# CS 5220

Parallelism and Locality in Simulations

David Bindel

2024-09-17

# Intro

# Parallelism and Locality

The world exhibits *parallelism* and *locality*

- Particles, people, etc function independently
- Near-field interactions stronger than far-field
- Can often simplify dependence on distant things

# Parallelism and Locality

Get more parallelism / locality through model

- Limited dependency between adjacent time steps
- Can neglet or approximate far-field effects

Often get parallelism at multiple levels

- Hierarchical circuit simulation
- Interacting models for climate
- Parallelizing individual experiments in MC or optimization

# Styles of Simulation

- Discrete event systems (continuous or discrete time)
- Particle systems
- Lumped parameter models (ODEs)
- Distributed parameter models (PDEs / IEs)

Often more than one type of simulation is approprate.

(Sometimes more than one at a time!)

# Discrete Event Systems

# Discrete Event Systems

May be discrete or continuous time.

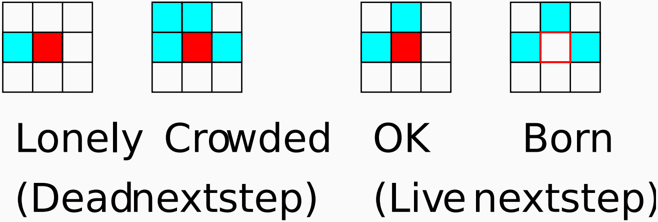
- Game of life
- Logic-level circuit simulation
- Network simulation

# Discrete Events

- Finite set of variables, transition function updates
- Synchronous case: finite state machine
- Asynchronous case: event-driven simulation
- Synchronous (?) example: Game of Life
- Nice starting point – no discretization concerns!

# Game of Life

Game of life (John Conway):



# Game of Life

Game of life (John Conway):

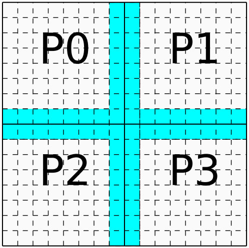
- Live cell dies with < 2 live neighbors
- Live cell dies with > 3 live neighbors
- Live cell lives with 2-3 live neighbors
- Dead cell becomes live with exactly 3 live neighbors

What to do if I really cared?

- Tile the problem for memory
- Try for high operational intensity
- Use instruction-level parallelism
- Don't output board too often!

Before doing anything with OpenMP/MPI!

East to parallelize by *domain decomposition*




- Update work involves *volume* of subdomains
- Communication per step on *surface* (cyan)

Also works with tiling.

Sketch of a kernel for tiled implementation:

- Bitwise representation of cells (careful with endian-ness)
- A “tile” is a 64-by-64 piece (64 `uint64_t`)
    - Keep two tiles (`ref` and `tmp`)
- Think of inner 48-by-48 as “live”
- Buffer of size 8 on all sides
- Compute saturating 3-bit neighbor counters
- Batches of eight steps (four `ref` to `tmp`, four back)

Some areas are more eventful than others!

What if pattern is dilute?

- Few or no live cells at surface at each step
- Think of live cell at a surface as an “event”
- Only communicate events!
  - This is *asynchronous*
  - Harder with message passing – when to receive?

# Asynchronous Life

How do we manage events?

- Speculative – assume no communication across boundary for many steps, back up if needed
- Conservative – wait when communication possible
    - Possible $\neq$ guaranteed!
    - Deadlock: everyone waits for a send
    - Can get around this with NULL messages

How do we manage load balance?

- No need to simulate quiescent parts of the game!
- Maybe dynamically assign smaller blocks to processors?

- There are also other algorithms!

# Beyond Life

- Forest fire model
- ns-3 network simulator
- Digital hardware

# Particle Systems

- Billiards, electrons, galaxies, ...
- Ants, cars, agents, ...?

Particles move via Newton ($F = ma$) with

- External forces: ambient gravity, currents, etc
- Local forces: collisions, Van der Waals ($r^{-6}$), etc
- Far-field forces: gravity and electrostatics ($r^{-2}$), etc
    - Simple approximations often apply (Saint-Venant)

# Forced Example

$$
\begin{aligned}
f_i &= \sum_j G m_i m_j \frac{(x_j - x_i)}{r_{ij}^3} \left(1 - \left(\frac{a}{r_{ij}}\right)^4\right), \\
r_{ij} &= \|x_i - x_j\|
\end{aligned}
$$

- Long-range attractive force ($r^{-2}$)
- Short-range repulsive force ($r^{-6}$)
- Go from attraction to repulsion at radius $a$

# Simple Serial Simulation

Using `Boost.Numeric.Odeint`, we can write

```
integrate(particle_system, x0, tinit, tfinal, h0,
          [](const auto& x, double t) {
              std::cout << "t=" << t << ": x=" << x << std::e
          });
```

where

- `particle_system` defines the ODE system
- `x0` is the initial condition
- `tinit` and `tfinal` are start and end times
- `h0` is the initial step size

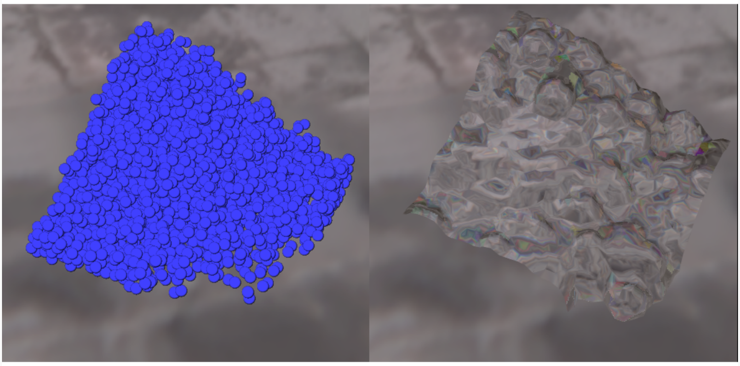
and the final lambda is an *observer* function.

## Beyond Serial Simulation

Can parallelize in

- Time (tricky): Parareal methods, asynchronous methods
- Space: Our focus!

# Plotting Particles



Smooth Particle Hydrodynamics (SPH) – Project 2

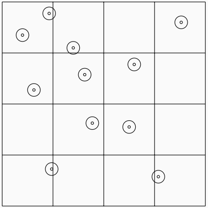
- Where do particles “live” (distributed mem)?
    - Decompose in space? By particle number?
    - What about clumping?
- How are long-range force computations organized?
- How are short-range force computations organized?
- How is force computation load balanced?
- What are the boundary conditions?
- How are potential singularities handled?
- Choice of integrator? Step control?

# External Forces

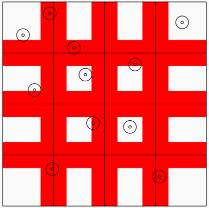
Simplest case: no particle interactions.

- Pleasingly parallel (like Monte Carlo!)
- Could just split particles evenly across processors
- Is it that easy?
  - Maybe some trajectories need short time steps?
  - Even with MC, load balance may not be trivial!

# Local Forces



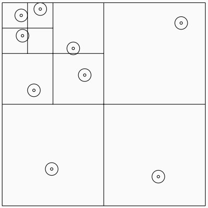
- Simplest all-pairs check is $O(n^2)$ (expensive)
- Or only check close pairs (via binning, quadtrees?)
- Communication required for pairs checked
- Usual model: domain decomposition



Minimize communication:

- Send particles that might affect a neighbor “soon”
- Trade extra computation against communication
- Want low surface area-to-volume ratios on domains

# Local Forces: Load Balance

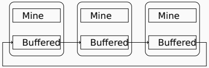


- Are particles evenly distributed?
- Do particles remain evenly distributed?
- Can divide space unevenly (e.g. quadtree/octtree)

# Far-Field Forces

- Every particle affects every other particle
- All-to-all communication required
  - Overlap communication with computation
  - Poor memory scaling if everyone keeps everything!
- Idea: pass particles in a round-robin manner

# Passing Particles (Far-Field Forces)




```
copy particles to current buf
for phase = 1 to p
  send current buf to rank+1 (mod p)
  recv next buf from rank-1 (mod p)
  interact local particles with current buf
  swap current buf with next buf
end
```

# Passing Particles (Far-Field Forces)

Suppose $n = N/p$ particles in buffer. At each phase

$$\begin{aligned} t_{\text{comm}} &\approx \alpha + \beta n \\ t_{\text{comp}} &\approx \gamma n^2 \end{aligned}$$

So mask communication with computation if

$$n \geq \frac{1}{2\gamma}\left(\beta + \sqrt{\beta^2 + 4\alpha\gamma}\right).$$

## Passing Particles (Far-Field Forces)

More efficient serial code

$\Longrightarrow$ larger $n$ needed to mask commujnication!

$\Longrightarrow$ worse speed-up as $p$ gets larger (fixed $N$)

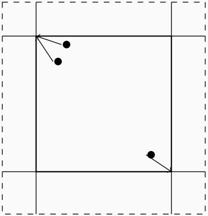
but scaled speed-up ($n$ fixed) remains unchanged.

# Far-Field Forces: Particle-Mesh

Consider $r^{-2}$ electrostatic potential interaction

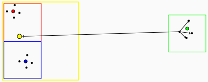
- Enough charges look like a continuum!
- Poisson maps charge distribution to potential
- Fast Poisson for regular grids (FFT, multigrid)
- Approx depends on mesh and particle density
- Can clean up leading part of approximation error

# Far-Field Forces: Particle-Mesh



- Map particles to mesh points (multiple strategies)
- Solve potential PDE on mesh
- Interpolate potential to particles
- Add correction term – acts like local force

# Far-Field Forces: Tree Methods



- Distance simplifies things
    - Andromeda looks like a point mass from here?
- Build tree, approx descendants at each node
- Variants: Barnes-Hut, FMM, Anderson's method
- More on this later in the semester

# Summary of Particle Example

- Model: Continuous motion of particles
  - Could be electrons, cars, whatever
- Step through discretized time

# Summary of Particle Example

- Local interactions
  - Relatively cheap
  - Load balance a pain
- All-pairs interactions
  - Obvious algorithm is expensive ($O(n^2)$)
  - Particle-mesh and tree-based algorithms help

An important special case of lumped/ODE models.