Homework 1

Performance results
(pretty good!)
Loop Blocking
- AVX
- padding/packing/copy optimization
- Alignment

- Loop unrolling
- Compiler flags
- Graphs
- Machine comparison
- Discuss future work
Homework 2
Improving the algorithm

Serial Algorithm
• Algorithm compares each particle with every other particle and checks for the interaction radius

Big Idea to go faster
• Most comparisons are outside interaction range as shown in figure
• Bin the particles into $p$ equal sized subsets, only check neighbors

Before parallelism work out best serial algorithm!
Faster 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 number of particles in each cell (assume bin size >= cutoff).
• Since it is said within the statement of the problem that density is uniform and we can see in the common files that the domain size increases to maintain a constant density we can consider the d number to be a small constant (In practice d<3-4 for bin size ~ cutoff).
• Overall complexity becomes O(9d 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:
  1) deleting list and rebinning every timestep (depending on how bins are implemented potentially time-consuming)
  2) maintaining bins and moving particles. (Can create a lot of overhead in checking for new particles being added to your bin – particles may “jump” past neighbor bin)

*Common problems*

- **Accelerating particles:**
  Interactions are probably not happening when they should be

- **Disappearing particles:**
  Check for total particle count and ensure no particles are lost in moving (common in 2\textsuperscript{nd} implementation of binning)
Going parallel

**OpenMP, Pthreads**
- Split the particles into $p$ equal sized subsets and accesses all non-owned particles in the interaction check

**MPI Algorithm**
- Code gathers all neighboring particles on local node and then compares with local particles.

**Get serial algorithm first!**
- Algorithm doesn’t change much, (e.g. rebinning, calculating forces, moving particles, etc.) just how work is shared

---

**Legend**
- Current particle
- Actual neighbor
- Checked particle
- Non-Checked particle
- Interaction Radius
**Common problems**

- **Dead lock**
  Threads compete for a particular set of bins. Implement locks carefully and avoid acquiring multiple locks.

- **Race conditions**
  Inaccurate results from race conditions for updating particle acceleration (harder to spot). Use enough locks and synchronization when updating shared.

- **Slowdown**
  Slower performance than serial. Avoid excessive synchronization between threads while ensuring that all threads are on the same step of the algorithm.

---

Legend

- **Current particle**
- **Actual neighbor**
- **Checked particle**
- **Non-Checked particle**

- **Interaction Radius**
- **Local Bin**
**O(n) MPI Implementation**

**Implementation options**
- Analogous to shared memory implementation with messages instead of shared variables
- Split particles based on location onto processors in addition to bins. 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
- Must deal with ghostzones

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