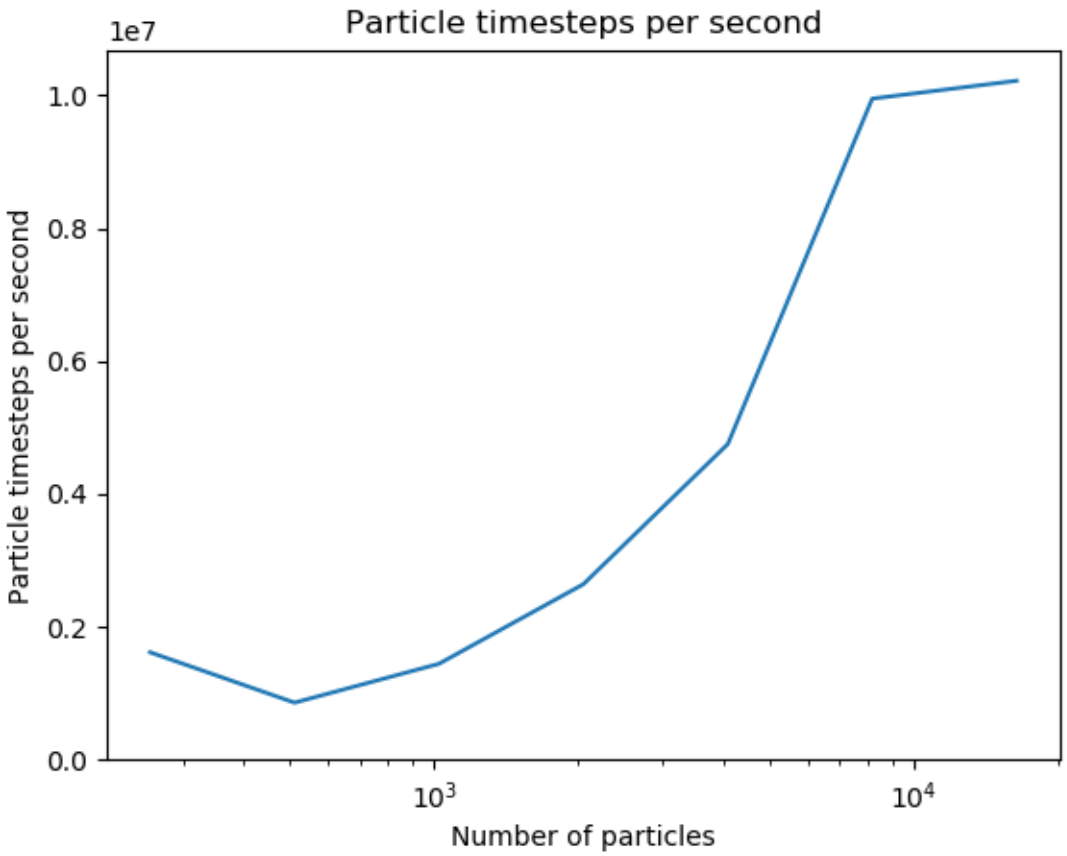
Exercise 4: molecular dynamics simulation

Tian-Ruei Kuan



Optimizations:

1. Make **accelerate_indirect()** in a parallel region.

```
474 +         #pragma omp parallel
475 +         {
476 +           accelerate_indirect(_L, _k, _r, Np, x, y, z, u, v, w);
477 +         }
```

2. Parallelize most for loops in **accelerate_indirect()**, **bin_particles()**, and **unbin_particles()** with static schedule.

```
180 +       #pragma omp for schedule(static)
```

3. Collapse the triple for loop.

```
327         // triple loop over boxes: by design, each instance of the main loop
328         // body can be done independently
329 +       #pragma omp for collapse(3) schedule(static)
330         for (size_t i = 0; i < bpd; i++) {
331           for (size_t j = 0; j < bpd; j++) {
332             for (size_t k = 0; k < bpd; k++) {
```

4. Apply nowait to velocity initialization because binning particles does not need velocity information.

```
315 +       #pragma omp for schedule(static) nowait
316         for (size_t p = 0; p < Np; p++) {
317           ub[p] = 0.;
318           vb[p] = 0.;
319           wb[p] = 0.;
320         }
321
```

5. In **bin_particles()**, use atomic capture when updating **particle_perm.**

```
202 +         #pragma omp atomic capture
203           particle_perm[p] = box_offsets[box+1]++;
```

6. In **bin_particles()**, keep prefix sum scan serial.

```
206 +       #pragma omp single
207 +       {
208 +         // Do a prefix sum (scan) on the counts to get the final box_offsets
209 +         for (size_t b = 0; b < Nb; b++) {
210 +           box_offsets[b + 1] += box_offsets[b];
211 +         }
212         }
```

7. Add **need_rebin()** function to only perform particle binning when necessary.

```
322 +       if(need_rebin(x, y, z)) {
323 +         // reorder the particles
324 +         bin_particles(Np, x, y, z, &_particle_box[0], &_particle_perm[0], &_box_offsets[0], xb, yb, zb);
325 +       }
```

8. In **need_rebin()** function, it will keep track of the particle positions from the last time the particles are binned, and calculate the maximum distance to decide whether particle binning is necessary. A parallel reduction for loop is used to calculate the maximum distance.

```
226 +     bool need_rebin(const double *x,
227 +                     const double *y,
228 +                     const double *z)
229 +     {
230 +       if(_X_last == nullptr) {
231 +         _X_last = new State(*_X);
232 +         return true;
233 +       }
234 +
235 +       double *x_last, *y_last, *z_last;
236 +       size_t Np_last;
237 +       _X_last->get_arrays(Np_last, x_last, y_last, z_last);
238 +
239 +       static double max_dist = 0;
240 +       #pragma omp for reduction(max:max_dist)
241 +       for (size_t p = 0; p < Np_last; p++) {
242 +         double dx, dy, dz;
243 +         double dist = sqrt(dist_and_disp(x[p], y[p], z[p], x_last[p], y_last[p], z_last[p], _L, &dx, &dy, &dz));
244 +         max_dist = std::max(dist, max_dist);
245 +       }
246 +
247 +       if(_box_width - 2 * max_dist > 2 * _r) {
248 +         return false;
249 +       }
250 +       else {
251 +         _X_last->copy(*_X);
252 +         return true;
253 +       }
254 +     }
```