Parallel Monte Carlo Simulation of Colloidal Crystals

Jeff Boghosian
Faculty Advisor: Dr. Talid Sinno
Special Thanks: Raynaldo Scarlett

Abstract:
- Develop a parallel version of Monte Carlo simulation to analyze colloidal crystal growth.

Serial Monte Carlo algorithm:
- At each step, loop through the entire list of particles.
- For each particle, give it a random displacement below some threshold distance $d_{	ext{max}}$.
- Calculate the resulting change in energy, $\Delta E$.
- If $\Delta E < 0$ (favorable), accept the displacement.
- If $\Delta E > 0$, accept the displacement with probability $\exp(-\Delta E/kT)$.
- Otherwise, reject the move and return the particle to its old position.

Verlet Neighbor Lists:
- Simply checking each particle against every other particle would require $n^2$ calculations.
- Use a Verlet list to store the neighbors of each particle in a giant array.
- When calculating energies for a particle, it is only necessary to look at the particles in the neighbor list.
- This reduces calculations from $n^2$ to $np$, where $p$ is the average number of neighbors for each particle.
- Add a skin thickness so neighbor lists do not need to be calculated every step (10-20 in practice).

Parallelization issues with spatial subdivision:
- Spatial subdivision – each processor controls the particles within a region of space.
- This should make parallelization easy, but issues arise with particles near the border between two cells.
- Consider two particles very close to each other, but owned by different cells.
  - Race conditions – both particles could be moved at the same time before the processors notify each other.
  - Which particle should be moved first?
  - Some sort of synchronization would be necessary, but at each step this would cripple performance.

Subdivision strategy:
- The simulation cube is broken down into cells, with each cell being assigned to a processor.
- Cells are further broken down into subcells. An extra 2 cells are added in each dimension to keep track of neighboring particles.
- The yellow subcells in each cell are the ones controlled by that processor.
- White cells are copies of the nearest layer of subcells surrounding the cell.
  - For example, cells 10, 15, and 20 in Cell 1 are copies of cells 7, 12, and 17 in Cell 2. These copies of subcells can become stale, so they have to be updated when necessary.
  - Keeping these fine-grained subcells also eliminates the need for Verlet neighbor lists – all of a particles neighbors can be found in the 3*3*3 cube of subcells centered around the particle.

Subcell-level Parallelization:
- In order to limit network traffic, one subcell is chosen and processed by each cell in parallel. Every particle in that subcell is sampled.
- Since the chosen subcell won't be neighboring its counterpart in other cells, there is no synchronization problem.
- Updated particle positions for the selected subcell are sent after the entire subcell has been processed. Internal subcells whose borders are entirely within the cell do not need to be sent.
- Every subcell is processed this way, and the order is chosen randomly to eliminate regional bias.
- This should lead to significant reductions in network traffic.

Abstract:
- Develop a parallel version of Monte Carlo simulation to analyze colloidal crystal growth.

Subdivision strategy:
- The simulation cube is broken down into cells, with each cell being assigned to a processor.
- Cells are further broken down into subcells. An extra 2 cells are added in each dimension to keep track of neighboring particles.

Subcell-level Parallelization:
- In order to limit network traffic, one subcell is chosen and processed by each cell in parallel. Every particle in that subcell is sampled.
- Since the chosen subcell won't be neighboring its counterpart in other cells, there is no synchronization problem.
- Updated particle positions for the selected subcell are sent after the entire subcell has been processed. Internal subcells whose borders are entirely within the cell do not need to be sent.
- Every subcell is processed this way, and the order is chosen randomly to eliminate regional bias.
- This should lead to significant reductions in network traffic.

Graphical User Interface:
- A GUI was created to monitor the development of the system.
- A separate component from the simulation - output files from the core simulation are read by the GUI.
- Used the Java Swing API for the GUI, with OpenGL integration using the JOGL libraries.
- Liquid particles are displayed as small dots, solids as large spheres.
- Multiple files can be read by the simulator, allowing one to see growth over time.
- Can export videos of the crystal from any angle and change particle colors and sizes.

Conclusions:
- Theoretically, spatial subdivision is a good way to cut down running times.
- Not known if spatial randomness is preserved – does subcell-level processing introduce a bias?
- Subcells are processed in random order to hopefully eliminate this issue.
- GUI is a very useful tool in analyzing the simulation run – can easily see what is happening to the system – growth, dissolution, simulation problems.