Lecture 5:
Parallelism and Locality in Scientific Codes

David Bindel

13 Sep 2011

Logistics

- Course assignments:
  - The cluster is online. Should receive your accounts today.
  - Short assignment 1 is due by Friday, 9/16 on CMS
  - Project 1 is due by Friday, 9/23 on CMS – find partners!
- Course material:
  - This finishes the “whirlwind tour” part of the class.
  - On Thursday, we start on nuts and bolts.
  - Preview of “lecture 6” is up (more than one lecture!)

Basic styles of simulation

- Discrete event systems (continuous or discrete time)
  - Game of life, logic-level circuit simulation
  - Network simulation
- Particle systems
  - 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

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

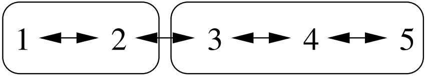
$$1 \longleftrightarrow 2 \longleftrightarrow 3 \longleftrightarrow 4 \longleftrightarrow 5$$

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

$$A = \left[\begin{array}{cc:ccc} * & * & & & \\ * & * & * & & \\ \hdashline & * & * & * & \\ & & * & * & * \\ & & & * & * \end{array}\right]$$



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:

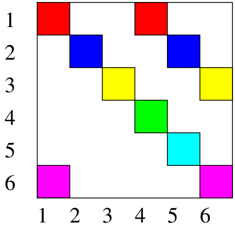
- Iterative linear solvers: repeated sparse matvec
- Iterative eigensolvers: repeated sparse matvec
- Explicit time marching: matvecs at each step
- Implicit time marching: iterative solves (involving matvecs)

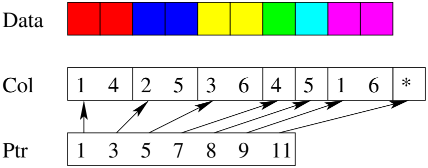
We need to figure out how to make matvec fast!

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

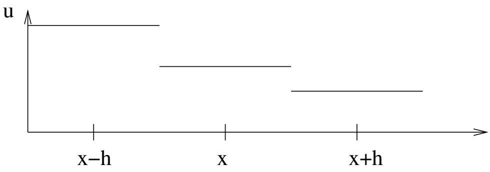
Mostly PDEs:

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 $\implies$ 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)

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

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

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"):

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

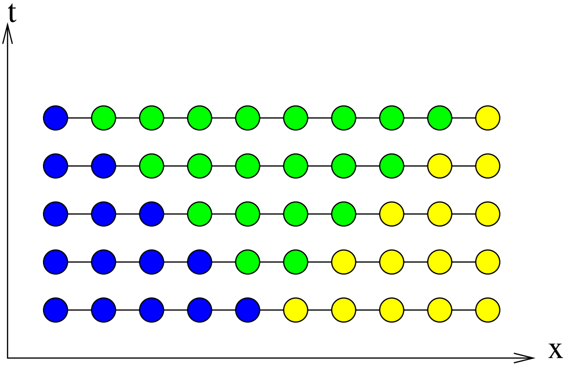
Now need a time-stepping scheme for the ODE:

- Simplest scheme is Euler:

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

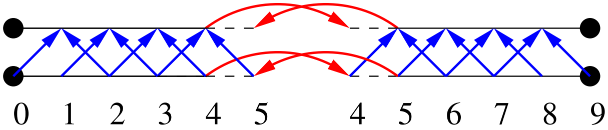
- Taking a time step $\equiv$ sparse matvec with $\left(I - C\frac{\delta}{h^2}T\right)$
- This may not end well...

Explicit time stepping data dependence



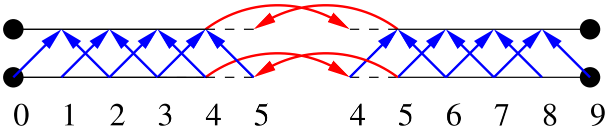
Nearest neighbor interactions per step $\Longrightarrow$
finite rate of numerical information propagation

Explicit time stepping in parallel



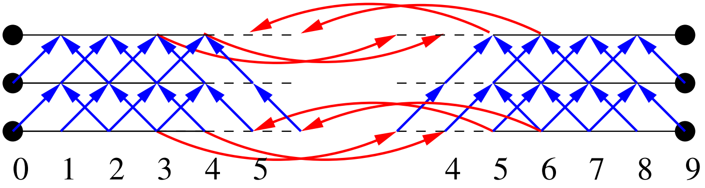

```
for t = 1 to N
  communicate boundary data ("ghost cell")
  take time steps locally
end
```

Overlapping communication with computation



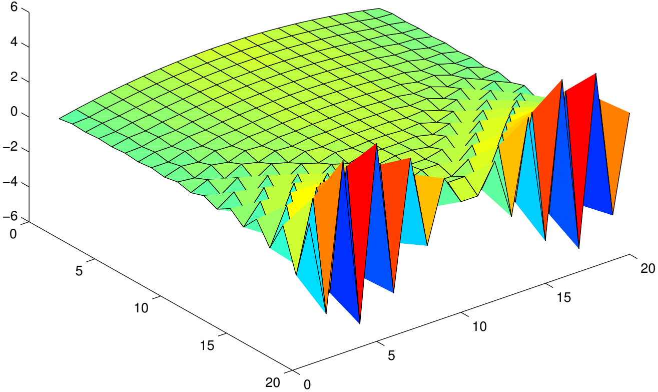

```
for t = 1 to N
  start boundary data sendrecv
  compute new interior values
  finish sendrecv
  compute new boundary values
end
```

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

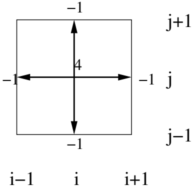
Poisson problems

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



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

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

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