High Performance Computing

Assignment 5

In the simulation of a collection of soft particles (such as proteins in a fluid), there is a repulsive force between a pair of particles when they overlap. The goal of this assignment is to use parallel computing to accelerate the computation of these repulsive forces, using multiple cores on Intel Xeon Phi with OpenMP. A function called `force_repulsion` shown below is an example of how to compute the repulsive forces.

In the `force_repulsion` function, the particles are assumed to have unit radius. The particles are in a "simulation box" of dimensions $L \times L \times L$. The dimension L is chosen such that the volume fraction of particles is $\phi = 0.3$. The simulation box has periodic (wrap-around) boundary conditions, which explains why we need to use the remainder function to compute the distance between two particles. If the particles overlap, i.e., the distance s between two particles is less than 2, then the repulsive force is proportional to

$$k(2 - s)$$

where k is a force constant. The force is along the vector joining the two particles.

For your implementation, note that you must avoid write conflicts in a correct parallelization. You may experiment with using atomic operations and with algorithms that avoid write conflicts, e.g., multicoloring.

(a) **20 marks.** For OpenMP, experiment with using different thread scheduling policies (static, dynamic, guided) and chunk sizes for problems of different sizes (100, 1000, 10000, and 100000 particles). The goal is to find the fastest strategy for different problem sizes. Use graphs to plot your results. You are also free to modify the code any way you wish to try to reduce computation time. Part of your score for this section depends on how fast your code runs for the different problem sizes.

(b) **10 marks.** Write a program that tests the correctness of your code. This can be done by computing the correct forces and comparing them to the forces computed by your optimized code. Give evidence in your report that your program works correctly using your test program. Note that floating point arithmetic is not associative, so you may not get bit-wise identical results.

(c) **5 marks.** How much faster is your accelerated code compared to the provided baseline code? Include timings for different problem sizes. Be sure to include a listing of your code in your report.

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <sys/time.h>

double get_walltime()
{
    struct timeval tp;
    gettimeofday(&tp, NULL);
    return (double) (tp.tv_sec + tp.tv_usec*1e-6);
}
```

```
void force_repulsion(int np, const double *pos, double L, double krepulsion,
    double *forces)
{
    int i, j;
    double posi[4];
    double rvec[4];
    double s2, s, f;

    // initialize forces to zero
    for (i=0; i<3*np; i++)
        forces[i] = 0.;

    // loop over all pairs
    for (i=0; i<np; i++)
    {
        posi[0] = pos[3*i  ];
        posi[1] = pos[3*i+1];
        posi[2] = pos[3*i+2];

        for (j=i+1; j<np; j++)
        {
            // compute minimum image difference
            rvec[0] = remainder(posi[0] - pos[3*j  ], L);
            rvec[1] = remainder(posi[1] - pos[3*j+1], L);
            rvec[2] = remainder(posi[2] - pos[3*j+2], L);

            s2 = rvec[0]*rvec[0] + rvec[1]*rvec[1] + rvec[2]*rvec[2];

            if (s2 < 4)
            {
                s = sqrt(s2);
                rvec[0] /= s;
                rvec[1] /= s;
                rvec[2] /= s;
                f = krepulsion*(2.-s);

                forces[3*i  ] +=  f*rvec[0];
                forces[3*i+1] +=  f*rvec[1];
                forces[3*i+2] +=  f*rvec[2];
                forces[3*j  ] += -f*rvec[0];
                forces[3*j+1] += -f*rvec[1];
                forces[3*j+2] += -f*rvec[2];
            }
        }
    }
}

int main(int argc, char *argv[])
{
    int i;
    int np = 100;             // default number of particles
    double phi = 0.3;         // volume fraction
    double krepulsion = 125.; // force constant
    double *pos;
    double *forces;
    double time0, time1;

    if (argc > 1)
        np = atoi(argv[1]);

    // compute simulation box width
    double L = pow(4./3.*3.1415926536*np/phi, 1./3.);

    // generate random particle positions inside simulation box
    forces = (double *) malloc(3*np*sizeof(double));
    pos    = (double *) malloc(3*np*sizeof(double));
    for (i=0; i<3*np; i++)
```

```
        pos[i] = rand()/(double)RAND_MAX*L;

    // measure execution time of this function

    time0 = get_walltime();
    force_repulsion(np, pos, L, krepulsion, forces);
    time1 = get_walltime();
    printf("number of particles: %d\n", np);
    printf("elapsed time: %f\n", time1-time0);

    free(forces);
    free(pos);

    return 0;
}
```