O(n²) algorithms

Serial Algorithm
• Algorithm compares each particle with every other particle and checks for the interaction radius.
• Most comparisons are outside interaction range.

OpenMP, Pthreads
• Algorithm simply splits the particles into p equal parts and accesses all non-owned particles in the interaction check.

MPI Algorithm
• Code gathers all particles on local node and then compares with local particles.

Big Idea
• Before implementing parallelism work out best serial algorithm.

Legend
- Current particle
- Actual neighbor
- Checked particle
- Non-Checked particle
- Interaction Radius
O(n) serial algorithm – “Binning”

**Main idea**
- Since all far-field interactions are ignored, creating a “local neighborhood” through binning can alleviate most of the unnecessary checks since all grey particles can be ignored.

**Time complexity**
- Checking only neighboring “bins“ reduces the checks from $O(n)$ for each particle to $O(9d)$ where $d$ is the average density of particles in each cell.
- Since it is said within the statement of the problem that density is uniform and it can be seen from the problem code that the domain size (x by y) actually increases to maintain a constant density we can consider the $d$ number to be a small constant (in practice 3-5 for a bin size = interaction radius).
- Overall complexity becomes $O(9d \cdot n) = O(n)$.
O(n) serial algorithm – “Binning”

*Implementation details*

• Particles need to be assigned to bins at every timestep which presents at least two different options:
  • deleting list and rebinning every timestep (depending on how bins are implemented potentially timeconsuming)
  • maintaining bins and moving particles. (Can create allot of overhead in checking for new particles being added to your bin – particles may “jump” past neighbor bin)

*Common problems*

• If particles seem to be accelerating most likely scenario is that interactions are not happening
• In case of 2nd implementation of binning remember to check for total particle count and ensure no particles are lost in moving
**O(n) Shared memory Implementation**

*Race conditions*
- While the algorithm doesn’t change much from the serial the biggest challenge is avoiding excessive synchronization between threads while ensuring that all threads are on the same step of the algorithm (ex: rebinning, calculating forces, moving particles, etc.) or at least that they are not too far ahead to risk correctness.

*Common problems*
- Deadlock between competing threads for a particular set of bins if locking not implemented carefully (avoid acquiring all locks needed at once).
- Inaccurate results from race conditions for updating particle acceleration (harder to spot).
- Slower performance than serial because of excessive synchronization.

**Legend**
- Current particle
- Actual neighbor
- Checked particle
- Non-Checked particle
- Interaction Radius
- Local Bin
O(n) MPI Implementation

Implementation options
• Can implement as shared memory implementation with messages instead of shared variables
• Second implementation splits particles based on location onto processors besides bins and must implement proper particle movement between processors

Common problems
• Takes much longer to code correctly
• Deadlock problems can occur if implemented with blocking send/receive pairs
• Particles not interacting with neighboring bins from other processors (both N,S,E,W and diagonally)
• Particles disappearing at processor borders

Legend
- Current particle
- Actual neighbor
- Checked particle
- Non-Checked particle
- Interaction Radius
- Local Bin
- Processor Boundary