1) Project Description

In molecular dynamics simulation of proteins or other bio-molecules, one must compute the bonded and non-bonded forces of each atom, before solving the system of Newton equations, i.e., system of ODEs, to compute the new positions of the atoms, for each time step. These issues motivate this project but you do not need to know any physics above high school levels to enjoy this project.

You place N particles in a 3D box of dimensions 12 x 12 x 12 randomly, i.e., you generate the initial coordinates (floating-point numbers) of these particles with (x_i, yi, z_i)$\in$ (–6.0, 6.0] $\forall$ i = 1, 2,…,N. These particles carry random charges Q_i = {–3,–2,–1, 1, 2, 3 }$\forall$ i = 1, 2,…,Nequally likely, i.e., each of your particles has an equal 1/6 chance to carry any one of the six charge values. The force vector for a pair of particles i and j is (ignoring a constant)

$$\boldsymbol{f}_{ij} = \frac{Q_i Q_j}{r_{ij}^3}\boldsymbol{r}_{ij}$$

where $\boldsymbol{r}_{ij} = x_i - x_j$ is the vector connecting the two particles and $r\_` = |\boldsymbol{r}\_`|$ magnitude of the vector or distance.
Naturally, the total force on particle i is

$$\boldsymbol{F}_i = \sum_{j \neq i}^{N} \boldsymbol{f}_{ij}$$

Please complete the following:
(1) Design algorithm(s) to compute the total force vectors for all N particles;
(2) Implement your algorithm(s) on a parallel computer;
(3) Test the performances of your algorithm(s) on your true parallel computer with P = 1, 4, 8, 16, 32 cores for N = 1000, 2000, 4000;
(4) Collect your performance results for the above 15 experiments, i.e., for each P value, you
perform timing experiments for thee N values;
(5) Plot the speedup curves;
(6) Comment on your performance results.

2) Algorithm description and pseudo-code describing the main structure of your programs. Source code with compile- and run-time options to enable grading tests (when and where appropriate).

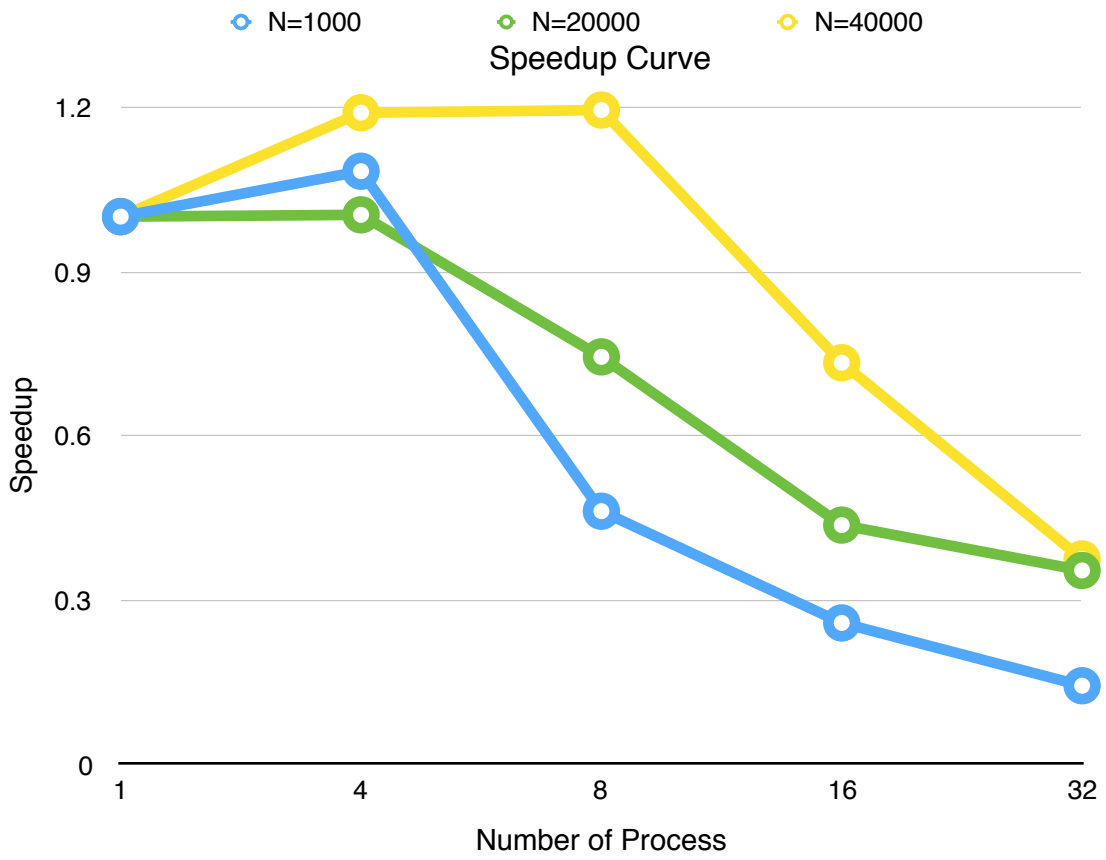
Algorithm:

```
//Separate the dataset into P subparts as evenly as possible and assign the sub-datasets into P processes.
if (rank < size -1 ) {
   generate N/size particles —> particle array I
} else {
   generate N-N/size*(size-1) particles —> particle array I
}

//assign the initial partial array J for each process
particle array I —> particle array J

Do {
   //calculate the interaction between particle array I and particle array J
   interaction(array I, array J);

   //rotate all processor's array J. Processor 0 gives its J to Processor 1, P1 gives its J to P2, …, P(size-1) gives its J to P0.
   rotation(array J);

} while (The initial J has not been passed back to its original processor yet);
```

3) Results (numbers, tables, and figures, etc)

Speedup Table (T(1)/T(P))

	N=1000	N=20000	N=40000
1	1	1	1
4	1.0828729281768	1.00325732899023	1.1897233201581
8	0.462264150943396	0.743961352657005	1.19444444444444
16	0.258234519104084	0.436879432624113	0.733252131546894
32	0.143589743589744	0.354022988505747	0.375077881619938



4) Analysis of program performance

Adding not too many processor, parallel algorithm works faster than sequential algorithm. But when there are too many processors, the parallel implementation is not obviously fast and could to even slower. This could be caused by the fact that the time cost of communication is too expensive so that it vanishes the advantage of less time cost of computation. However, when there are more particles, the advantages of less time cost of computational is more obvious, which makes the speedup curve drops slower.