Minh Tran

Assignment 4 CSCI 596: Parallel Molecular Dynamics

I. Asynchronous Messages

The new program pmd_irecv.c is modifed from pmd.c by inserting proper asynchronous messaging lines of code. Note that nrc is number of atoms received while nsd is number of atoms sent. The MPI_Irecv requires adjustment from MPI_Recv by changing &status to &request and adding MPI_Wait. The changes are marked by //NEW CODE

```
// NEW CODE FOR Asynchronous
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); //handle to know which message to
wait
```

A similar adjustment is made for the buffer (note the commented out code as we do not need the deadlock-avoidance scheme anymore)

```
// NEW CODE FOR Asynchronous, move this ahead of message
buffering to optimize
MPI_Irecv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,
         MPI_COMM_WORLD,&request);

/* Message buffering */
// compose 1-d buffer file: move behind ireceive but before
send will help optimize computation time
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];

// COMMENT OUT BLOCKS OF CODE BELOW
/* Even node: send & recv */
/*if (myparity[kd] == 0) {
  MPI_Send(dbuf,3*nsd,MPI_DOUBLE,inode,20,MPI_COMM_WORLD);
  MPI_Recv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,
           MPI_COMM_WORLD,&status);
}
/* Odd node: recv & send */
/*else if (myparity[kd] == 1) {
  MPI_Recv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,
           MPI_COMM_WORLD,&status);
  MPI_Send(dbuf,3*nsd,MPI_DOUBLE,inode,20,MPI_COMM_WORLD);
}
/* Single layer: Exchange information with myself */
/*else
  for (i=0; i<3*nrc; i++) dbufr[i] = dbuf[i];
*/

// NEW CODE FOR Asynchronous
MPI_Send(dbuf,3*nsd,MPI_DOUBLE,inode,20,MPI_COMM_WORLD);
MPI_Wait(&request,&status); //handle to know which message to
wait
```

To complete this part, run the following on cluster:

- Compile the program by: (The input file pmd.in and header pmd.h are automatically read by the following scripts in both C programs: pmd.c and pmd_irecv.c)

```
#include "pmd.h"
```

```
/* Read control parameters */
fp = fopen("pmd.in","r");
fscanf(fp,"%d%d%d",&InitUcell[0],&InitUcell[1],&InitUcell[2]);
fscanf(fp,"%le",&Density);
fscanf(fp,"%le",&InitTemp);
fscanf(fp,"%le",&DeltaT);
fscanf(fp,"%d",&StepLimit);
fscanf(fp,"%d",&StepAvg);
fclose(fp);
```

  - **Mpicc -o pmd pmd.c -lm**
  - **Mpicc -o pmd_irecv pmd_irecv.c -lm**

- Run sbatch:
  - **Sbatch pmd_irecv.sl**
  - This sbatch will get 4 computing nodes, each with 4 processors. The job consists of 3 runs, each run has two tasks: pmd_irecv and pmd. Then it will generate output pmd_recv.out.

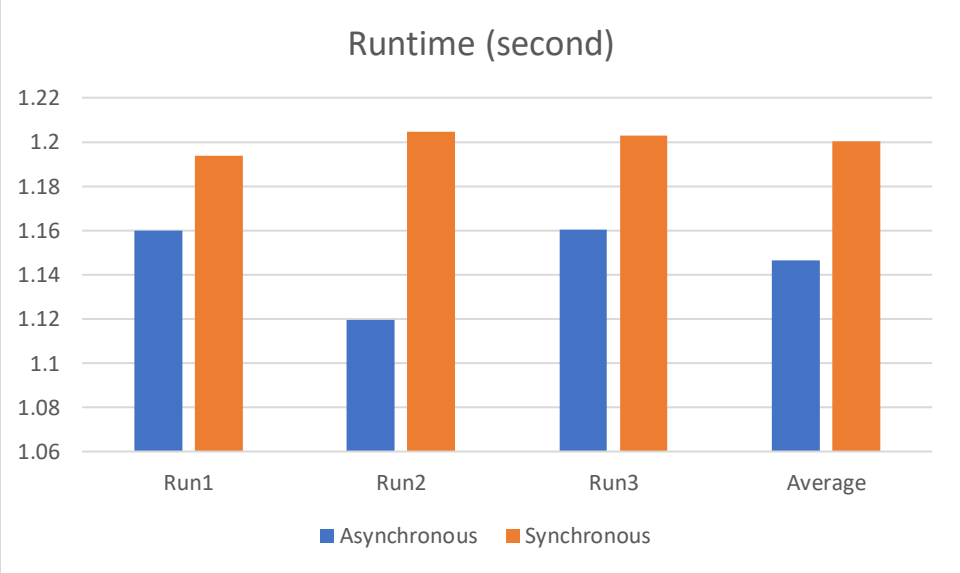
```
#!/bin/bash
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=4
#SBATCH --time=00:01:59
#SBATCH --output=pmd_irecv.out
#SBATCH -A anakano_429

counter=0
while [ $counter -lt 3 ]; do
  echo "***** Asynchronous *****"
  mpirun -n $SLURM_NTASKS ./pmd_irecv
  echo "***** Synchronous *****"
  mpirun -n $SLURM_NTASKS ./pmd
  let counter+=1
done
```

Below is the output pmd_irecv.out. Note that each run consits of one asynchronous part (in red shadow) and one synchronous part (in yellow shadow). The processing computational and the communication time is highlighted in green.

```
***** 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 = 1.159760e+00 1.619362e-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 = 1.193646e+00 1.969182e-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 = 1.161966e+00 1.667682e-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 = 1.204815e+00 2.069850e-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 = 1.160477e+00 1.654520e-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 = 1.202903e+00 2.042630e-01
```

The results of computing time is shown below:

	Asynchronous	Synchronous
Run1	1.15976	1.19364
Run2	1.11966	1.204815
Run3	1.160477	1.202903
Average	1.146632	1.200453



This part can be done interactively:

- **Run salloc –nodes=4 –ntasks-per-node=4 -t 30**

```
[tranmt@d11-02 assignment4]$ salloc --nodes=4 --ntasks-per-node=4 -t 30
salloc: Pending job allocation 298862
salloc: job 298862 queued and waiting for resources
salloc: job 298862 has been allocated resources
salloc: Granted job allocation 298862
salloc: Waiting for resource configuration
salloc: Nodes d11-[02-04,09] are ready for job
```

- In another terminal, run **ssh d11-02** and **top** to see the reserved nodes

```
top - 11:13:44 up 52 days, 10:57,  1 user,  load average: 2.81, 1.50, 0.67
Tasks: 386 total,   2 running, 384 sleeping,   0 stopped,   0 zombie
%Cpu(s):  8.8 us,  0.1 sy,  0.0 ni, 91.1 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem : 19648168+total, 15283179+free, 13537916 used, 30111976 buff/cache
KiB Swap:        0 total,        0 free,        0 used. 17825059+avail Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
230365 brendajm  20   0   28.4g   8.4g  15520 S 192.4  4.5   7:06.99 java
230423 yusongwu  20   0 2317560   1.3g  14276 R  90.4  0.7   5:04.43 python3
230799 tranmt    20   0   60828   2524   1556 R   0.7  0.0   0:00.20 top
     9 root      20   0       0      0      0 S   0.3  0.0  80:30.01 rcu_sched
     1 root      20   0   43624   3952   2528 S   0.0  0.0   4:32.58 systemd
     2 root      20   0       0      0      0 S   0.0  0.0   0:05.22 kthreadd
```

II. Communicators

1. Modified source code

The modifications are clearly marked with //NEW CODE, which are in several places:

- In the header file pmd_split.h, the variable definition is added from calc_pv() as well as the change in the number of processors

```
//NEW CODE copied from calc_pv
#define VMAX 5.0  // Max. velocity value to construct a velocity histogram
#define NBIN 100  // # of bins in the histogram
```

```
int vproc[3] = {2,2,2}, nproc = 8; // NEW CODE for communicators
```

- In the input file pmd_split.in, I change the parameters to the following:

```
5 5 5
0.8
1.0
0.005
30
10
```

- In the program file pmd_split.c (the original copy is from pmd_irecv.c), the major changes are:
  - Change input file name

```
*/
#include "pmd_split.h"
```

  - Add this following the initialization part

```
// MEW CODE
//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);// dovode tp 2 subsets
MPI_Comm_rank(workers,&sid); // Rank in workers
```

- Modify to add rank-dependent task assignment

```
// NEW CODE: need modification by assigning to specfic ranl
// page 3 splitMD note
init_params();
if (md){
  set_topology();
  init_conf();
  atom_copy();
  compute_accel(); /* Computes initial accelerations */
}
else
  if (sid == 0) fpv = fopen("pv.dat","w");
```

```
// NEW CODE: need modification by assigning to specfic ranl
cpu1 = MPI_Wtime();

for (stepCount=1; stepCount<=StepLimit; stepCount++) {
  // page 5 splitMD note
  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(i=0;i<n;i++)
        for (a=0;a<3;a++)
          dbuf[3*i+a] =rv[i][a]; // update 1d array of velocity from
      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 f
      //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);
      //store in array
      for(i=0;i<n;i++)
        for (a=0;a<3;a++)
          rv[i][a]=dbufr[3*i+a];
      calc_pv();
    }

}

cpu = MPI_Wtime() - cpu1;
```

- Add this to the finalization

```
// NEW CODE for finalization slide 7 splitMD.pdf
//if (sid == 0) printf("CPU & COMT = %le %le\n",cpu,comt);
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;
```

- Change most MPI_COMM_WORLD to workers

2. Run the program

   Compile it by **mpicc -o pmd_split pmd_split.c -lm**

   Then run **sbatch pmd_split.sl** on the computing node

   The first output is pmd_split.out, stating the computational time

```
al = 8.549880e+00 8.549880e+00 8.549880e+00
lc = 3 3 3
rc = 2.849960e+00 2.849960e+00 2.849960e+00
al = 8.549880e+00 8.549880e+00 8.549880e+00
lc = 3 3 3
rc = 2.849960e+00 2.849960e+00 2.849960e+00
nglob = 4000
 0.050000  0.878759 -5.139483 -3.821345
 0.100000  0.467645 -4.521725 -3.820257
 0.150000  0.505831 -4.579989 -3.821243
CPU & COMT = 1.356073e-01 9.745559e-03
```

   The second output is the pv.dat, which can be post-processed in excel by separating data of the three timesteps to plot the pdf:

10 time steps			20 time steps			30 time steps	
0.00	0.00E+00		0.00	5.00E-03		0.00	5.00E-03
0.05	0.00E+00		0.05	2.00E-02		0.05	1.50E-02
0.10	0.00E+00		0.10	1.50E-02		0.10	1.50E-02
0.15	0.00E+00		0.15	7.50E-02		0.15	2.00E-02
0.20	0.00E+00		0.20	1.20E-01		0.20	9.50E-02
0.25	0.00E+00		0.25	1.45E-01		0.25	9.50E-02
0.30	0.00E+00		0.30	1.75E-01		0.30	1.20E-01
0.35	0.00E+00		0.35	1.70E-01		0.35	1.55E-01
0.40	0.00E+00		0.40	2.45E-01		0.40	1.60E-01
0.45	0.00E+00		0.45	2.85E-01		0.45	2.80E-01
0.50	0.00E+00		0.50	2.85E-01		0.50	2.40E-01
0.55	0.00E+00		0.55	4.40E-01		0.55	4.05E-01
0.60	0.00E+00		0.60	4.70E-01		0.60	4.40E-01
0.65	0.00E+00		0.65	4.35E-01		0.65	5.10E-01
0.70	0.00E+00		0.70	5.55E-01		0.70	5.25E-01
0.75	0.00E+00		0.75	5.75E-01		0.75	5.75E-01