CSCI_596_HW4
USC ID: 4612363893
Student Name: Cancan Hua

(Part I—Asynchronous Messages)

Code modified in *atom_copy* method:

```
/* Send & receive the # of boundary atoms----------------------*/

nsd = lsb[ku][0]; /* # of atoms to be sent */

  MPI_Irecv(&nrc,1,MPI_INT,MPI_ANY_SOURCE,10,
            MPI_COMM_WORLD,&request);
  MPI_Send(&nsd,1,MPI_INT,inode,10,MPI_COMM_WORLD);
  MPI_Wait(&request, &status);

/* Now nrc is the # of atoms to be received */

/* Send & receive information on boundary atoms----------------*/


  MPI_Irecv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,
            MPI_COMM_WORLD,&request);

  /* Message buffering */
  for (i=1; i<=nsd; i++)
      for (a=0; a<3; a++) /* Shift the coordinate origin */
      dbuf[3*(i-1)+a] = r[lsb[ku][i]][a]-sv[ku][a];

  MPI_Send(dbuf,3*nsd,MPI_DOUBLE,inode,20,MPI_COMM_WORLD);
  MPI_Wait(&request, &status);
```

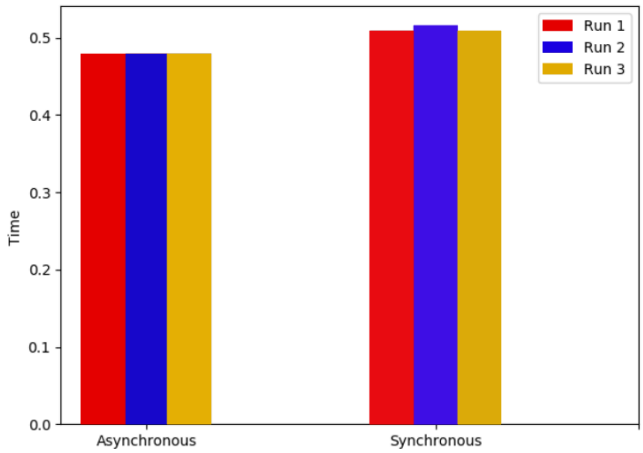
Code modified in *atom_move* method:

```
/* Send atom-number information---------------------------------*/

nsd = mvque[ku][0]; /* # of atoms to-be-sent */
  MPI_Irecv(&nrc,1,MPI_INT,MPI_ANY_SOURCE,110,
            MPI_COMM_WORLD,&request);
  MPI_Send(&nsd,1,MPI_INT,inode,110,MPI_COMM_WORLD);
  MPI_Wait(&request, &status);

/* Now nrc is the # of atoms to be received */

/* Send & receive information on boundary atoms-----------------*/

  MPI_Irecv(dbufr,6*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,120,
            MPI_COMM_WORLD,&request);

  /* Message buffering */
  for (i=1; i<=nsd; i++)
      for (a=0; a<3; a++) {
      /* Shift the coordinate origin */
      dbuf[6*(i-1)  +a] = r [mvque[ku][i]][a]-sv[ku][a];
      dbuf[6*(i-1)+3+a] = rv[mvque[ku][i]][a];
      r[mvque[ku][i]][0] = MOVED_OUT; /* Mark the moved-out atom */
      }

  MPI_Send(dbuf,6*nsd,MPI_DOUBLE,inode,120,MPI_COMM_WORLD);
  MPI_Wait(&request, &status);
```

Result:

```
=========================================
***** Asynchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 5.042868e-01 1.189794e-01
***** Synchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 4.960647e-01 1.582093e-01
***** Asynchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 4.861809e-01 1.162938e-01
***** Synchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 4.905210e-01 1.523095e-01
***** Asynchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 4.823616e-01 1.203581e-01
***** Synchronous *****
al = 5.129928e+00 5.129928e+00 5.129928e+00
lc = 2 2 2
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 1728
CPU & COMT = 5.005948e-01 1.493154e-01
```

Comparison: We can see that asynchronous is faster.



(Part II—Communicators)

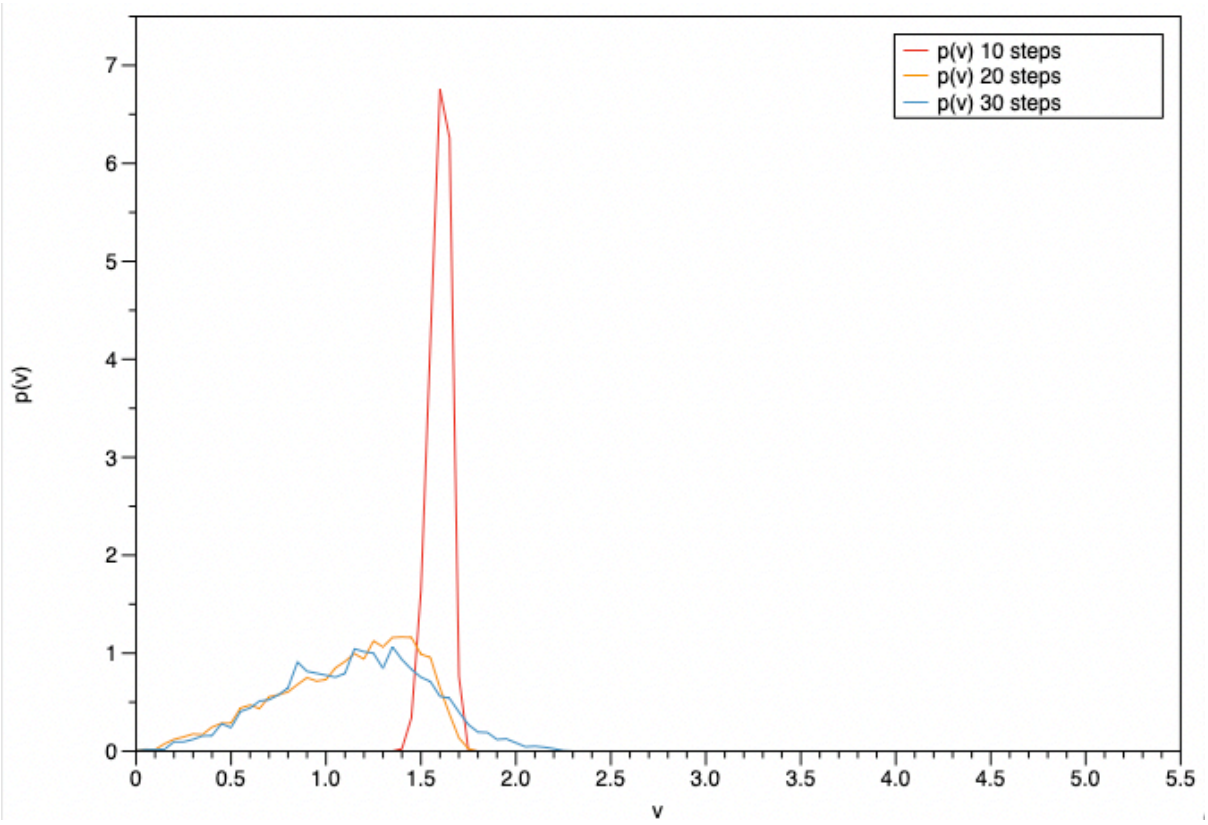
Code modified in pmd split.c

```c
#include "pmd_split.h"

void calc_pv() {
  double lpv[NBIN],pv[NBIN],dv,v;
  int i;

  // Each MPI rank computes local probability density function (PDF), lpv
  dv = VMAX/NBIN;  // Bin size
  for (i=0; i<NBIN; i++) lpv[i] = 0.0; // Reset local histogram
  for (i=0; i<n; i++) {
    v = sqrt(pow(rv[i][0],2)+pow(rv[i][1],2)+pow(rv[i][2],2));
    lpv[v/dv < NBIN ? (int)(v/dv) : NBIN-1] += 1.0;
  }
  // Global sum to obtain global PDF, pv
  MPI_Allreduce(lpv,pv,NBIN,MPI_DOUBLE,MPI_SUM,workers);
  MPI_Allreduce(&n,&nglob,1,MPI_INT,MPI_SUM,workers);  // Get global # of atoms, nglob
  for (i=0; i<NBIN; i++) pv[i] /= (dv*nglob);  // Normalization
  if (sid == 0) {
    for (i=0; i<NBIN; i++) fprintf(fpv,"%le %le\n",i*dv,pv[i]);
    fprintf(fpv,"\n");
  }
}
```

```c
/*--------------------------------------------------------------------------*/
int main(int argc, char **argv)
{
    /*--------------------------------------------------------------------------*/
    double cpu1;

    MPI_Init(&argc, &argv);              /* Initialize the MPI environment */
    // MPI_Comm_rank(MPI_COMM_WORLD, &sid); /* My processor ID */
    MPI_Comm_rank(MPI_COMM_WORLD,&gid); //Global rank
    md = gid%2; // = 1 (MD workers) or 0 (analysis workers)
    MPI_Comm_split(MPI_COMM_WORLD,md,0,&workers);
    MPI_Comm_rank(workers,&sid); // Rank in workers


    /* Vector index of this processor */
    vid[0] = sid / (vproc[1] * vproc[2]);
    vid[1] = (sid / vproc[2]) % vproc[1];
    vid[2] = sid % vproc[2];

    init_params();
    if(md) {
        set_topology();
        init_conf();
        atom_copy();
        compute_accel(); /* Computes initial accelerations */
    }
    else
        if (sid == 0) fpv = fopen("pv.dat", "w");


    cpu1 = MPI_Wtime();
    for (stepCount = 1; stepCount <= StepLimit; stepCount++)
    {
        if (md) single_step();
        if (stepCount % StepAvg == 0) {
            if (md) {
                // Send # of atoms, n, to rank gid-1 in MPI_COMM_WORLD
                MPI_Send(&n, 1, MPI_INT,gid-1, 1000, MPI_COMM_WORLD);
                // Send velocities of n atoms to rank gid-1 in MPI_COMM_WORLD
                for (int i = 0; i < n; i++)
                    for (int a = 0; a < 3; a++)
                        dbuf[3*i+a] = rv[i][a];
                MPI_Send(dbuf, 3*n, MPI_DOUBLE, gid-1, 2000, MPI_COMM_WORLD);
                eval_props();
```

```
        compute_accel(); /* Computes initial accelerations */
    }
    else
        if (sid == 0) fpv = fopen("pv.dat", "w");


    cpu1 = MPI_Wtime();
    for (stepCount = 1; stepCount <= StepLimit; stepCount++)
    {
        if (md) single_step();
        if (stepCount % StepAvg == 0) {
            if (md) {
                // Send # of atoms, n, to rank gid-1 in MPI_COMM_WORLD
                MPI_Send(&n, 1, MPI_INT,gid-1, 1000, MPI_COMM_WORLD);
                // Send velocities of n atoms to rank gid-1 in MPI_COMM_WORLD
                for (int i = 0; i < n; i++)
                    for (int a = 0; a < 3; a++)
                        dbuf[3*i+a] = rv[i][a];
                MPI_Send(dbuf, 3*n, MPI_DOUBLE, gid-1, 2000, MPI_COMM_WORLD);
                eval_props();
            }
            else {
                // Receive # of atoms, n, from rank gid+1 in MPI_COMM_WORLD
                MPI_Recv(&n, 1, MPI_INT, gid+1, 1000, MPI_COMM_WORLD, &status);
                // Receive velocities of n atoms from rank gid+1 in MPI_COMM_WORLD
                MPI_Recv(dbufr, 3*n, MPI_DOUBLE, gid+1, 2000, MPI_COMM_WORLD, &status);
                for (int i = 0; i < n; i++)
                    for (int a = 0; a < 3; a++)
                        rv[i][a] = dbufr[3*i+a];
                calc_pv();
            }
        }
    }
    cpu = MPI_Wtime() - cpu1;
    if (md && sid == 0)
        printf("CPU & COMT = %le %le\n",cpu,comt);
    if (!md && sid == 0)
        fclose(fpv);

    MPI_Finalize(); /* Clean up the MPI environment */
    return 0;
}
```

Result:


p(v) 10 steps
p(v) 20 steps
p(v) 30 steps
p(v)
v
0
1
2
3
4
5
6
7
0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5