**Part I: Profiling (analysis/report)**

1. **Profile the Code**

the updateBody() function has 3 loops. The first one is to calculate distances and forces between each particle. The second one is to calculate the position of each particle in next delta t. The last one is to calculate velocity for every particle. Also, the first and the third loop calculate min distance between every two particles and the max speed for a particle.

1. **Set up a proper performance/upscaling model**

Considering that most of parts in this program can be parallel, according to Amadhl’s Law(Strong scaling), t(p) = f · t(1) + (1 − f)t(1) p, most of its part can be the part of 1-f where f is the part has to be serial. But it t(1) = f · t(p) + (1 − f)t(p) · p(strong scaling) still can work. It is only dependent on the size of data can change or not.

Vectorization: the first loop has a branch (if (i != j)) and it means that i is dependent on j so it cannot be vectorized. At first, the second and third loop cannot be vectorized because of internal dependence in variable x and v. Considering that they should be independent, I add a #pragma ivdep to ignore internal dependence. In addition, for the third loop, we should build up a temp velocity array to store the velocity and calculate the max velocity in the end in order to make it vectorize. From the report, we can accelerate 1.2x in loop 3. But it still hard to accelerate loop 2 because the x and v are unaligned according to the report.

OpenMP: It can be applied into every three loops. Using more threads, we can split a loop into many smaller loops and modify the memory (like force, x and v) simultaneously. Then in critical path we can have a reduction to get max/min value (like minDx and maxV) from all threads.

MPI: Similar but slightly different from OpenMP, MPI uses different tasks instead of threads to do nearly same things as OpenMP do in loops. However, we need to split the for loops by ourselves. We can use rank and size of the task to split for loops. For example, every loop need to do NumberOfBodies/taskSize loop and we can use taskRank to ensure that they will not do same job as other tasks. For the first and third loop, it can use MPI\_ALLReduce function to send and get message between every tasks and they will finally get the minDx and maxV value from every tasks.

**3. contextualise your statements with Flynn’s taxonomy**

Vectorization: SIMD, it is synchronous execution of one single instruction. We use some array operations like dot product instead of using loops to get values like x, v and force.

OpenMP: MIMD, they do asynchronous executions on same data space like calculating the local min and max value and using critical path to collect all results from threads to calculate the global min/max value.

MPI: SPMD, multiple autonomous processors simultaneously execute the same program at independent points by passing message to get max/min