CSCI596 Assignment 4—Parallel MD—Explanation

Part I: Asynchronous Messages

Programming: Major changes from `pmd.c` are:

1. In `pmd.h`:

```
...
MPI_Request request;
...
```

2. In `atom_copy()`:

```
...
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);
...
MPI_Irecv(dbufr,3*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,20,MPI_COMM_WORLD,&request);
for (i=1; i<=nsd; i++)
  for (a=0; a<3; a++)
    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);
...
```

3. In `atom_move()`:

```
...
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);
...
MPI_Irecv(dbufr,6*nrc,MPI_DOUBLE,MPI_ANY_SOURCE,120,MPI_COMM_WORLD,&request);
for (i=1; i<=nsd; i++)
  for (a=0; a<3; a++) {
    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);
...
```

Run results: For simulating a 108×16 = 1,728-atom Lennard-Jones system on four dual-octocore Intel Xeon (clock speed 2.4 GHz) nodes (using in total of 4 × 4 = 16 processors) for 1,000 molecular-dynamics steps, the table below shows that the asynchronous program is 4.5% faster (averaged over 3 runs). No significant change in execution time is observed, because there is little computation-communication overlap in this particular case. Nevertheless, the result demonstrates the promise of computation-communication overlapping for reducing the execution time.

	Wall-clock time for total execution (seconds)
Original `pmd.c` with synchronous receive	0.638 ± 0.004
Modified `pmd.c` with asynchronous receive	0.609 ± 0.006

Part II: Communicators

Programming: Major changes from `pmd.c` in Part I are:

1. In `pmd.h`:

```
...
#define VMAX 5.0
#define NBIN 100
```

```
int gid,md;
MPI_Comm workers;
FILE *fpv;
...
```

2. In `main()`:

```
...
MPI_Comm_rank(MPI_COMM_WORLD, &gid);
md = gid%2;
MPI_Comm_split(MPI_COMM_WORLD,md,0,&workers);
MPI_Comm_rank(workers, &sid);
...
init_params();
if (md) {
  set_topology();
  init_conf();
  atom_copy();
  compute_accel(); /* Computes initial accelerations */
} else
  if (sid == 0) fpv = fopen("pv.dat","w");
...
for (stepCount=1; stepCount<=StepLimit; stepCount++) {
  if (md) single_step();
  if (stepCount%StepAvg == 0) {
    if (md) {
      MPI_Send(&n,1,MPI_INT,gid-1,1000,MPI_COMM_WORLD);
      for (i=0; i<n; i++)
        for (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 {
      MPI_Recv(&n,1,MPI_INT,gid+1,1000,MPI_COMM_WORLD,&status);
      MPI_Recv(dbufr,3*n,MPI_DOUBLE,gid+1,2000,MPI_COMM_WORLD,&status);
      for (i=0; i<n; i++)
        for (a=0; a<3; a++) rv[i][a] = dbufr[3*i+a];
      calc_pv();
    }
  }
}
...
if (md && sid == 0) printf("CPU & COMT = %le %le\n",cpu,comt);
if (!md && sid == 0) fclose(fpv);
    ...
```

In addition, (1) copy function, `calc_pv()`, in the lecture note before `main()`; and (2) change all occurrence of `MPI_COMM_WORLD` to `workers` in `init_conf()`, `atom_copy()`, `compute_accel()`, `eval_props()`, and `atom_move()` functions.

Run result: