CSCI 596: HW 4

M. Oneeb H. Khan

October 1, 2021

1 Asynchronous Messaging

1.1 Source code of pmd_irecv.c

The text below is the code for the modifications made to the *pmd_irecv.c* file. To be precise the changes made were in the functions *atom_copy* and *atom_move*. The first code block shows the changes made in *atom_copy*.

```
void atom_copy()
{
  /*-------------------------------------------------------------------------
PREVIOUS CODE NOT SHOWN HERE
-------------------------------------------------------------------------*/

      // INFORMATION ABOUT NUMBER OF ATOMS TO BE SENT
      /* Even node: send & recv */
      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);

/*-------------------------------------------------------------------------
REMAINDER CODE NOT SHOWN HERE
-------------------------------------------------------------------------*/
}
```

The second code block shows the changes made in *atom_move*.

```
void atom_move()
{
  /*-------------------------------------------------------------------------
PREVIOUS CODE NOT SHOWN HERE
-------------------------------------------------------------------------*/
```

```
6
7         MPI_Irecv(&nrc, 1, MPI_INT, MPI_ANY_SOURCE, 110,
8               MPI_COMM_WORLD, &request);
9         MPI_Send(&nsd, 1, MPI_INT, inode, 110, MPI_COMM_WORLD);
10        MPI_Wait(&request, &status);
11
12        /* Now nrc is the # of atoms to be received */
13
14        /* Send & receive information on boundary atoms------------------*/
15
16        MPI_Irecv(dbufr, 6 * nrc, MPI_DOUBLE, MPI_ANY_SOURCE, 120,
17              MPI_COMM_WORLD, &request);
18        /* Message buffering */
19        for (i = 1; i <= nsd; i++)
20          for (a = 0; a < 3; a++)
21          {
22            /* Shift the coordinate origin */
23            dbuf[6 * (i - 1) + a] = r[mvque[ku][i]][a] - sv[ku][a];
24            dbuf[6 * (i - 1) + 3 + a] = rv[mvque[ku][i]][a];
25            r[mvque[ku][i]][0] = MOVED_OUT; /* Mark the moved-out atom */
26          }
27        MPI_Send(dbuf, 6 * nsd, MPI_DOUBLE, inode, 120, MPI_COMM_WORLD);
28        MPI_Wait(&request, &status);
29
30  /*------------------------------------------------------------------------
31  REMAINDER CODE NOT SHOWN HERE
32  ------------------------------------------------------------------------*/
33  }
```

1.2 Printout

The figure below shows the printout of running the *pmd_irecv.sl* script.

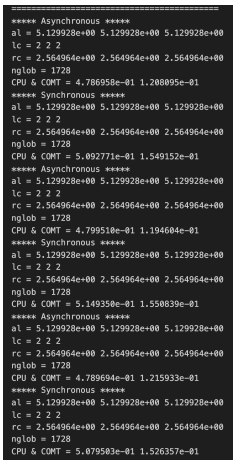

Figure 1: Printout of pmd_irecv.sl

1.3 Timing Plots

The figure below shows the plot of the timing data of three runs each for Asynchronous and Synchronous message passing. We can clearly see that the asynchronous message passing is faster than the synchronous approach.

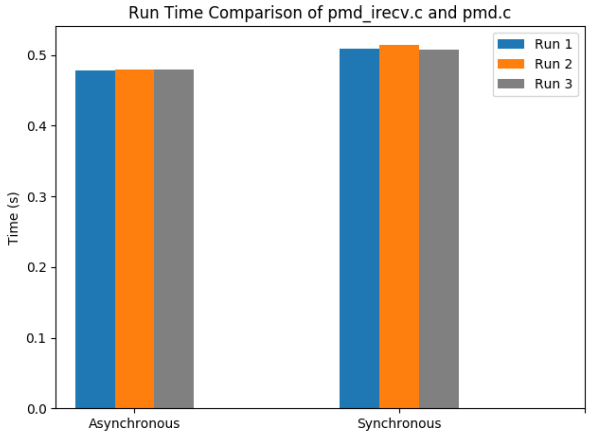

Figure 2: Timing Plot of Asynchronous vs. Synchronous message passing

2 Communicators

2.1 Source code of pmd_split.c

The text below is the code for the modifications made to the *pmd_split.c* file. To be precise the changes made were in the *main* function and the *calc_pv* function was copied from *calc_pv.c*. We also changed all instances of *MPI_COMM_WORLD* after the main function to *workers*.

```
#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");
```

```
  }
}

/*--------------------------------------------------------------------------*/
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();
      }
      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();
      }
    }
  }
```

```
81    cpu = MPI_Wtime() - cpu1;
82    if (md && sid == 0)
83      printf("CPU & COMT = %le %le\n",cpu,comt);
84    if (!md && sid == 0)
85      fclose(fpv);
86
87    MPI_Finalize(); /* Clean up the MPI environment */
88    return 0;
89  }
90
91  /*----------------------------------------------------------------------
92  REMAINDER CODE NOT SHOWN HERE
93  ----------------------------------------------------------------------*/
```

2.2 Plot of calculated PDFs at time step 10, 20 and 30

The figure below shows the plot of the Probability Density Functions calculated of the velocity with different step sizes.

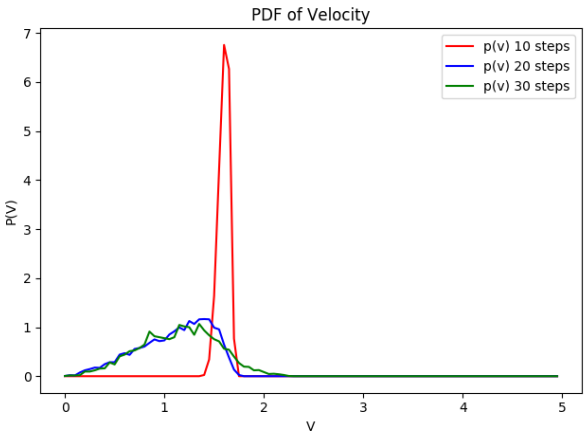

Figure 3: Calculated PDFs at time step 10, 20 and 30