Assignment 3- Parallel Molecular Dynamics
CSCI 596: Scientific Computing & Visualization

Anup V Kanale

October 4, 2017

This purpose of this assignment is run molecular dynamics on multiple processors to understand asynchronous message passing and "in-situ" data analysis.

1 Asynchronous Messages

In this section, we use `MPI_Irecv()` and `MPI_Send()` with an `MPI_Wait()` in the program `pmd.c`. The asynchronous messages make the deadlock-avoidance scheme unnecessary, and thus there is no need to use different orders of send and receive calls for even and odd-parity processes.

The new asynchronous implementation takes $5.13e-01$ seconds, whereas the original implementation takes $6.28e-01$ seconds. This observed speedup is $\approx 20\%$. The major modifications were made in the functions `atom_copy` and `atom_move`. To keep this report short, only these parts are shown in Appendix A.1.

2 Communicators

In this section, we basically allocate half the processors for the computations, and the other half for analysing the data from the computations. Specifically, following the lecture note on "In situ analysis of molecular dynamics simulation data using communicators", `pmd.c` is modified such that as many number of processes as that for MD simulations are spawned to calculate the probability density function (PDF) for the atomic velocity.

The plot of the probability density function is as shown below: The major modification was in

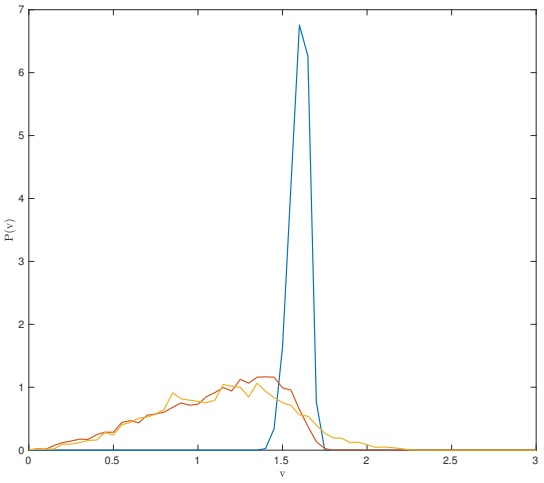

Figure 1: Probability density function

the `main` function, `calc_pv()` was provided. Only the modified portion is shown in appendix A.2 to keep this report short.

A Appendix

A.1 Task I

```
1 /*-------------------------------------------------------------------*/
2 void atom_copy() {
3 /*-------------------------------------------------------------------
4 Exchanges boundary-atom coordinates among neighbor nodes:  Makes
5 boundary-atom list, LSB, then sends & receives boundary atoms.
6 -------------------------------------------------------------------*/
7   int kd,kdd,i,ku,inode,nsd,nrc,a;
8   int nbnew = 0; /* # of "received" boundary atoms */
9   double com1;
10
11 /* Main loop over x, y & z directions
      starts-----------------------------*/
12
13   for (kd=0; kd<3; kd++) {
14
15     /* Make a boundary-atom list,
          LSB-------------------------------------*/
16
17     /* Reset the # of to-be-copied atoms for lower&higher directions */
18     for (kdd=0; kdd<2; kdd++) lsb[2*kd+kdd][0] = 0;
19
20     /* Scan all the residents & copies to identify boundary atoms */
21     for (i=0; i<n+nbnew; i++) {
22       for (kdd=0; kdd<2; kdd++) {
23         ku = 2*kd+kdd; /* Neighbor ID */
24         /* Add an atom to the boundary-atom list, LSB, for neighbor ku
25            according to bit-condition function, bbd */
26         if (bbd(r[i],ku)) lsb[ku][++(lsb[ku][0])] = i;
27       }
28     }
29
30     /* Message
          passing---------------------------------------------------*/
31
32     com1=MPI_Wtime(); /* To calculate the communication time */
33
34     /* Loop over the lower & higher directions */
35     for (kdd=0; kdd<2; kdd++) {
36
37       inode = nn[ku=2*kd+kdd]; /* Neighbor node ID */
```

```
38
39        /* Send & receive the # of boundary
              atoms------------------------*/
40
41        nsd = lsb[ku][0]; /* # of atoms to be sent */
42
43    //--START
          MODIFIED-----------------------------------------------------------------
44
45        MPI_Irecv(&nrc,1,MPI_INT,MPI_ANY_SOURCE,10,MPI_COMM_WORLD,&Request);
46        MPI_Send(&nsd,1,MPI_INT,inode,10,MPI_COMM_WORLD);
47        MPI_Wait(&Request, &status);
48
49        /* Send & receive information on boundary
              atoms--------------------*/
50        MPI_Irecv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,MPI_COMM_WORLD,&Request);
51
52        /* Message buffering */
53        for (i=1; i<=nsd; i++)
54          for (a=0; a<3; a++) /* Shift the coordinate origin */
55            dbuf[3*(i-1)+a] = r[lsb[ku][i]][a]-sv[ku][a];
56
57        MPI_Send(dbuf,3*nsd,MPI_DOUBLE,inode,20,MPI_COMM_WORLD);
58        MPI_Wait(&Request, &status);
59    //--END
          MODIFIED-----------------------------------------------------------------
60
61
62        /* Message storing */
63        for (i=0; i<nrc; i++)
64          for (a=0; a<3; a++) r[n+nbnew+i][a] = dbufr[3*i+a];
65
66        /* Increment the # of received boundary atoms */
67        nbnew = nbnew+nrc;
68
69        /* Internode synchronization */
70        MPI_Barrier(MPI_COMM_WORLD);
71
72      } /* Endfor lower & higher directions, kdd */
73
74      comt += MPI_Wtime()-com1; /* Update communication time, COMT */
75
76    } /* Endfor x, y & z directions, kd */
77
78    /* Main loop over x, y & z directions
          ends-------------------------------*/
79
80    /* Update the # of received boundary atoms */
```

```
81    nb = nbnew;
82  }
83
84
85  /*-------------------------------------------------------------------*/
86  void atom_move() {
87  /*-------------------------------------------------------------------
88  Sends moved-out atoms to neighbor nodes and receives moved-in atoms
89  from neighbor nodes.  Called with n, r[0:n-1] & rv[0:n-1], atom_move
90  returns a new n' together with r[0:n'-1] & rv[0:n'-1].
91  -------------------------------------------------------------------*/
92
93  /* Local
       variables ---------------------------------------------------*/
94
95  mvque[6][NBMAX]: mvque[ku][0] is the # of to-be-moved atoms to neighbor
96    ku; MVQUE[ku][k>0] is the atom ID, used in r, of the k-th atom to be
97    moved.
98  -------------------------------------------------------------------*/
99    int mvque[6][NBMAX];
100   int newim = 0; /* # of new immigrants */
101   int ku,kd,i,kdd,kul,kuh,inode,ipt,a,nsd,nrc;
102   double com1;
103
104   /* Reset the # of to-be-moved atoms, MVQUE[][][0] */
105   for (ku=0; ku<6; ku++) mvque[ku][0] = 0;
106
107   /* Main loop over x, y & z directions
        starts ------------------------*/
108
109   for (kd=0; kd<3; kd++) {
110
111     /* Make a moved-atom list,
          mvque ---------------------------------------*/
112
113     /* Scan all the residents & immigrants to list moved-out atoms */
114     for (i=0; i<n+newim; i++) {
115       kul = 2*kd  ; /* Neighbor ID */
116       kuh = 2*kd+1;
117       /* Register a to-be-copied atom in mvque[kul/kuh][][] */
118       if (r[i][0] > MOVED_OUT) { /* Don't scan moved-out atoms */
119         /* Move to the lower direction */
120         if (bmv(r[i],kul)) mvque[kul][++(mvque[kul][0])] = i;
121         /* Move to the higher direction */
122         else if (bmv(r[i],kuh)) mvque[kuh][++(mvque[kuh][0])] = i;
123       }
124     }
125
```

```
126     /* Message passing with neighbor
           nodes---------------------------------*/
127
128     com1 = MPI_Wtime();
129
130     /* Loop over the lower & higher
           directions-----------------------------*/
131
132     for (kdd=0; kdd<2; kdd++) {
133
134       inode = nn[ku=2*kd+kdd]; /* Neighbor node ID */
135
136       /* Send atom-number
             information-----------------------------------------*/
137
138       nsd = mvque[ku][0]; /* # of atoms to-be-sent */
139
140   //--START
        MODIFIED-------------------------------------------------------------------
141       /* Send & receive information on boundary
             atoms-------------------*/
142       MPI_Irecv(&nrc,1,MPI_INT,MPI_ANY_SOURCE,110,MPI_COMM_WORLD,&Request);
143       MPI_Send(&nsd,1,MPI_INT,inode,110,MPI_COMM_WORLD);
144       MPI_Wait(&Request, &status);
145
146       MPI_Irecv(dbufr,6*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,120,MPI_COMM_WORLD,&Request);
147
148       /* Message buffering */
149       for (i=1; i<=nsd; i++)
150         for (a=0; a<3; a++) {
151           /* Shift the coordinate origin */
152           dbuf[6*(i-1)  +a] = r [mvque[ku][i]][a]-sv[ku][a];
153           dbuf[6*(i-1)+3+a] = rv[mvque[ku][i]][a];
154           r[mvque[ku][i]][0] = MOVED_OUT; /* Mark the moved-out atom */
155         }
156
157       MPI_Send(dbuf,6*nsd,MPI_DOUBLE,inode,120,MPI_COMM_WORLD);
158       MPI_Wait(&Request, &status);
159   //--END
        MODIFIED-------------------------------------------------------------------
160
161
162       /* Message storing */
163       for (i=0; i<nrc; i++)
164         for (a=0; a<3; a++) {
165           r [n+newim+i][a] = dbufr[6*i  +a];
166           rv[n+newim+i][a] = dbufr[6*i+3+a];
167         }
```

```
168
169         /* Increment the # of new immigrants */
170         newim = newim+nrc;
171
172         /* Internode synchronization */
173         MPI_Barrier(MPI_COMM_WORLD);
174
175       } /* Endfor lower & higher directions, kdd */
176
177       comt=comt+MPI_Wtime()-com1;
178
179     } /* Endfor x, y & z directions, kd */
180
181     /* Main loop over x, y & z directions
           ends------------------------------*/
182
183     /* Compress resident arrays including new immigrants */
184
185     ipt = 0;
186     for (i=0; i<n+newim; i++) {
187       if (r[i][0] > MOVED_OUT) {
188         for (a=0; a<3; a++) {
189           r[ipt][a] = r[i][a];
190           rv[ipt][a] = rv[i][a];
191         }
192         ++ipt;
193       }
194     }
195
196     /* Update the compressed # of resident atoms */
197     n = ipt;
198 }
199
200 /*-------------------------------------------------------------------
201 Bit condition functions:
202
203 1. bbd(ri,ku) is .true. if coordinate ri[3] is in the boundary to
204       neighbor ku.
205 2. bmv(ri,ku) is .true. if an atom with coordinate ri[3] has moved out
206       to neighbor ku.
207 -------------------------------------------------------------------*/
208 int bbd(double* ri, int ku) {
209   int kd,kdd;
210   kd = ku/2; /* x(0)|y(1)|z(2) direction */
211   kdd = ku%2; /* Lower(0)|higher(1) direction */
212   if (kdd == 0)
213     return ri[kd] < RCUT;
214   else
```

```
215     return al[kd]-RCUT < ri[kd];
216 }
217 int bmv(double* ri, int ku) {
218   int kd,kdd;
219   kd = ku/2; /* x(0)|y(1)|z(2) direction */
220   kdd = ku%2; /* Lower(0)|higher(1) direction */
221   if (kdd == 0)
222     return ri[kd] < 0.0;
223   else
224     return al[kd] < ri[kd];
225 }
```

pmd_async.c

A.2 Task II

```
1 /*-------------------------------------------------------------------------
2 Program pmd.c performs parallel molecular-dynamics for Lennard-Jones
3 systems using the Message Passing Interface (MPI) standard.
4 -------------------------------------------------------------------------*/
5 #include "pmd_async_task2.h"
6 #define VMAX 5.0  // Max. velocity value to construct a velocity
      histogram
7 #define NBIN 100  // # of bins in the histogram
8
9 FILE *fpv;
10
11 /*-------------------------------------------------------------------------*/
12 int main(int argc, char **argv) {
13 /*-------------------------------------------------------------------------*/
14   double cpu1,cpu;
15   int i,a;
16
17   MPI_Init(&argc,&argv); /* Initialize the MPI environment */
18   //MPI_Comm_rank(MPI_COMM_WORLD, &sid);  /* My processor ID */
19
20   MPI_Comm_rank(MPI_COMM_WORLD, &gid); //Global rank
21   md_shit = gid%2; // = 1 (MD workers) or 0 (analysis workers)
22   MPI_Comm_split(MPI_COMM_WORLD,md_shit,0,&workers);
23   MPI_Comm_rank(workers,&sid); // Rank in workers
24
25   /* Vector index of this processor */
26   vid[0] = sid/(vproc[1]*vproc[2]);
27   vid[1] = (sid/vproc[2])%vproc[1];
28   vid[2] = sid%vproc[2];
29
30   init_params();
31   if (md_shit) {
32     set_topology();
```

```
33      init_conf();
34      atom_copy();
35      compute_accel();
36    }
37    else
38      if (sid == 0)
39       fpv = fopen("pv.dat","w");
40
41    //printf("%d\n", md_shit);
42    cpu1 = MPI_Wtime();
43    for (stepCount=1; stepCount<=StepLimit; stepCount++) {
44    if (md_shit){
45     printf("%d\n", md_shit);
46     single_step();
47    }
48    if (stepCount%StepAvg == 0) {
49      if (md_shit) {
50        // Send # of atoms, n, to rank gid-1 in MPI_COMM_WORLD
51        MPI_Send(&n, 1, MPI_INT, gid-1, 1000, MPI_COMM_WORLD);
52        // Compose message to be sent
53        for(i=0; i<n; i++)
54          for(a=0; a<3; a++)
55            dbuf[3*i+a] = rv[i][a];
56        // Send velocities of n atoms to rank gid-1 in MPI_COMM_WORLD
57        MPI_Send(dbuf, 3*n, MPI_DOUBLE, gid-1, 2000, MPI_COMM_WORLD);
58        eval_props();
59      }
60      else {
61        // Receive # of atoms, n, from rank gid+1 in MPI_COMM_WORLD
62        MPI_Recv(&n, 1, MPI_INT, gid+1, 1000, MPI_COMM_WORLD,&status);
63        // Receive velocities of n atoms from rank gid+1 in
             MPI_COMM_WORLD
64        MPI_Recv(dbufr, 3*n, MPI_DOUBLE, gid+1, 2000,
             MPI_COMM_WORLD,&status);
65        for(i=0; i<n; i++)
66          for(a=0; a<3; a++)
67            rv[i][a] = dbufr[3*i+a];
68        calc_pv();
69      } // end if
70    } // end if
71    } // end for
72
73
74    cpu = MPI_Wtime() - cpu1;
75    if(md_shit && sid==0)
76      printf("CPU & COMT = %le %le\n",cpu,comt);
77    if(!md_shit && sid==0)
78      fclose(fpv);
```

```
79
80    MPI_Finalize(); /* Clean up the MPI environment */
81    return 0;
82 }
```

pmd_async_task2.c