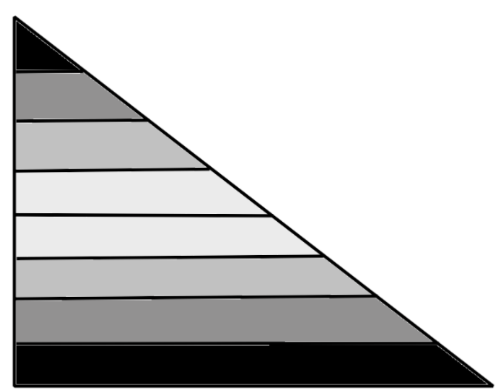
Group 5

Brandon Bratcher	R#11520450
James Gibson	R#11671129
Richard Pearson	R#11786900
Dhruv Maniar	R#11713343

We were tasked with parallelizing a section of code with two for loops. This code has two arrays, one represents the position of particles in space. The second array will store the net force on each particle applied by every other particle. The two for loops do the calculations necessary to determine the net force, adding all forces to the right of the particles and subtracting all forces to the left of the particle.

The second for loop's number of iterations depends on the iteration of the outer for loop. This results in a triangular shaped workload, in the sense that each iteration the work done by the inner loop will increase by a linear amount. We need to balance this load among a given number of processes. Each process must iterate through the entire array. We divide the iterations of the outer loop among the process such that the sum of work in the sections each process gets will always be the same. This is shown in the figure below. For example, the black process will get the shortest and longest sections while the white process gets the two sections in the middle. This means that each process will have the exact same total workload.



To achieve a log2(p) communication complexity, a many-to-one communication was used. To begin, all odd processes send their entire f array to the closest even process to the left of it. Once a process sends, it becomes 'inactive' and will no longer send or receive data. Now, every other 'active' process sends to its nearest neighbor to the left of it. The processes that did not send during an iteration will instead receive from its closest neighbor to the right of it. This communication pattern repeats until process 0 is the only active process left, meaning that it has all the data from every other process. By sending/receiving the entire n array during each communication the total communication complexity ends up being n log2(p).