Lecture 11: Parallelism and Locality in Scientific Codes

David Bindel

1 Mar 2010

Logistics

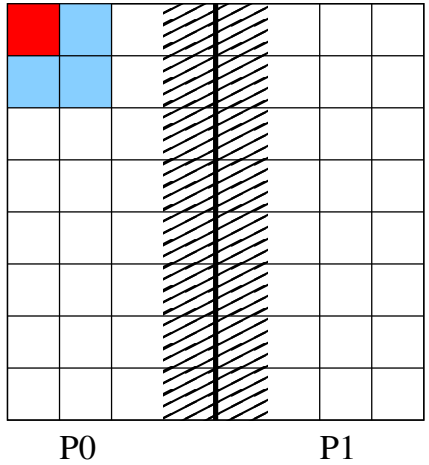
Thinking about projects:

- Informal proposal in one week (March 8)
- Free-for-all at end of class

HW 2 notes (how I spent my weekend)

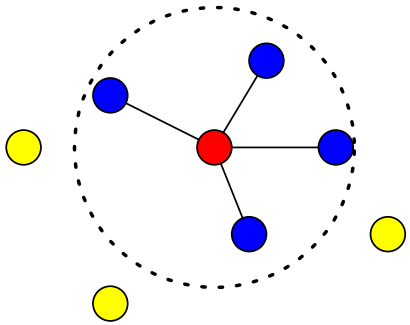
- Code started at zero velocity — fixed now!
- Issues with `-cwd` on cluster?
  - Symptom: status `Eqw` on queue
  - Solution: `qdel` stalled job, add `cd $HOME/my/path` at start of script, and resubmit

HW 2 notes (how I spent my weekend)

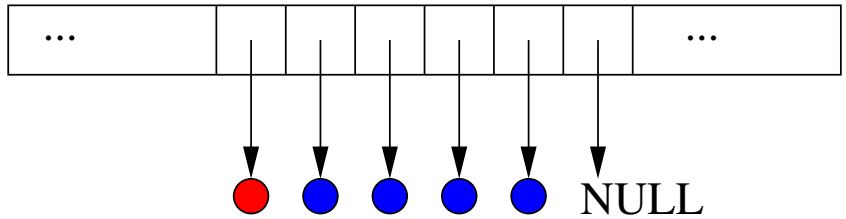


- Huge improvement from binning (and some tuning)
  - Test case: 500 particles, 20000 time steps
  - Baseline serial code: 45 s
  - Improved serial code: 1 s (even better for more particles!)
- “Obvious” parallel decomposition slowed down the code!
  - Partition space
  - Evaluate independently away from boundary region
  - Evaluate boundary region independently
  - ... got killed by parallel overhead!

HW 2 notes (how I spent my weekend)



Particle neighborhood



Particle neighbor list

- Did a little reading (Plimpton 1995, JCP)
  - Neighbor lists are a good idea (recompute every few steps)
    - ... and profiling showed lots of time in bin lookups
  - Could partition by space, by atom, or by force
  - By force is attractive for small particle counts!
- Switched to neighbor list — 2× speedup!
  - ... but still trying to get my shared memory code as fast
  - Had to backtrack to slower code to figure out parallelism

HW 2 notes (how I spent my weekend)

Current best guess on the final version

- Re-tuned neighbor list access
- Batch communications with extra computation (MPI)
  - Multiple steps with particle subset between communication
  - Time step with 500 particles == 2.5 message latencies!
    (at least in the tuned code)
  - Changes propagate by one neighbor / step
  - Redundant computation to avoid communication?
  - Will see this idea later today

Some notes on debugging tools

Debugging tools

- `printf` — the classic
- `assert` — equally classic!
- GCC `-g` flag — include debug symbol information
- `valgrind` — catch memory errors (useful!)
- `gdb` — step through code, find error locations

All the above at least work with threaded code...

Some notes on profiling tools

Profiling tools

- Timers
  - `MPI_Wtime`, `omp_get_wtime`
  - `clock_gettime`, `gettimeofday`
- Shark — awesome, but OS X only
- VTune — Intel's proprietary profiler
- `callgrind` — cycle timing on emulated CPU (20 $\times$ $--$ 100$\times$ slowdown)
- `google-perftools` — Google it!
- `kcachegrind` — visualize `callgrind` (or other) output
- `gprof` and `-pg` — classic

Note: Optimizing compilers may hide calls by inlining.

Recap

Basic styles of simulation:

- Discrete event systems (continuous or discrete time)
- Particle systems (our homework)
- Lumped parameter models (ODEs)
- Distributed parameter models (PDEs / integral equations)

Common issues:

- Load balancing
- Locality
- Tensions and tradeoffs

Today: Distributed parameter models

but first...

Reminder: sparsity

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

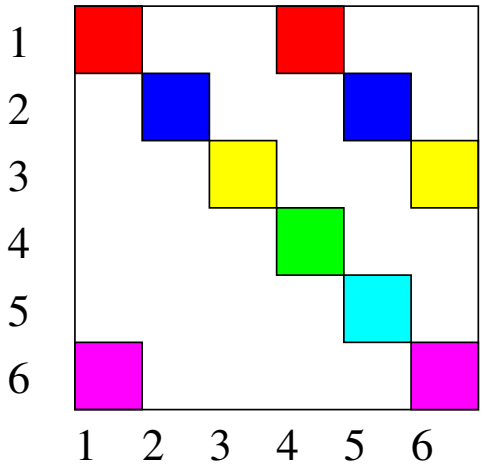
Consider system of ODEs $x' = f(x)$ (special case: $f(x) = Ax$)

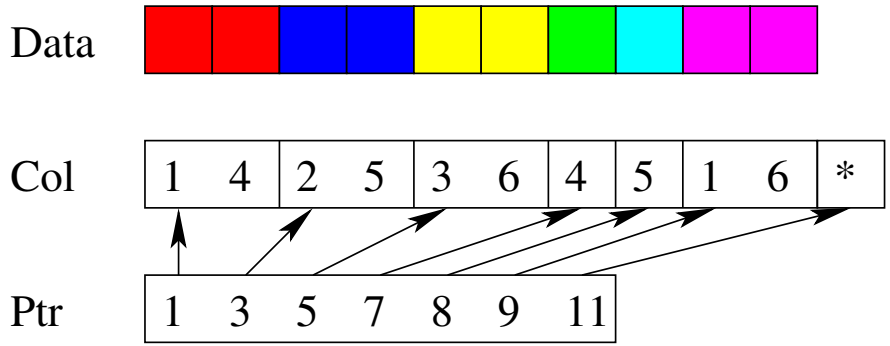
- Dependency graph has edge (i, j) if f_j depends on x_i
- Sparsity means each f_j depends on only a few x_i
- Often arises from physical or logical locality
- Corresponds to A being a sparse matrix (mostly zeros)

An aside on sparse matrix storage

- Sparse matrix $\implies$ mostly zero entries
  - Can also have "data sparseness" — representation with less than $O(n^2)$ storage, even if most entries nonzero
- Could be implicit (e.g. directional differencing)
- Sometimes explicit representation is useful
- Easy to get lots of indirect indexing!
- Compressed sparse storage schemes help

Example: Compressed sparse row storage





This can be even more compact:

- Could organize by blocks (block CSR)
- Could compress column index data (16-bit vs 64-bit)
- Various other optimizations — see OSKI

Distributed parameter problems

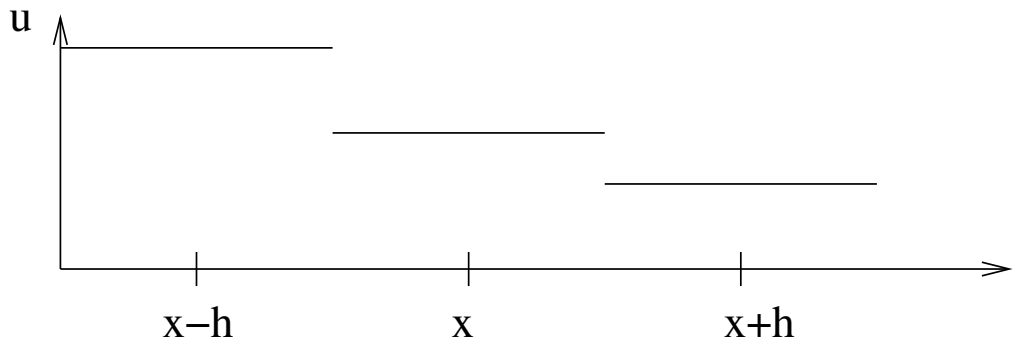
Mostly PDEs:

Type	Example	Time?	Space dependence?
Elliptic	electrostatics	steady	global
Hyperbolic	sound waves	yes	local
Parabolic	diffusion	yes	global

Different types involve different communication:

- Global dependence $\Longrightarrow$ lots of communication (or tiny steps)
- Local dependence from finite wave speeds; limits communication

Example: 1D heat equation



Consider flow (e.g. of heat) in a uniform rod

- Heat (Q) $\propto$ temperature (u) $\times$ mass (ρh)
- Heat flow $\propto$ temperature gradient (Fourier's law)

$$\frac{\partial Q}{\partial t} \propto h\frac{\partial u}{\partial t} \approx C\left[\left(\frac{u(x-h)-u(x)}{h}\right)+\left(\frac{u(x)-u(x+h)}{h}\right)\right]$$

$$\frac{\partial u}{\partial t} \approx C\left[\frac{u(x-h)-2u(x)+u(x+h)}{h^2}\right] \rightarrow C\frac{\partial^2 u}{\partial x^2}$$

Spatial discretization

Heat equation with $u(0) = u(1) = 0$

$$\frac{\partial u}{\partial t} = C\frac{\partial^2 u}{\partial x^2}$$

Spatial semi-discretization:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{u(x-h) - 2u(x) + u(x+h)}{h^2}$$

Yields a system of ODEs

$$\frac{du}{dt} = Ch^{-2}(-T)u = -Ch^{-2}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}\begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}$$

Explicit time stepping

Approximate PDE by ODE system ("method of lines"):

$$\frac{du}{dt} = Ch^{-2}Tu$$

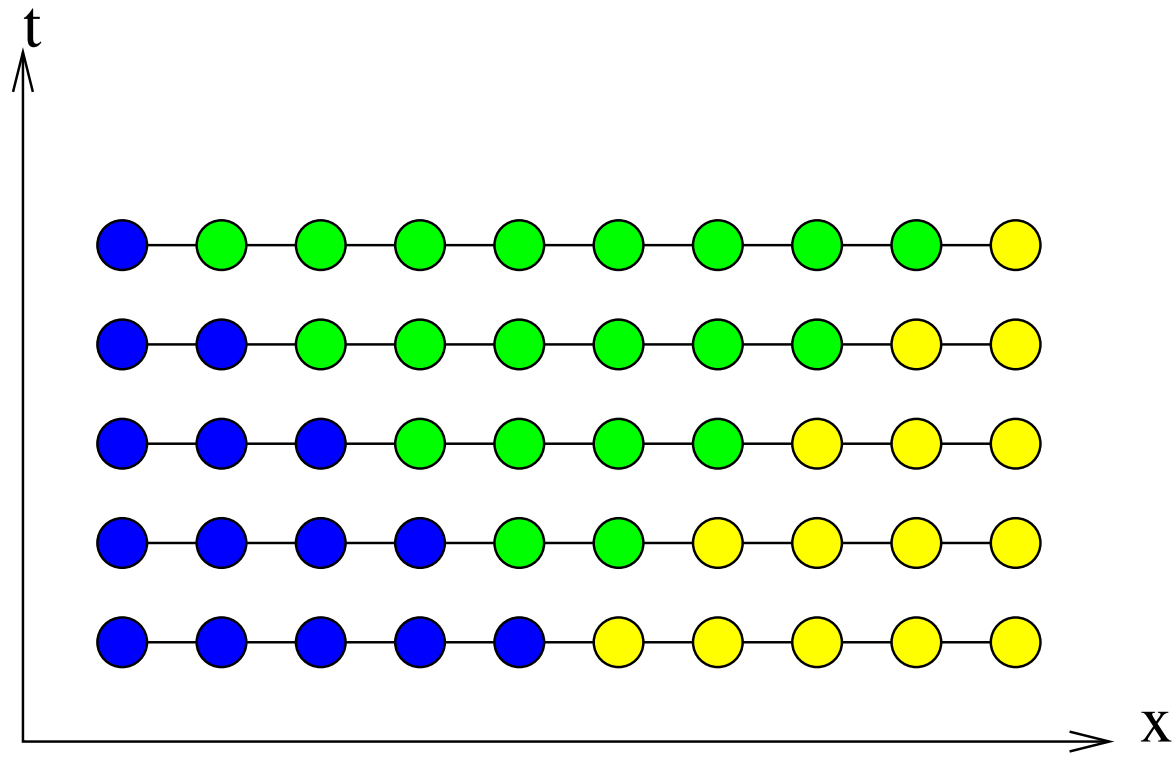
Now need a time-stepping scheme for the ODE:

- Simplest scheme is Euler:

$$u(t+\delta) \approx u(t) + u'(t)\delta = \left(I - C\frac{\delta}{h^2}T\right)u(t)$$

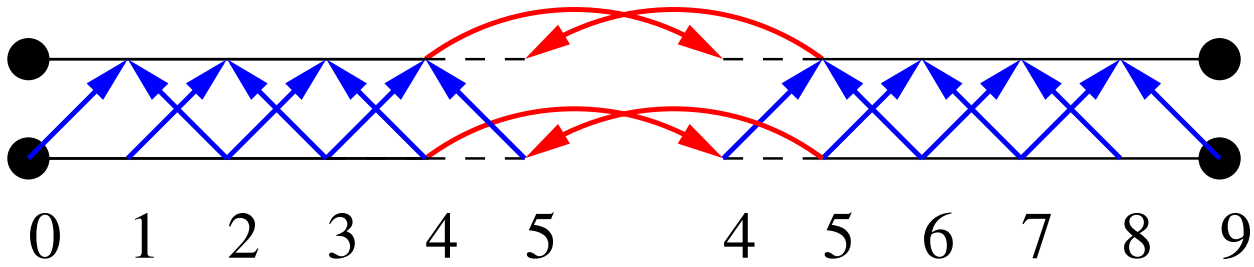
- Taking a time step $\equiv$ sparse matvec with $\left(I + C\frac{\delta}{h^2}T\right)$
- This may not end well...

Explicit time stepping data dependence



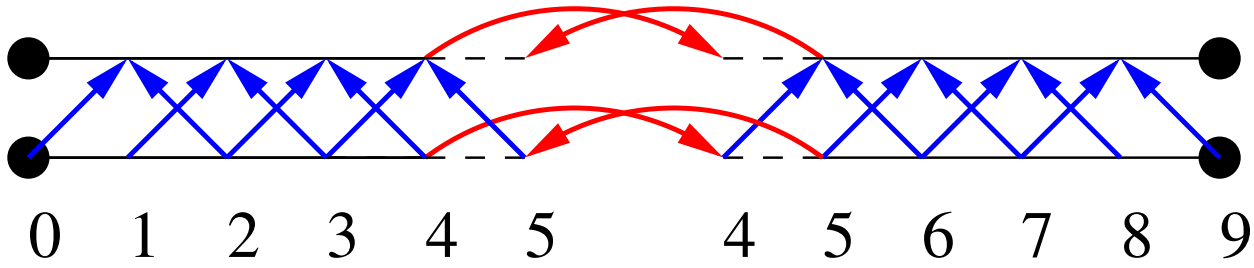
Nearest neighbor interactions per step $\Longrightarrow$
finite rate of numerical information propagation

Explicit time stepping in parallel



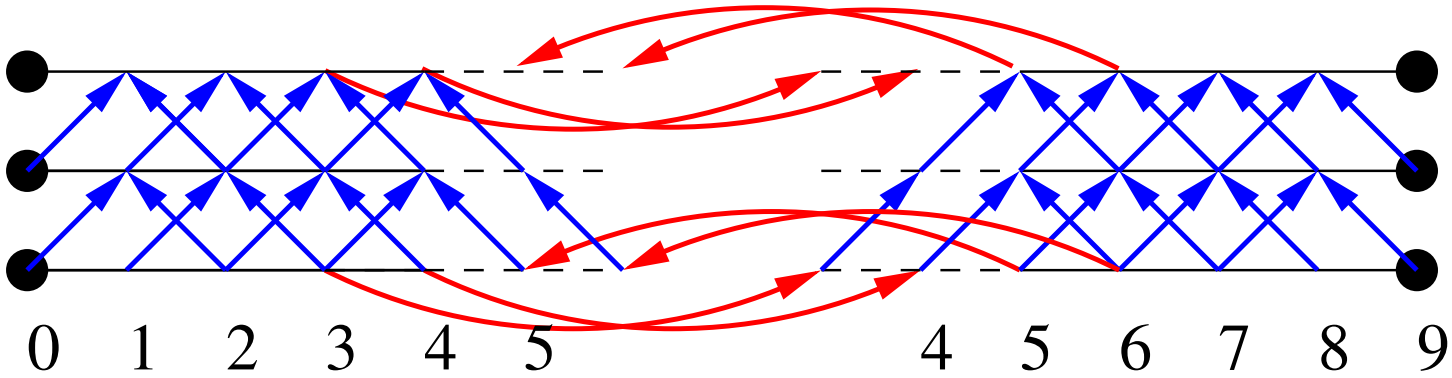

```
for t = 1 to N
  communicate boundary data ("ghost cell")
  take time steps locally
end
```

Overlapping communication with computation



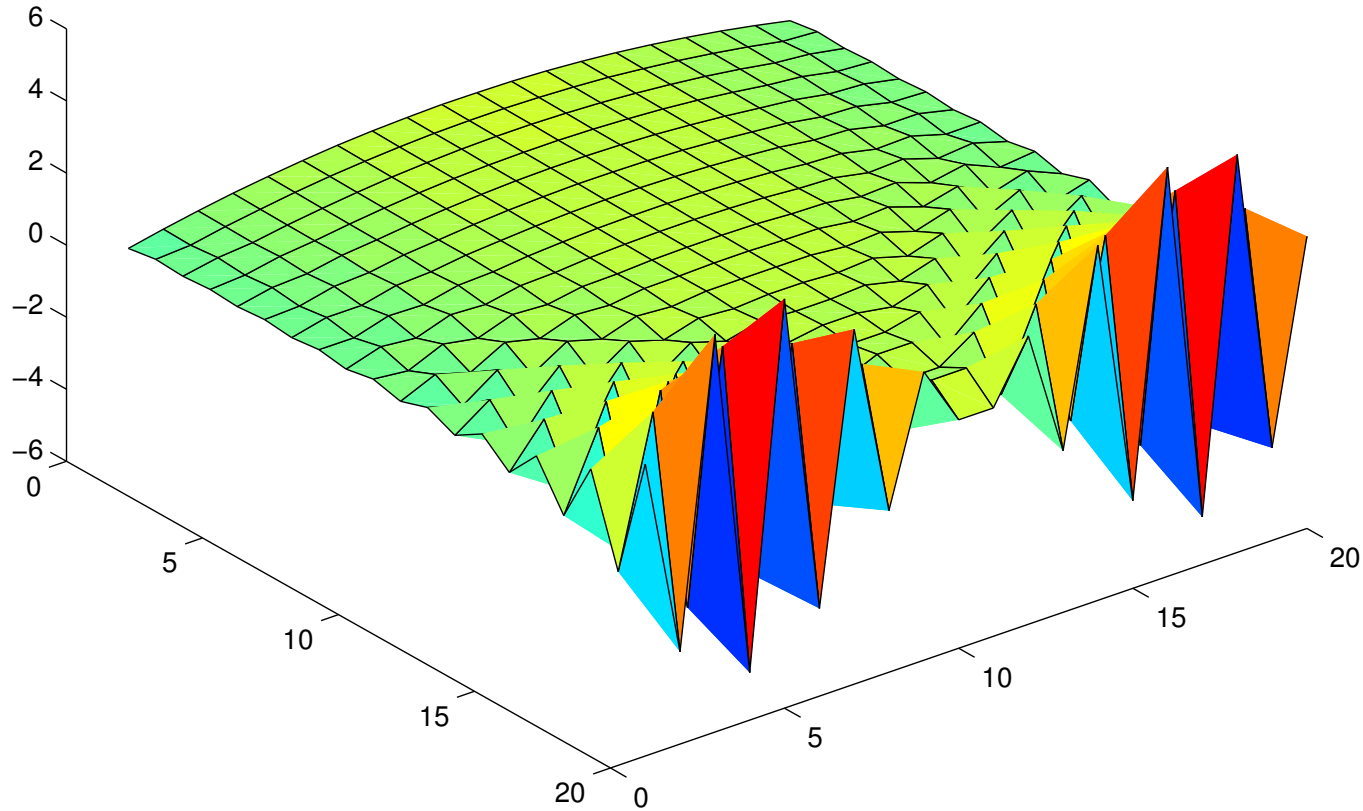

```
for t = 1 to N
  start boundary data sendrecv
  compute new interior values
  finish sendrecv
  compute new boundary values
end
```

Batching time steps




```
for t = 1 to N by B
   start boundary data sendrecv (B values)
   compute new interior values
   finish sendrecv (B values)
   compute new boundary values
end
```

Explicit pain



Unstable for $\delta > O(h^2)$!

Implicit time stepping

- Backward Euler uses backward difference for d/dt

$$u(t+\delta) \approx u(t) + u'(t+\delta t)\delta$$

- Taking a time step $\equiv$ sparse matvec with $\left(I + C\frac{\delta}{h^2}T\right)^{-1}$
- No time step restriction for stability (good!)
- But each step involves linear solve (not so good!)
  - Good if you like numerical linear algebra?

Explicit and implicit

Explicit:

- Propagates information at finite rate
- Steps look like sparse matvec (in linear case)
- Stable step determined by fastest time scale
- Works fine for *hyperbolic* PDEs

Implicit:

- No need to resolve fastest time scales
- Steps can be long... but expensive
  - Linear/nonlinear solves at each step
  - Often these solves involve sparse matvecs
- Critical for parabolic PDEs

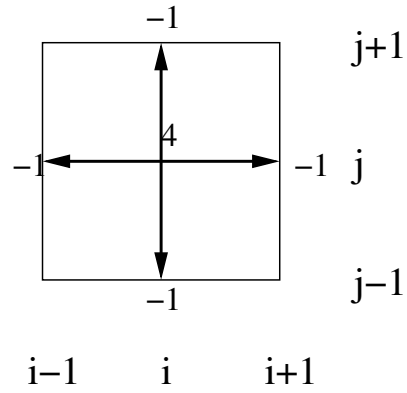
Poisson problems

Consider 2D Poisson

$$-\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f$$

- Prototypical elliptic problem (steady state)
- Similar to a backward Euler step on heat equation

Poisson problem discretization



$$u_{i,j} = h^{-2}\left(4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}\right)$$

$$L = \left[\begin{array}{ccc|ccc|ccc}
4 & -1 & & -1 & & & & & \\
-1 & 4 & -1 & & -1 & & & & \\
 & & -1 & & & -1 & & & \\
\hline
-1 & & & 4 & -1 & & -1 & & \\
 & -1 & & -1 & 4 & -1 & & -1 & \\
 & & -1 & & -1 & 4 & & & -1 \\
\hline
 & & & -1 & & & 4 & -1 & \\
 & & & & -1 & & -1 & 4 & -1 \\
 & & & & & -1 & & -1 & 4
\end{array}\right]$$

Poisson solvers in 2D/3D

$N = n^d$ = total unknowns

Method	Time	Space
Dense LU	N^3	N^2
Band LU	N^2 ($N^{7/3}$)	$N^{3/2}$ ($N^{5/3}$)
Jacobi	N^2	N
Explicit inv	N^2	N^2
CG	$N^{3/2}$	N
Red-black SOR	$N^{3/2}$	N
Sparse LU	$N^{3/2}$	$N \log N$ ($N^{4/3}$)
FFT	$N \log N$	N
Multigrid	N	N

Ref: Demmel, *Applied Numerical Linear Algebra*, SIAM, 1997.

Remember: best MFlop/s $\neq$ fastest solution!

General implicit picture

- Implicit solves or steady state $\implies$ solving systems
- Nonlinear solvers generally linearize
- Linear solvers can be
  - Direct (hard to scale)
  - Iterative (often problem-specific)
- Iterative solves boil down to matvec!

PDE solver summary

- Can be implicit or explicit (as with ODEs)
  - Explicit (sparse matvec) — fast, but short steps?
    - works fine for hyperbolic PDEs
  - Implicit (sparse solve)
    - Direct solvers are hard!
    - Sparse solvers turn into matvec again
- Differential operators turn into local mesh stencils
  - Matrix connectivity looks like mesh connectivity
  - Can partition into subdomains that communicate only through boundary data
  - More on graph partitioning later
- Not all nearest neighbor ops are equally efficient!
  - Depends on mesh structure
  - Also depends on flops/point

Project planning

- Should involve understanding performance
  - Could look at existing code on different platforms
  - Could apply performance modeling
  - Could involve tuning some kernel
- Should involve 2–3 people (one with special dispensation)
- May overlap research/classes (with approval)

Project idea free-for-all

Multiple people interested in

- Fluid simulations
- Solid mechanics / FEM (maybe with Diffpack)
- Monte Carlo simulations
- Computer graphics / rendering
- Computational sustainability

Have at it!