CSCI596_HW5
Student Name: Cancan Hua
USC ID: 4612363893

1. Modifications of source code of hmd.c:

```
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 */
    /* Vector index of this processor */
    vid[0] = sid / (vproc[1] * vproc[2]);
    vid[1] = (sid / vproc[2]) % vproc[1];
    vid[2] = sid % vproc[2];

    /*-------changed here----------*/
    omp_set_num_threads(nthrd);
    /* ---------------------------*/
```

```
/*------------------------------------------------------------------------*/
void init_params()
{
    /*------------------------------------------------------------------------
Initializes parameters.
------------------------------------------------------------------------*/
    int a;
    double rr, ri2, ri6, r1;
    FILE *fp;

    /* 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);

    /* Compute basic parameters */
    DeltaTH = 0.5 * DeltaT;
    for (a = 0; a < 3; a++)
        al[a] = InitUcell[a] / pow(Density / 4.0, 1.0 / 3.0);
    if (sid == 0)
        printf("al = %e %e %e\n", al[0], al[1], al[2]);
```

```
    /*-------changed here---------*/

    /* Compute the # of cells for linked cell lists */
    for (a = 0; a < 3; a++)
    {
        lc[a] = al[a] / RCUT;

        /* Size of cell block that each thread is assigned */
        thbk[a] = lc[a]/vthrd[a];
        /* # of cells = integer multiple of the # of threads */
        lc[a] = thbk[a]*vthrd[a]; /* Adjust # of cells/MPI process */

        rc[a] = al[a] / lc[a];
    }
    /*----------------*/
```

```
/*-------changed here---------*/

/*----------------------------------------------------------------------*/
void compute_accel()
{
    /*----------------------------------------------------------------------
Given atomic coordinates, r[0:n+nb-1][], for the extended (i.e.,
resident & copied) system, computes the acceleration, ra[0:n-1][], for
the residents.
----------------------------------------------------------------------*/
    int i, j, a, lc2[3], lcyz2, lcxyz2, mc[3], c, mc1[3], c1;
    /*  changed  */
    // int bintra;
    // double dr[3], rr, ri2, ri6, r1, rrCut, fcVal, f, vVal, lpe;
    double rrCut, lpe;
    double lpe_td[nthrd];

    /* Reset the potential & forces */
    lpe = 0.0;
    for (i = 0; i < n; i++) {
        for (a = 0; a < 3; a++) {
            ra[i][a] = 0.0;
        }
    }

    for (i = 0; i < nthrd; i++) {
        lpe_td[i] = 0.0;
    }

    /*  changed  */

    /* Make a linked-cell list, lscl--------------------------------------*/
```

```
    for (a = 0; a < 3; a++)
        lc2[a] = lc[a] + 2;
    lcyz2 = lc2[1] * lc2[2];
    lcxyz2 = lc2[0] * lcyz2;

    /* Reset the headers, head */
    for (c = 0; c < lcxyz2; c++)
        head[c] = EMPTY;

    /* Scan atoms to construct headers, head, & linked lists, lscl */

    for (i = 0; i < n + nb; i++) {
        for (a = 0; a < 3; a++)
            mc[a] = (r[i][a] + rc[a]) / rc[a];

        /* Translate the vector cell index, mc, to a scalar cell index */
        c = mc[0] * lcyz2 + mc[1] * lc2[2] + mc[2];

        /* Link to the previous occupant (or EMPTY if you're the 1st) */
        lscl[i] = head[c];

        /* The last one goes to the header */
        head[c] = i;
    } /* Endfor atom i */

    /* Calculate pair interaction---------------------------------------*/

    rrCut = RCUT * RCUT;

    /*----------------*/
```

```
/*-------changed here---------*/
    #pragma omp parallel private(mc, c, mc1, c1, i, j, a)
    {
    double dr[3], rr, ri2, ri6, r1, fcVal, f, vVal;
    int std,vtd[3],mofst[3];

    std = omp_get_thread_num();
    vtd[0] = std/(vthrd[1]*vthrd[2]);
    vtd[1] = (std/vthrd[2])%vthrd[1];
    vtd[2] = std%vthrd[2];
    for (a = 0; a < 3; a++) {
        mofst[a] = vtd[a] * thbk[a];
    }
    /* Scan inner cells */
    // for (mc[0] = 1; mc[0] <= lc[0]; (mc[0])++)
    // for (mc[1] = 1; mc[1] <= lc[1]; (mc[1])++)
```

```
  // for (mc[2] = 1; mc[2] <= lc[2]; (mc[2])++)
for (mc[0]=mofst[0]+1; mc[0]<=mofst[0]+thbk[0]; (mc[0])++)
    for (mc[1]=mofst[1]+1; mc[1]<=mofst[1]+thbk[1]; (mc[1])++)
        for (mc[2]=mofst[2]+1; mc[2]<=mofst[2]+thbk[2]; (mc[2])++)
        {

            /* Calculate a scalar cell index */
            c = mc[0] * lcyz2 + mc[1] * lc2[2] + mc[2];
            /* Skip this cell if empty */
            if (head[c] == EMPTY)
                continue;

            /* Scan the neighbor cells (including itself) of cell c */
            for (mc1[0] = mc[0] - 1; mc1[0] <= mc[0] + 1; (mc1[0])++)
                for (mc1[1] = mc[1] - 1; mc1[1] <= mc[1] + 1; (mc1[1])++)
                    for (mc1[2] = mc[2] - 1; mc1[2] <= mc[2] + 1; (mc1[2])++)
                    {
                        /* Calculate the scalar cell index of the neighbor cell */
                        c1 = mc1[0] * lcyz2 + mc1[1] * lc2[2] + mc1[2];
                        /* Skip this neighbor cell if empty */
                        if (head[c1] == EMPTY)
                            continue;
                        /* Scan atom i in cell c */
                        i = head[c];
                        while (i != EMPTY)
                        {
                            /* Scan atom j in cell c1 */
                            j = head[c1];
                            while (j != EMPTY)
                            {
                                /* No calculation with itself */
                                if (j != i) {
                                    /* Logical flag: intra(true)- or inter(false)-pair atom */
                                    // bintra = (j < n);

                                    /* Pair vector dr = r[i] - r[j] */
                                    for (rr = 0.0, a = 0; a < 3; a++) {
                                        dr[a] = r[i][a] - r[j][a];
                                        rr += dr[a] * dr[a];
                                    }

                                    /* Calculate potential & forces for intranode pairs (i < j)
                & all the internode pairs if rij < RCUT; note that for
                any copied atom, i < j */
                                   // if (i < j && rr < rrCut)
                                    if (rr < rrCut) {
                                        ri2 = 1.0 / rr;
                                        ri6 = ri2 * ri2 * ri2;
                                        r1 = sqrt(rr);
                                        fcVal = 48.0 * ri2 * ri6 * (ri6 - 0.5) + Duc / r1;
                                        vVal = 4.0 * ri6 * (ri6 - 1.0) - Uc - Duc * (r1 - RCUT);
```

```
                                        // if (bintra) lpe += vVal; else lpe_td[std] += 0.5 * vVal;
                                        lpe_td[std] += 0.5 * vVal;
                                        for (a = 0; a < 3; a++)
                                        {
                                            f = fcVal * dr[a];
                                            ra[i][a] += f;
                                            // if (bintra) ra[j][a] -= f;
                                        }
                                    }
                                } /* Endif not self */

                                j = lscl[j];
                            } /* Endwhile j not empty */

                            i = lscl[i];
                        } /* Endwhile i not empty */

                    } /* Endfor neighbor cells, c1 */

            } /* Endfor central cell, c */

    } // End omp parallel


    for (i=0; i<nthrd; i++) lpe += lpe_td[i];

    /* Global potential energy */
    MPI_Allreduce(&lpe, &potEnergy, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
}
/*----------------*/
```

2. The Standard Output from The Run:

```
[cancan@discovery2 hw5]$ more hmd.out
========================================
SLURM_JOB_ID = 6187844
SLURM_JOB_NODELIST = d05-[35-36]
TMPDIR = /tmp/SLURM_6187844
========================================
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 = 5.624527e+00 3.497457e-02
[cancan@discovery2 hw5]$
```

3. Standard Output of Running “An 8-core node with one MPI process and the number of threads varying from 1, 2, 4, to 8, with certain input parameters”:

```
[cancan@discovery2 hw5]$ more hmd-scale.out
=========================================
SLURM_JOB_ID = 6188068
SLURM_JOB_NODELIST = d05-08
TMPDIR = /tmp/SLURM_6188068
=========================================
8 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 1.178163e+01 2.274214e-02
4 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 1.172163e+01 1.919024e-02
2 threads
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 9.165723e+00 1.666681e-02
1 thread
al = 4.103942e+01 4.103942e+01 4.103942e+01
lc = 16 16 16
rc = 2.564964e+00 2.564964e+00 2.564964e+00
nglob = 55296
CPU & COMT = 1.442221e+01 1.576104e-02
[cancan@discovery2 hw5]$
```

Plot of strong-scaling parallel efficiency as a function of the number of threads: