By, Naveen Chalawadi

Justin K Augustine

**PROJECT 2**

**Introduction**

The project discuss about parallelizing a particle system on a shared memory system and a distributed memory system. In our particle system, particles interact by repelling each other. The interaction is by near-by force, and there is a relatively short cutoff distance. If the distance between two particles is larger than the cutoff distance they do not interact. Otherwise there is a certain repelling force between them. The particles move according to Newton's Laws. Here we are dividing the entire area into bins. It will help to select the area around a particle which will exert force on the it. We simulate the particle system using discrete time steps: we divide time into short intervals of length *dt* during which we think that the particles move uniformly in straight line.

We provide an implementation that naively computes the forces on the particles by iterating through every pair of particles, and we would expect the asymptotic complexity of our simulation to be O(n^2). However, in our simulation we have chosen a density of particles sufficiently low so that with n particles we expect only O(n) interactions. An efficient implementation can reach this time complexity. The first part of the project will be to implement this linear time solution in a serial code, given a naive O(n^2) implementation. After running in time T=O(n) on a serial processor, we run close to time T/p using p processors attempt to reach this speedup using both OpenMP and MPI respectively.

**Advantages and Disadvantages of Open MP and MPI**

Open MP and MPI will reduce the compilation time when compared to serial compilation. But Open MP is easier to program and easier to debug than MPI. It is easy to convert a serial program into Open MP than to MPI and it can still run the code as a serial program. Here also we add the directives gradually to the program which is not possible in MPI. MPI can be used in more compilers whereas Open MP works only in compilers that support it. MPI also works on both shared or distributed memory architecture whereas Open MP works only on shared memory computers. Open MP is easier to understand than MPI. Open MP uses loop parallelization. MPI’s performance is limited by the communication between the nodes.

**Instructions on building the program:**

1. use **make all** command to create all the executables needed to run the program

2. use **chmod 777 job-crill-\*** to set the permissions for the batch files

3. use **chmod 777 auto-crill-\*** to set the permissions for batch files

4. use **./job-crill-\*** to schedule jobs

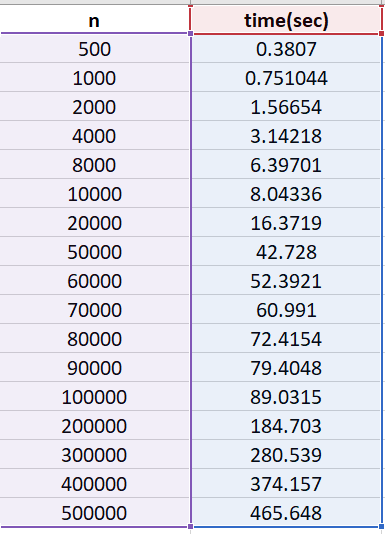
5. use **./auto-crill-\*** to schedule jobs

**1. Serial Implementation:**

Our serial code uses a binning structure to execute the interaction step in O(n) time. The 2D grid is divided into small squared bins. The size of cutoﬀ was used to determine the bin size, since it is the maximum distance of interaction. Each bin stores the location of the nearby particles including neighbors that fall on the processor boundaries. Thus, each particle can only interact with particles in its own square and its 8 neighboring squares on a single iteration. Furthermore, bins that are attached to the edges will have less neighbors and thus they were ﬁxed accordingly. The small bins are stored in a data structure type where each bin has a unique consecutive index. Each bin points to the particles indices contained inside the bin. After each iteration, we empty the bins and update them according to the new particle locations. In the interaction step, each particle only need to check for particles within its bin and neighboring bins. Since the number of particles per square is bound (and equal to 1 on average), the interaction step takes O(n) time.

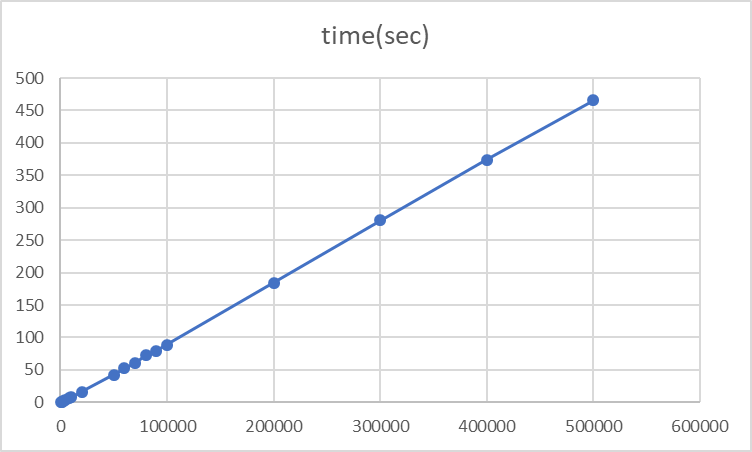
**Graphs**

**Time taken for different number of particle in serial computation (using binning).**



Number of particles (n) is given in the x- axis

Time taken is given in the y- axis



**2. Parallel Implementation using OpenMP**

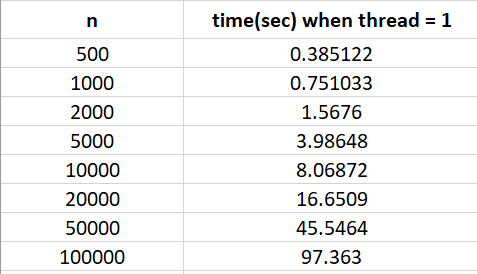
We implement OpenMP for shared memory parallelization. Synchronization is used after each parallel region in order to prevent race conditions. First, we assign each particle to bins in parallel. Next, we assign each bin to a lock (using omp lock t). When we compute forces in each bin in parallel, there will be little contention in most bins because they are empty or relatively sparse. Lastly, we move each particle and clear each bin in parallel for the next timestep.

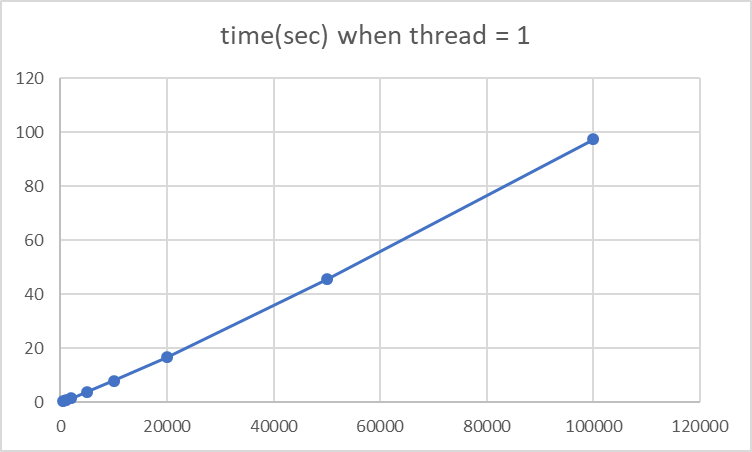
**Time taken for different number of particle in Open MP computation for different threads.**

Number of particles (n) is given in the x- axis

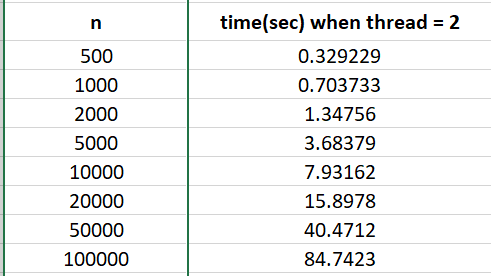
Time taken is given in the y- axis

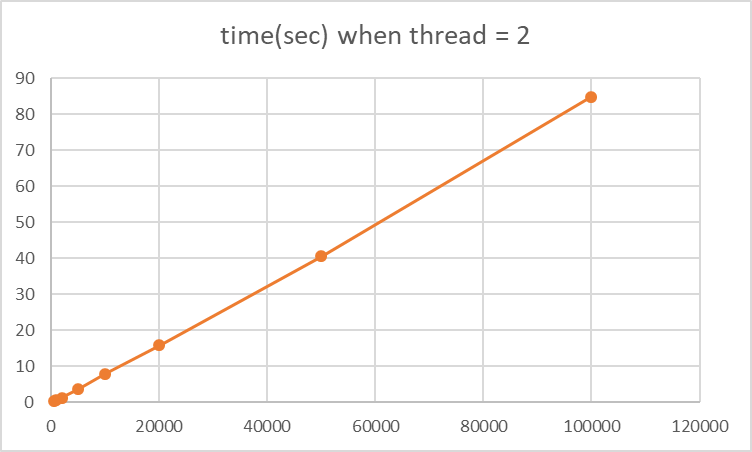
**When thread = 1**



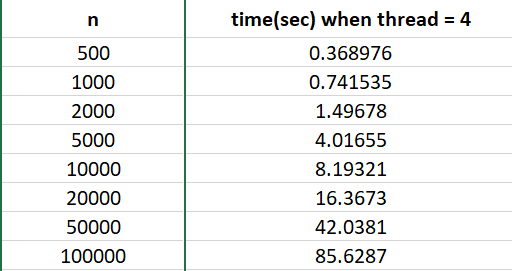


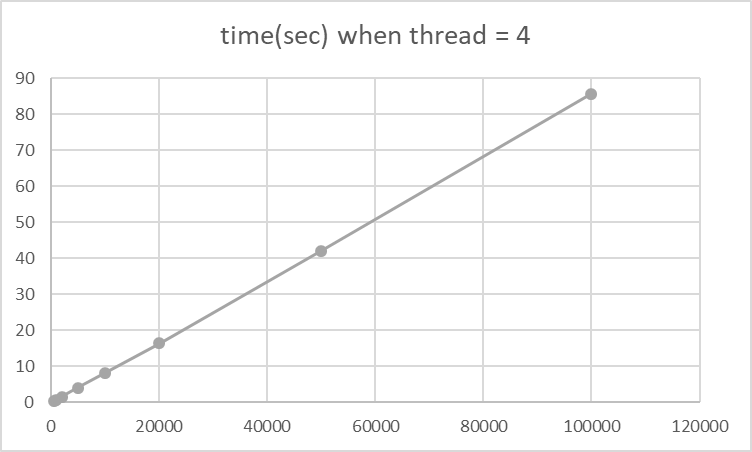
**When thread = 2**



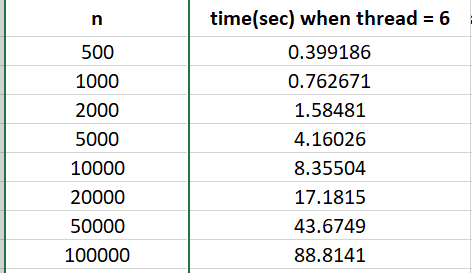


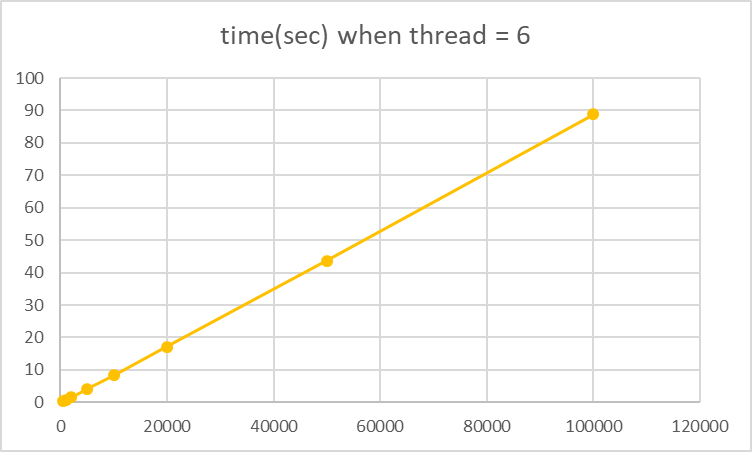
**When thread = 4**



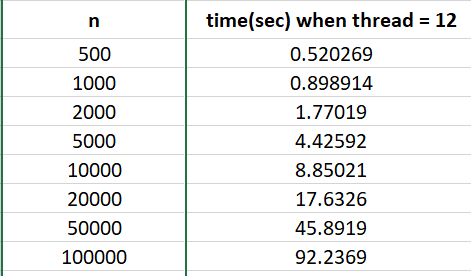


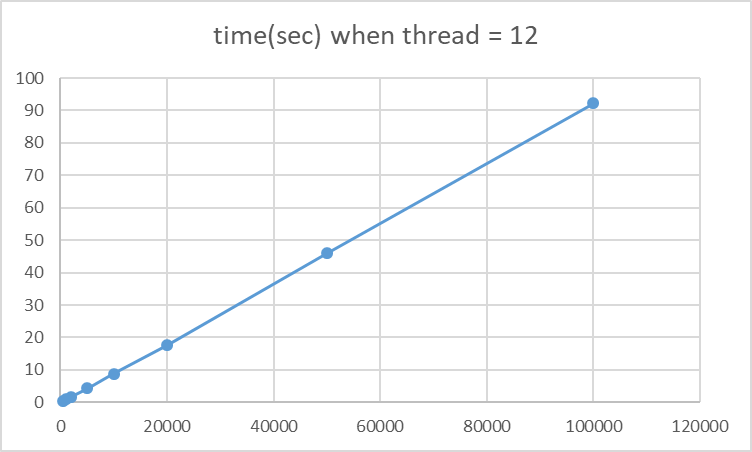
**When thread = 6**



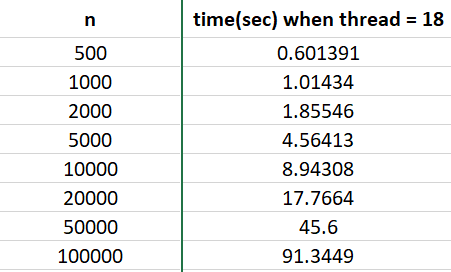


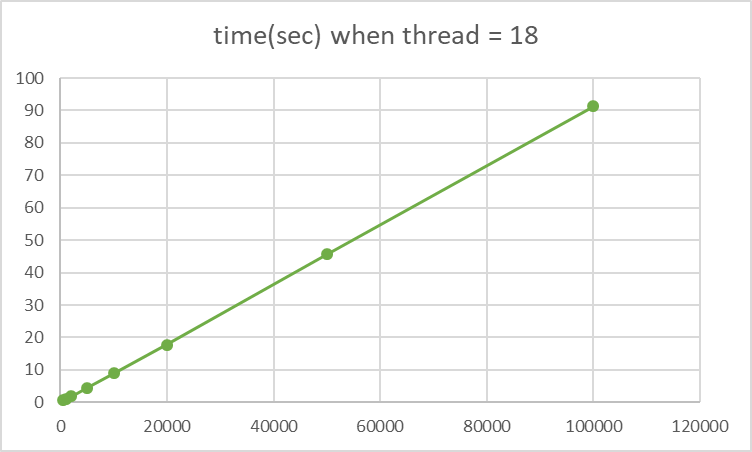
**When thread = 12**



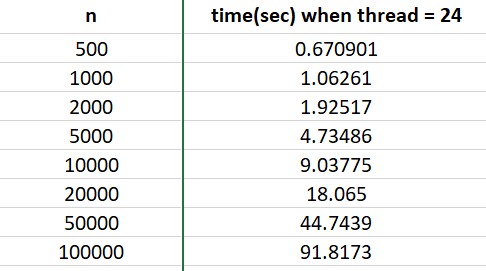


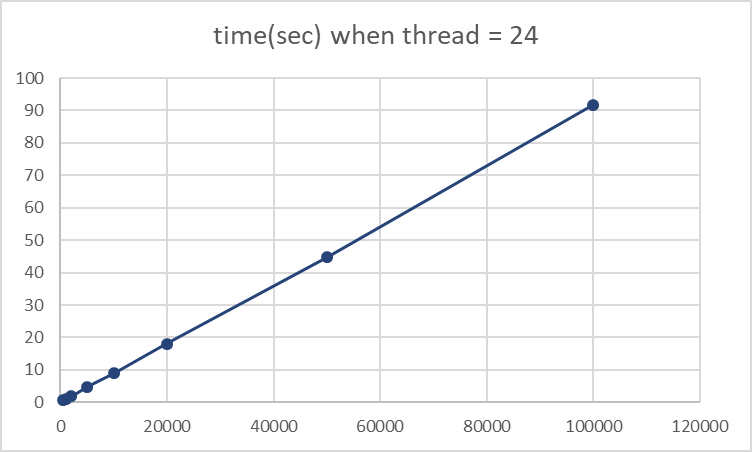
**When thread = 18**



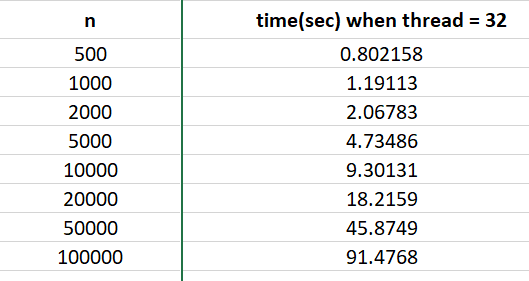


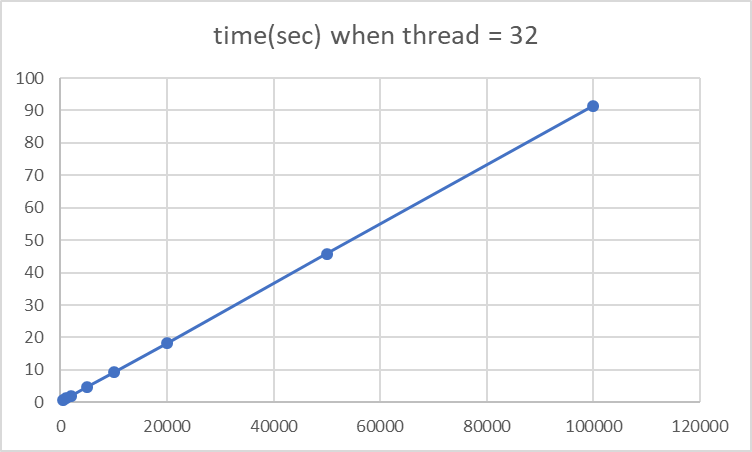
**When thread = 24**





**When thread = 32**





**3. Parallel Implementation using MPI**

If B is the number of bins, then we have a 2D grid with bxb distribution where, b = √B. Then we divide them by row. Suppose we have p processors, each processor will have b2 p = B p bins. We use the root node (rank is zero) to initiate the particles and then use MPI Scatterv to scatter the particles to all the nodes. Concretely, if a certain particle (pt) belongs to a bin (bi), and bi is owned by a certain node (nd), then pt will only be sent to nd. In each step, we need to make the bins in boundary exchange some particles with the bins in neighbor processor’s boundary. We use MPI Ibsend to conduct this kind of communication. For more details, please check exchange neighbors. Because we have got the boundary information from the neighboring node, we can conduct force computation on each local node. Each node will get its own davg, navg, dmin variables. Then we have used MPI Reduce to get the global variables rdavg, rnavg, rdmin by MPI SUM, MPI SUM, and MPI MIN respectively.

Problems faced:

Major problems faced during writing the code when try to achieve communication between the processes.

It is difficult to understand what processor has to send/receive data in the right direction. We are not entirely sure if the code that we are submitting for MPI is correct.

Problems running the code: We could not run the code on crill cluster. We got an error saying that the user partition is not available to execute the code.

The values obtained without using mpirun command and by just using ./mpi command.

**Time taken for different number of particle in MPI computation.**

Number of particles (n) is given in the x- axis

Time taken is given in the y- axis

