CS 5220

Parallelism and Locality in Simulations

David Bindel

2024-09-17

Lumped Parameter Models

Lumped Parameter Simulations

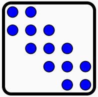
Examples include:

- 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 ordinary differential equations (ODEs)
- Constraints: differential-algebraic equations (DAEs)

Often (not always) *sparse*.

Sparsity

A = 

Matrix

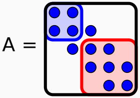
1 — 2 — 3 — 4 — 5

Graph

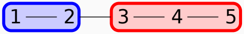
Consider ODEs $\dot{x} = f(x)$ (special case $f(x) = Ax$).

- Dependency graph: 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 sparse (mostly zeros)

Sparsity and Partitioning



Matrix



Graph

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.

Consider ODEs $\dot{x} = f(x)$ (special case $f(x) = Ax$).

Might want $f(x_*) = 0$.

- Boils down to $Ax = b$ (e.g. for Newton-like steps)
- Can solve directly or iteratively
- Sparsity matters a lot!

Consider ODEs $\dot{x} = f(x)$ (special case $f(x) = Ax$).

Might want $x(t)$ for many t given x_0

- Involves time-stepping (explicit or implicit)
- Implicit methods involve linear/nonlinear solves
- Need to understand stiffness and stability issues

Consider ODEs $\dot{x} = f(x)$ (special case $f(x) = Ax$).

Might want eigenvalues/vectors of A or $f'(x_*)$.

Explicit Time Stepping

- Example: forward Euler: $x_{k+1} = x_k + (\Delta t) f(x_k)$
- Next step depends only on earlier steps
- Simple algorithms
- May have stability issues with stiff systems

Implicit Time Stepping

- Example: backward Euler: $x_{k+1} = x_k + (\Delta t) f(x_{k+1})$
- 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 cases, 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!

Sparse 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

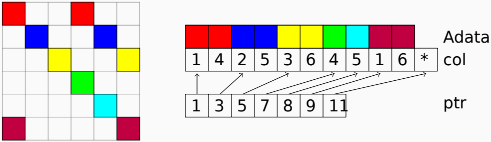


Figure 1: Illustration of compressed sparse row format

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

Summary

- ODE and DAE models widely used in engineering
- Different analyses: static, dynamic, modal
- Sparse linear algebra is often key

Distributed Parameter Models

Types of PDEs

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

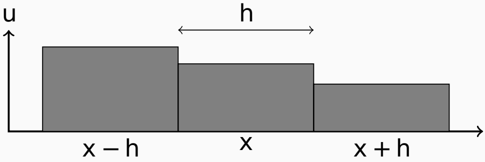
Types of PDEs

Different types involve different communication:

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

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

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



Example: 1D Heat Equation

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

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

$$
\begin{aligned}
\frac{\partial Q}{\partial t} &\propto h\frac{\partial u}{\partial t} \\
&\approx C\left[\frac{u(x-h)-u(x)}{h}+\frac{u(x+h)-u(x)}{h}\right] \\
&= C\left[\frac{u(x-h)-2u(x)+u(x+h)}{h^2}\right] \to C\frac{\partial^2 u}{\partial x^2}
\end{aligned}
$$

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 (second-order finite difference):

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

Spatial Discretization

Yields system of ODEs ("method of lines"):

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

$$T = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$$

Now need to time step!

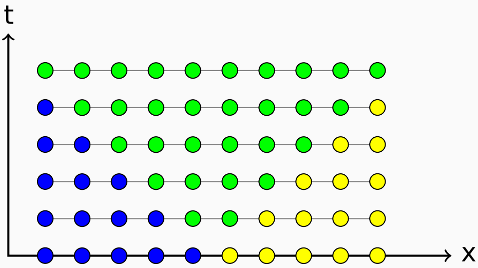
Explicit Time Stepping

- Simplest scheme is Euler:

$$u(t + \Delta t) \approx u(t) + u'(t)\Delta t = (I - Ch^2 T)u(t)$$

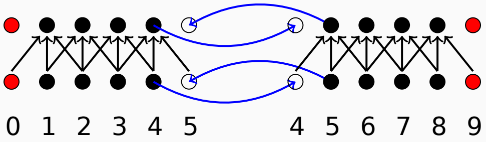
- Time step $\equiv$ sparse matvec with $(I - Ch^2 T)$
- This may not end well…

Explicit Data Dependence



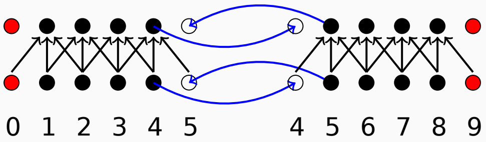
Nearest neighbor interactions per step $\implies$ finite rate of numerical information propagation

Explicit Time Stepping in Parallel



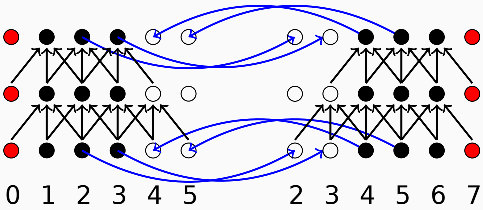

```
for t = 1 to N
  communicate boundary data ("ghost cell")
  take time steps locally
end
```

Overlapping Communication with Computation



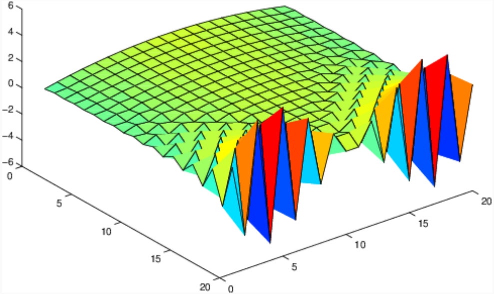

```
for t = 1 to N
  start boundary data sendrecv
  compute new interior values
  finish sendrecv
  compute new boundary values
end
```

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



Explicit Pain

- Unstable for $\Delta t > O(h^2)$
- Generally happens for parabolic (diffusive) equations
- But these ideas are great for hyperbolic equations!

Implicit Stepping

- Backward Euler: $u(t + \Delta t) \approx u(t) + \dot{u}(t + \Delta t)$
- Discretized time step: $u(t + \Delta T) = (I + Ch^2 T)^{-1} u(t)$
- No time step restriction for stabliity (good!)
- But each step involves a 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

Explicit and Implicit

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

Second-Order Finite Differences

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

Second-Order Finite Differences

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

Poisson Solvers in 2D/3D

$N = n^d$ total unknowns

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

Method	Time	Space
Dense LU	N^3	N^2
Band LU	N^2	$N^{3/2}$
Jacobi	N^2	N
Explicit inv	N^2	N^2

Poisson Solvers in 2D

Method	Time	Space
CG	$N^{3/2}$	N
Red-black SOR	$N^{3/2}$	N
Sparse LU	$N^{3/2}$	$N \log N$
FFT	$N \log N$	N
Multigrid	N	N

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

PDE Solver Summary

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

PDE Solver Summary

Not all nearest neighbor ops are equally efficient!

- Depends on mesh structure
- Also depends on flops/point

Onward!

- Next week: Distributed memory with MPI
- HW1 is posted: please run on Perlmutter!