Lecture 10:
Parallelism and Locality in Scientific Codes

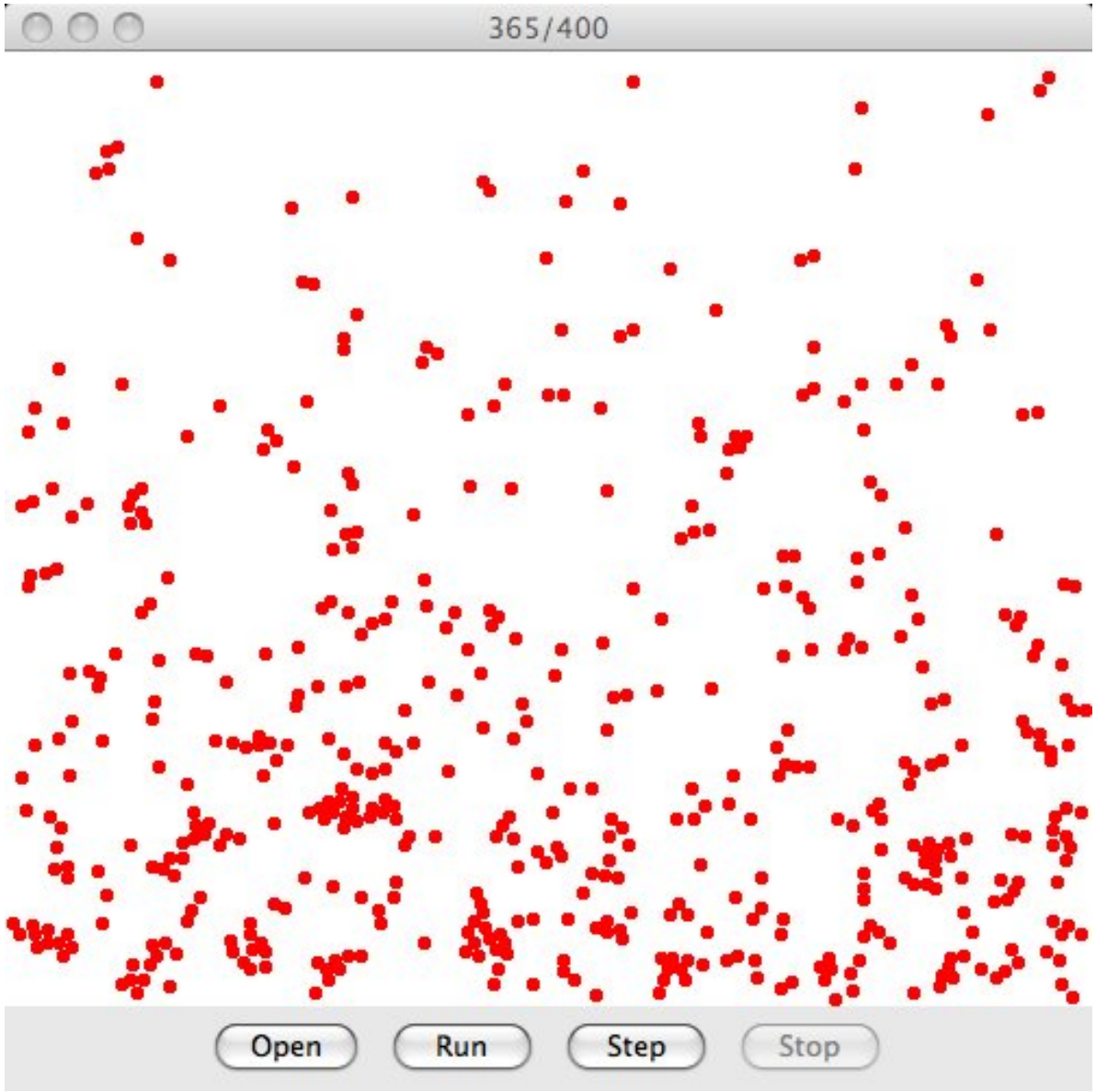
David Bindel

22 Feb 2010

Logistics

- HW 2 posted – due March 10.
  - Groups of 1–3; use the wiki to coordinate.
- Thinking about projects:
  - Small teams (2–3, 1 by special dispensation)
  - *Understanding* performance, tuning, scaling is key
  - Feel free to leverage research, other classes (with approval)!
  - Want something high quality...
    but also something you can finish this semester!
  - Ideas...

HW 2 discussion



(On board / screen)

HW 2

1. Time the baseline code.
   - How does the timing scale with the number of particles?
   - How does the timing scale with the number of processors?
   - How well is the serial code performing?
2. Use spatial decomposition to accelerate the code.
   - Example: bin sort the particles into grid squares and only compare neighboring bins (could also do other spatial data structures, neighbor lists, etc)
   - What speedup do you see vs the original code?
   - How does the scaling change in the revised code?
   - What should the communication change?
3. Time permitting: do some fun extension!
   - Is the code “right”? What are the numerical properties?
   - Can you improve the time integration?
   - Can you further tune the inner loops (e.g. with SSE)?

Basic styles of simulation

- Discrete event systems (continuous or discrete time)
  - Game of life, logic-level circuit simulation
  - Network simulation
- Particle systems (our homework)
  - Billiards, electrons, galaxies, ...
  - Ants, cars, ...?
- Lumped parameter models (ODEs)
  - Circuits (SPICE), structures, chemical kinetics
- Distributed parameter models (PDEs / integral equations)
  - Heat, elasticity, electrostatics, ...

Often more than one type of simulation appropriate.
Sometimes more than one at a time!

Common ideas / issues

- Load balancing
  - Imbalance may be from lack of parallelism, poor distributin
  - Can be static or dynamic
- Locality
  - Want big blocks with low surface-to-volume ratio
  - Minimizes communication / computation ratio
  - Can generalize ideas to graph setting
- Tensions and tradeoffs
  - Irregular spatial decompositions for load balance at the cost of complexity, maybe extra communication
  - Particle-mesh methods — can't manage moving particles and fixed meshes simultaneously without communicating

Lumped parameter simulations

Examples include:

- SPICE-level circuit simulation
  - nodal voltages vs. voltage distributions
- Structural simulation
  - beam end displacements vs. continuum field
- Chemical concentrations in stirred tank reactor
  - concentrations in tank vs. spatially varying concentrations

Typically involves ordinary differential equations (ODEs),
or with constraints (differential-algebraic equations, or DAEs).

Often (not always) *sparse*.

Sparsity

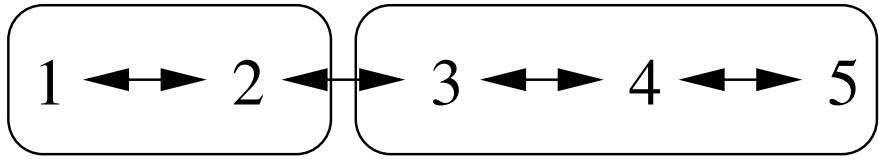
$$A = \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$

Consider system of ODEs $x' = f(x)$ (special case: $f(x) = Ax$)

- Dependency graph has edge (i, j) if f_j depends on x_i
- Sparsity means each f_j depends on only a few x_i
- Often arises from physical or logical locality
- Corresponds to A being a sparse matrix (mostly zeros)

Sparsity and partitioning

$$A = \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$



Want to partition sparse graphs so that

- Subgraphs are same size (load balance)
- Cut size is minimal (minimize communication)

We'll talk more about this later.

Types of analysis

Consider $x' = f(x)$ (special case: $f(x) = Ax + b$). Might want:

- Static analysis ($f(x_*) = 0$)
  - Boils down to $Ax = b$ (e.g. for Newton-like steps)
  - Can solve directly or iteratively
  - Sparsity matters a lot!
- Dynamic analysis (compute $x(t)$ for many values of t)
  - Involves time stepping (explicit or implicit)
  - Implicit methods involve linear/nonlinear solves
  - Need to understand stiffness and stability issues
- Modal analysis (compute eigenvalues of A or $f'(x_*)$)

Explicit time stepping

- Example: forward Euler
- Next step depends only on earlier steps
- Simple algorithms
- May have stability/stiffness issues

Implicit time stepping

- Example: backward Euler
- Next step depends on itself and on earlier steps
- Algorithms involve solves — complication, communication!
- Larger time steps, each step costs more

A common kernel

In all these analyses, spend lots of time in sparse matvec:

- Iterative linear solvers: repeated sparse matvec
- Iterative eigensolvers: repeated sparse matvec
- Explicit time marching: matvecs at each step
- Implicit time marching: iterative solves (involving matvecs)

We need to figure out how to make matvec fast!