh_2_99.plt"
"ptcl_2_99.plt"
"mesh_3_99.plt"
"ptcl_3_99.plt"