OpenMP-enabled Simulation of Biological Crystal Growth via Diffusion-Limited Aggregation

Overview

The basic idea is to write an OpenMP-based multi-threaded simulation of the physical process known as diffusion-limited aggregation (DLA).

Background and Specifications

Diffusion-limited aggregation was proposed by Witten and Sander as a way to explain (and model) the propensity of metal ions in battery fluid near an electrode to aggregate into a crystal structure (a phenomenon known as electrodeposition).

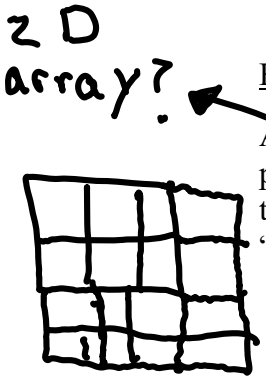
It has also been proposed as a technique used to model the growth of crystals in a liquid. In this case, simple diffusion is implemented by inducing particles to conduct a "random walk", simulating the Brownian motion they would experience when suspended in a liquid. The particles tend to aggregate into crystals as they adhere when coming into contact with a "seed" particle (or the cluster it has formed).

One example of biological crystal growth occurs in the disease called gout (a form of arthritis). In human physiology, a compound called uric acid is formed as a byproduct of the breakdown of purines, notable examples of which are the nucleotides adenine and guanine. Afflicted patients typically have elevated uric acid levels. The excess uric acid precipitates into crystals typically found in joints, causing the painful and characteristic symptom of this disease.

Fluid flow in a biological system is usually modeled using the Stokes equations:

$$\nabla p \approx \mu \nabla^2 u + f$$

where p is the pressure gradient, μ is the fluid viscosity, **u** is the velocity vector, and **f** is the force vector. The equations direct the computation of fluid gradients and movements at each spatial location. However, it is mathematically difficult to incorporate solid objects into the computations. Fortunately, for purposes of a simulation, the fluid can be modeled as a lattice: a large, two-dimensional (or three-dimensional) grid of points or cells. Particles are then introduced into the system and move discretely among the cells of the simulated space.



Basic Idea:

A 2D/3D space is defined, with a single seed particle placed at the origin. New particles are introduced into the system, and move randomly through the liquid. If they encounter a fixed particle, they adhere (becoming fixed). If they ever walk "outside" of the lattice, they disappear. Here is a more detailed view of the algorithm:

Algorithm

what is a "Particle". can it be any random data type

Initialization:

- Place a seed particle at origin, clear all lattice points, set Radius to 0.

center of 2D Array?

For <number of particles> in simulation:

- Introduce a new particle at a distance of at least (Radius + 1) from the origin.
- Perform a random walk with the new particle:
  - If it comes into contact with a fixed particle that is part of the growing crystal, it sticks. Contact is defined as adjacency in any of the defined directions of the simulation. Sticking is defined as becoming fixed. That lattice point is updated appropriately. The Radius is also updated (if necessary).
  - If the new particle ever walks outside the lattice space, or exceeds some maximum number of moves, it disappears from the simulation.

does this mean the particle can only move to an index adjacent to it?

The process continues until there are no new particles to introduce.

The results are then visualized. (Note: real uric acid crystals are needle-like, due to the chemistry involved. This is not required for the visualization.)

One characteristic of this problem to exploit when implementing a concurrent solution is the independence of particle movements. They may be performed in any order and still be correct (i.e. there are only *local* data dependencies). Of course, you still have to isolate old/new values that represent lattice state.

There are also a number of different ways to implement concurrency. For example, are various threads responsible for a pre-defined portion of the lattice? Is a thread spawned and associated with each new particle? Put some thought into your implementation.

Requirements / Deliverables:

Perform a multi-threaded simulation of DLA.

- □ Use the OpenMP API to create a multi-threaded solution. Several shared-memory platforms (eos, arch) are available.
- □ Try to create a visualization of the final state of your simulation. Any package, framework, language is acceptable.
- □ Instrument and time the core of your simulation code (the actual simulation, not writing out results or visualizing points). Conduct an analysis of the effect of threading your code.
- □ Submit a hard-copy of your source-code, visualization, and analysis. Be prepared to present your results in class.