

Assignment 5- Hybrid MPI + MP

CSCI 596: Scientific Computing & Visualization

Anup V Kanale

October 18, 2017

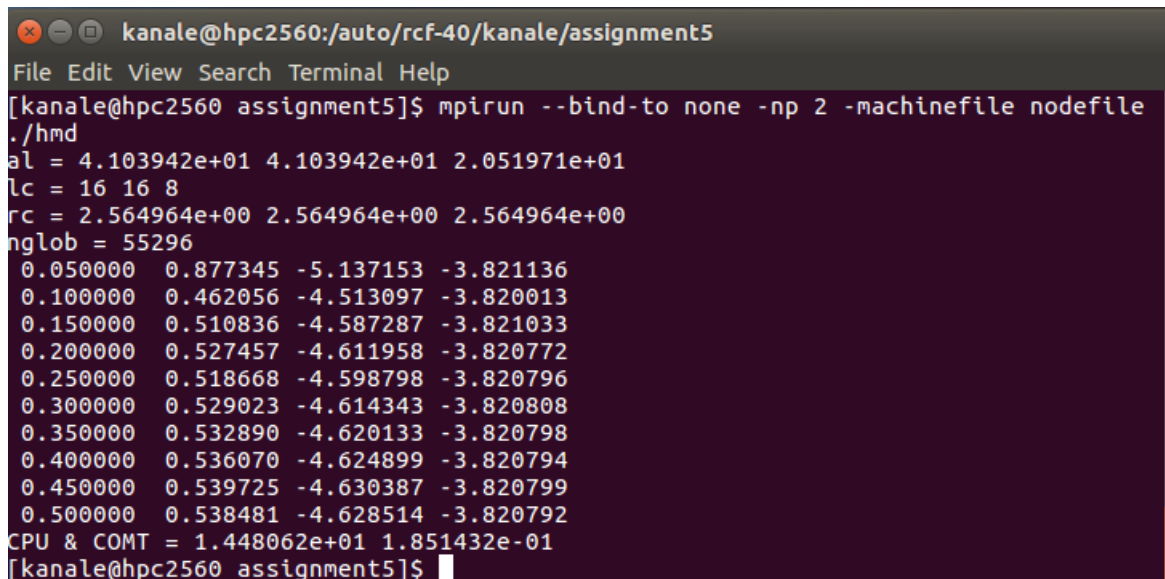
The goal of this assignment is to make use of both MPI and OpenMP in the supplied parallel molecular dynamics code. MPI is used to pass messages between processors, whereas OpenMP uses cores within a processor.

1 Task I– hmd.c

The modifications were made mainly in `compute-accel()` function (apart from `init_params()`) and the `main()`. `compute_accel` and the `main` function are attached in appendix. Apart from this, minor modifications were made in the header file, mainly to declare the number of threads. The rest of the code, which already has MPI was retained from `pmd.c`.

2 Task II– Verification

To verify, the program `hmd.c` was run on HPC. The results (energy) match the parallel molecular dynamics code (`pmd.c`). The output of the run is shown below.



```
kanale@hpc2560:/auto/rcf-40/kanale/assignment5
File Edit View Search Terminal Help
[kanale@hpc2560 assignment5]$ mpirun --bind-to none -np 2 -machinefile nodefile
./hmd
al = 4.103942e+01 4.103942e+01 2.051971e+01
lc = 16 16 8
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
0.050000 0.877345 -5.137153 -3.821136
0.100000 0.462056 -4.513097 -3.820013
0.150000 0.510836 -4.587287 -3.821033
0.200000 0.527457 -4.611958 -3.820772
0.250000 0.518668 -4.598798 -3.820796
0.300000 0.529023 -4.614343 -3.820808
0.350000 0.532890 -4.620133 -3.820798
0.400000 0.536070 -4.624899 -3.820794
0.450000 0.539725 -4.630387 -3.820799
0.500000 0.538481 -4.628514 -3.820792
CPU & COMT = 1.448062e+01 1.851432e-01
[kanale@hpc2560 assignment5]$
```

Figure 1: Output for task 2– Verification

3 Task III– Scalability

In this task, the scalability of the program is tested w.r.t the number of threads. The plot is attached below.

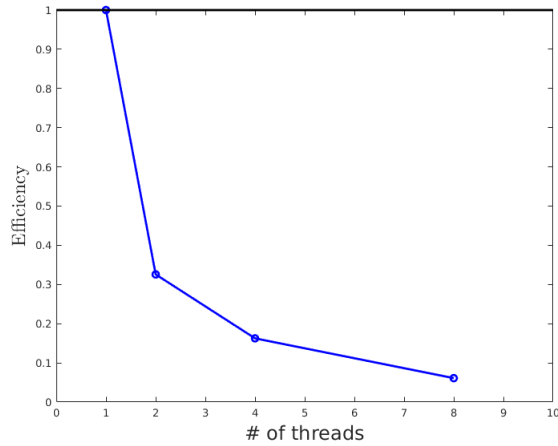


Figure 2: Output for task 3– Scaling

A Appendix– hmd.c

```

1  /*-----
2  Program pmd.c performs parallel molecular-dynamics for Lennard-Jones
3  systems using the Message Passing Interface (MPI) standard.
4  -----*/
5  #include "hmd.h"
6
7  /*-----*/
8  int main(int argc, char **argv) {
9  /*-----*/
10     double cpu1;
11
12     MPI_Init(&argc,&argv); /* Initialize the MPI environment */
13     MPI_Comm_rank(MPI_COMM_WORLD, &sid); /* My processor ID */
14     /* Vector index of this processor */
15     vid[0] = sid/(vproc[1]*vproc[2]);
16     vid[1] = (sid/vproc[2])%vproc[1];
17     vid[2] = sid%vproc[2];
18
19     omp_set_num_threads(nthrd); // for the pragma stuff
20
21     init_params();
22     set_topology();
23     init_conf();
24     atom_copy();
25     compute_accel(); /* Computes initial accelerations */
26
27     cpu1 = MPI_Wtime();
28     for (stepCount=1; stepCount<=StepLimit; stepCount++) {
29         single_step();

```

```

30     if (stepCount%StepAvg == 0) eval_props();
31 }
32 cpu = MPI_Wtime() - cpu1;
33 if (sid == 0) printf("CPU & COMT = %le %le\n",cpu,comt);
34
35 MPI_Finalize(); /* Clean up the MPI environment */
36 return 0;
37 }
38
39 /*-----*/
40 void init_params() {
41 /*-----
42  Initializes parameters.
43  -----*/
44     int a;
45     double rr,ri2,ri6,r1;
46     FILE *fp;
47
48     /* Read control parameters */
49     fp = fopen("hmd.in","r");
50     fscanf(fp,"%d%d%d",&InitUcell[0],&InitUcell[1],&InitUcell[2]);
51     fscanf(fp,"%le",&Density);
52     fscanf(fp,"%le",&InitTemp);
53     fscanf(fp,"%le",&DeltaT);
54     fscanf(fp,"%d",&StepLimit);
55     fscanf(fp,"%d",&StepAvg);
56     fclose(fp);
57
58     /* Compute basic parameters */
59     DeltaTH = 0.5*DeltaT;
60     for (a=0; a<3; a++) al[a] = InitUcell[a]/pow(Density/4.0,1.0/3.0);
61     if (sid == 0) printf("al = %e %e %e\n",al[0],al[1],al[2]);
62
63     /* Compute the # of cells for linked cell lists */
64     for (a=0; a<3; a++) {
65         lc[a] = al[a]/RCUT; /* Cell size  $\hat{a}L\check{e}$  potential cutoff */
66         /* Size of cell block that each thread is assigned */
67         thbk[a] = lc[a]/vthrd[a];
68         /* # of cells = integer multiple of the # of threads */
69         lc[a] = thbk[a]*vthrd[a]; /* Adjust # of cells/MPI process */
70         rc[a] = al[a]/lc[a]; /* Linked-list cell length */
71     }
72     if (sid == 0) {
73         printf("lc = %d %d %d\n",lc[0],lc[1],lc[2]);
74         printf("rc = %e %e %e\n",rc[0],rc[1],rc[2]);
75     }
76

```

```

77  /* Constants for potential truncation */
78  rr = RCUT*RCUT; ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1=sqrt(rr);
79  Uc = 4.0*ri6*(ri6 - 1.0);
80  Duc = -48.0*ri6*(ri6 - 0.5)/r1;
81
82  // printf("finished init params");
83 }
84
85
86 /*-----*/
87 void compute_accel() {
88 /*-----*/
89 Given atomic coordinates, r[0:n+nb-1][], for the extended (i.e.,
90 resident & copied) system, computes the acceleration, ra[0:n-1][], for
91 the residents.
92 -----*/
93  int i,j,a,lc2[3],lcyz2,lcxyz2,mc[3],c,mcl[3],c1;
94  //int bintra;
95  double rrCut,lpe;
96  double lpe_td[nthrd];
97
98  /* Reset the potential & forces */
99  lpe = 0.0;
100  for (i=0; i<n; i++) for (a=0; a<3; a++) ra[i][a] = 0.0;
101  for (i=0; i<nthrd; i++) lpe_td[i] = 0.0;
102
103  /* Make a linked-cell list,
104     lscl-----*/
105  for (a=0; a<3; a++) lc2[a] = lc[a]+2;
106  lcyz2 = lc2[1]*lc2[2];
107  lcxyz2 = lc2[0]*lcyz2;
108
109  /* Reset the headers, head */
110  for (c=0; c<lcxyz2; c++) head[c] = EMPTY;
111
112  /* Scan atoms to construct headers, head, & linked lists, lscl */
113
114  for (i=0; i<n+nb; i++) {
115      for (a=0; a<3; a++) mc[a] = (r[i][a]+rc[a])/rc[a];
116
117      /* Translate the vector cell index, mc, to a scalar cell index */
118      c = mc[0]*lcyz2+mc[1]*lc2[2]+mc[2];
119
120      /* Link to the previous occupant (or EMPTY if you're the 1st) */
121      lscl[i] = head[c];
122
123      /* The last one goes to the header */

```

```

124     head[c] = i;
125 } /* Endfor atom i */
126
127 /* Calculate pair
      interaction_____*/
128
129 rrCut = RCUT*RCUT;
130
131 #pragma omp parallel private(mc,a,c,c1,mc1,i,j,)
132 {
133
134     int std,vtd[3],mofst[3];
135     double dr[3],rr,ri2,ri6,r1,fcVal,f,vVal;
136
137     std = omp_get_thread_num();
138     vtd[0] = std/(vthrd[1]*vthrd[2]);
139     vtd[1] = (std/vthrd[2])%vthrd[1];
140     vtd[2] = std%vthrd[2];
141     for (a=0; a<3; a++)
142         mofst[a] = vtd[a]*thbk[a];
143
144
145     /* Scan inner cells */
146     for (mc[0]=mofst[0]+1; mc[0]<=mofst[0]+thbk[0]; (mc[0])++)
147     for (mc[1]=mofst[1]+1; mc[1]<=mofst[1]+thbk[1]; (mc[1])++)
148     for (mc[2]=mofst[2]+1; mc[2]<=mofst[2]+thbk[2]; (mc[2])++) {
149
150         /* Calculate a scalar cell index */
151         c = mc[0]*lcyz2+mc[1]*lc2[2]+mc[2];
152         /* Skip this cell if empty */
153         if (head[c] == EMPTY) continue;
154
155         /* Scan the neighbor cells (including itself) of cell c */
156         for (mc1[0]=mc[0]-1; mc1[0]<=mc[0]+1; (mc1[0])++)
157         for (mc1[1]=mc[1]-1; mc1[1]<=mc[1]+1; (mc1[1])++)
158         for (mc1[2]=mc[2]-1; mc1[2]<=mc[2]+1; (mc1[2])++) {
159
160             /* Calculate the scalar cell index of the neighbor cell */
161             c1 = mc1[0]*lcyz2+mc1[1]*lc2[2]+mc1[2];
162             /* Skip this neighbor cell if empty */
163             if (head[c1] == EMPTY) continue;
164
165             /* Scan atom i in cell c */
166             i = head[c];
167             while (i != EMPTY) {
168
169                 /* Scan atom j in cell c1 */
170                 j = head[c1];

```

```

171     while (j != EMPTY) {
172
173         /* No calculation with itself */
174         if (j != i) {
175             /* Logical flag: intra(true)- or inter(false)-pair atom */
176             // bintra = (j < n);
177
178             /* Pair vector dr = r[i] - r[j] */
179             for (rr=0.0, a=0; a<3; a++) {
180                 dr[a] = r[i][a]-r[j][a];
181                 rr += dr[a]*dr[a];
182             }
183
184             /* Calculate potential & forces for intranode pairs (i < j)
185              & all the internode pairs if rij < RCUT; note that for
186              any copied atom, i < j */
187             if (rr<rrCut) {
188                 ri2 = 1.0/rr; ri6 = ri2*ri2*ri2; r1 = sqrt(rr);
189                 fcVal = 48.0*ri2*ri6*(ri6-0.5) + Duc/r1;
190                 vVal = 4.0*ri6*(ri6-1.0) - Uc - Duc*(r1-RCUT);
191                 //if (bintra) lpe += vVal; else lpe += 0.5*vVal;
192                 lpe_td[std] += 0.5*vVal;
193                 for (a=0; a<3; a++) {
194                     f = fcVal*dr[a];
195                     ra[i][a] += f;
196                     //if (bintra) ra[j][a] -= f;
197                 }
198             }
199             } /* Endif not self */
200
201             j = lscl[j];
202             } /* Endwhile j not empty */
203
204             i = lscl[i];
205             } /* Endwhile i not empty */
206
207         } /* Endfor neighbor cells, c1 */
208
209     } /* Endfor central cell, c */
210
211     } // Ending pragma or omp parallel
212     for(i=0; i<nthrd; i++) lpe += lpe_td[i];
213
214     /* Global potential energy */
215     MPI_Allreduce(&lpe,&potEnergy,1,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
216 }

```

hmd.c