

---

COSC 462

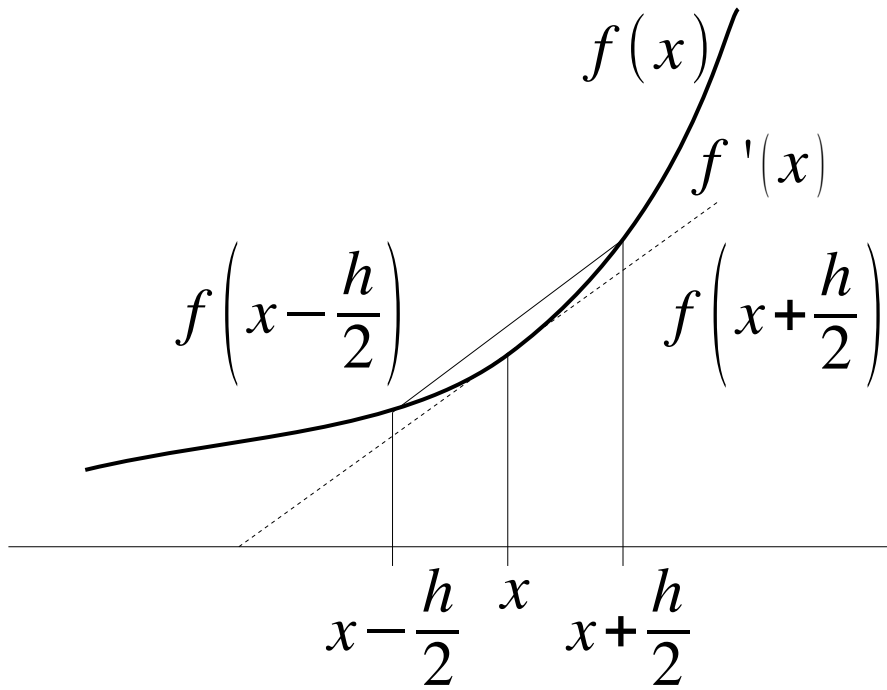
Finite Difference Methods

Piotr Luszczek

# Finite Difference Methods and PDEs

- Finite Difference Methods are commonly used to solve PDEs
- PDEs are used in many applications
  - Computational Fluid Dynamics
    - Water and gas flow
    - Multi-scale models
      - Weather prediction
  - Structural mechanics
    - Deformations of rigid structures
  - Wave propagation
    - Acoustics

# Approximating Derivative with Finite Difference



$$f'(x) = \frac{f(x+h/2) - f(x-h/2)}{h} + O(h^2)$$

$$f''(x) = \frac{f'(x+h/2) - f'(x-h/2)}{h} + O(h^2)$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

- The formulas assume that function  $f()$  is continuous
  - And so is its derivative  $f'()$

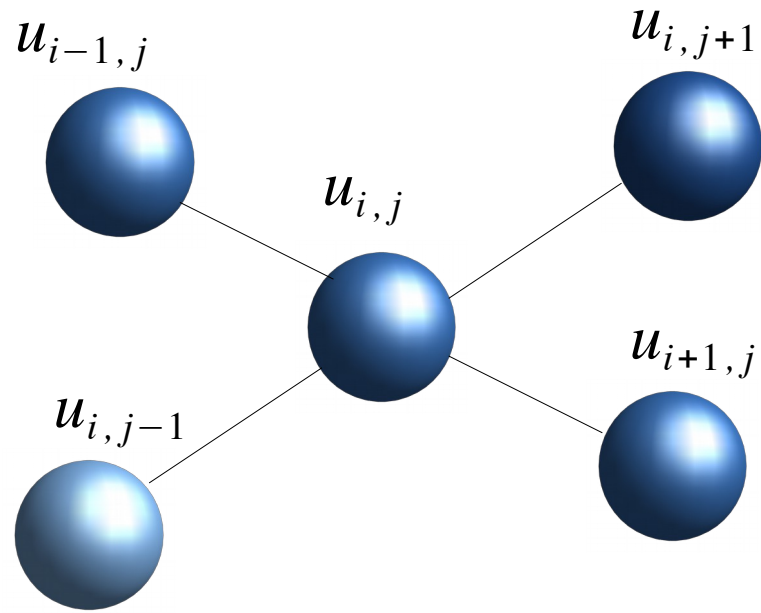
# Sample PDE: Poisson Equation

- Poisson equation has a simple form in 2D
  - $u_{xx} + u_{yy} = f(x, y)$
- Applications include
  - Electricity
  - Magnetism
  - Gravity
  - Heat distribution
  - Fluid flow
  - Torsion
- When  $f(x, y) = 0$  we call it Laplace equation

$$u_{xx} = \frac{\partial^2 u}{\partial x \partial x} \approx \frac{u(x+h, y) - 2u(x, y) + u(x-h, y)}{h^2}$$

# Mapping Formulas to Geometry

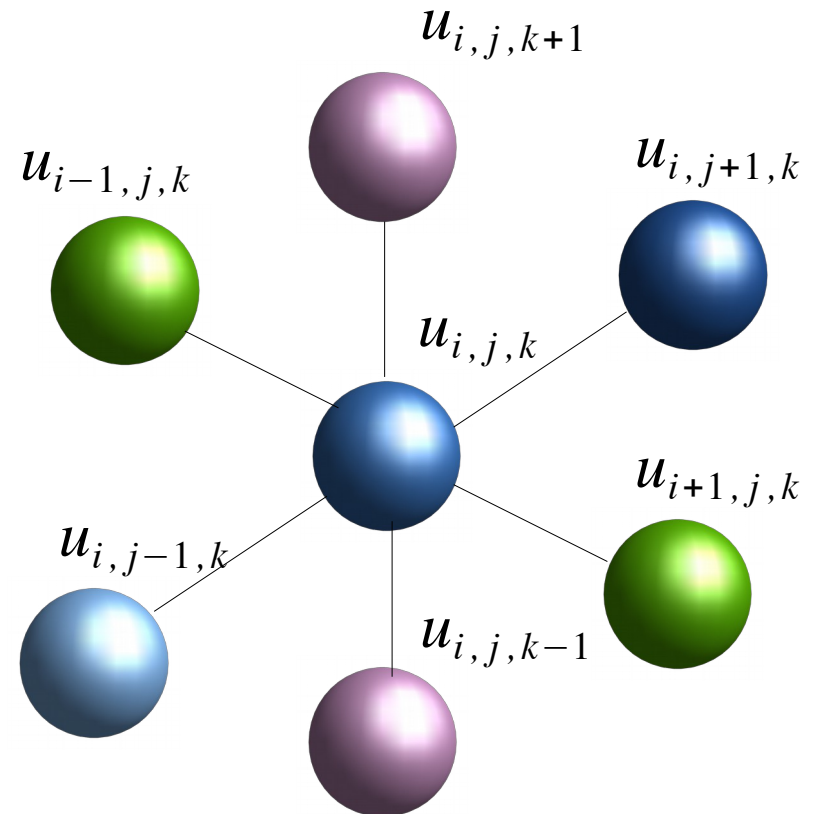
In 2D...



$$u_{xx} \approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2}$$

$$u_{yy} \approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2}$$

and in 3D...



$$u_{xx} + u_{yy} + u_{zz} = f(x, y, z)$$

# Iterating Towards Steady-State

Start with  $u_{i,j}$  estimates

$$\rightarrow \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f_{i,j}$$

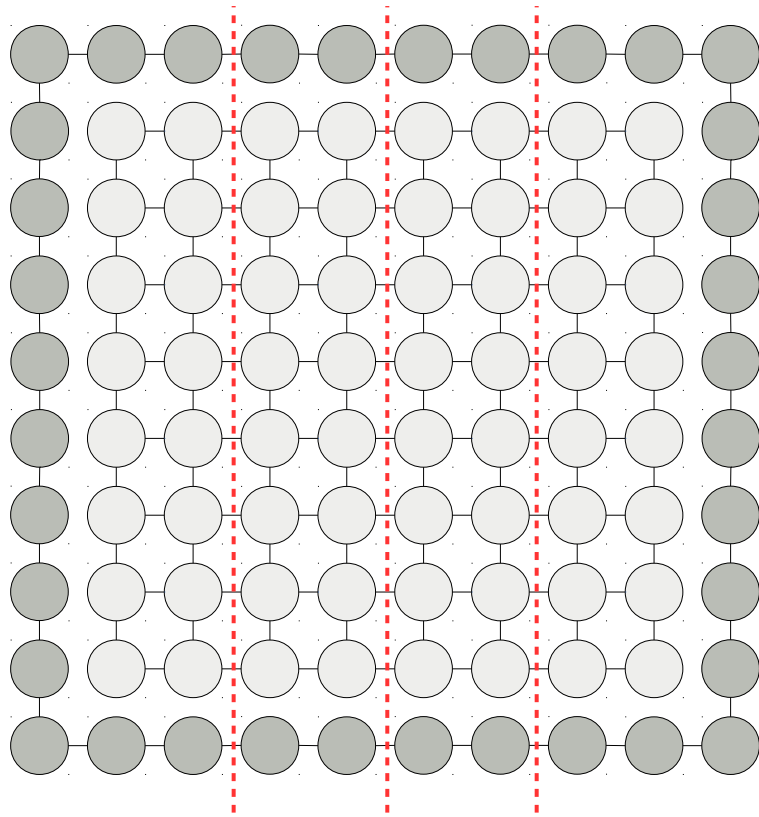
$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f_{i,j}$$

$$\frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f_{i,j}$$

Steady-state with final values of  $u_{i,j}$

$$\leftarrow \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h^2} + \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h^2} = f_{i,j}$$

# Meshes: Partitioning and Agglomeration

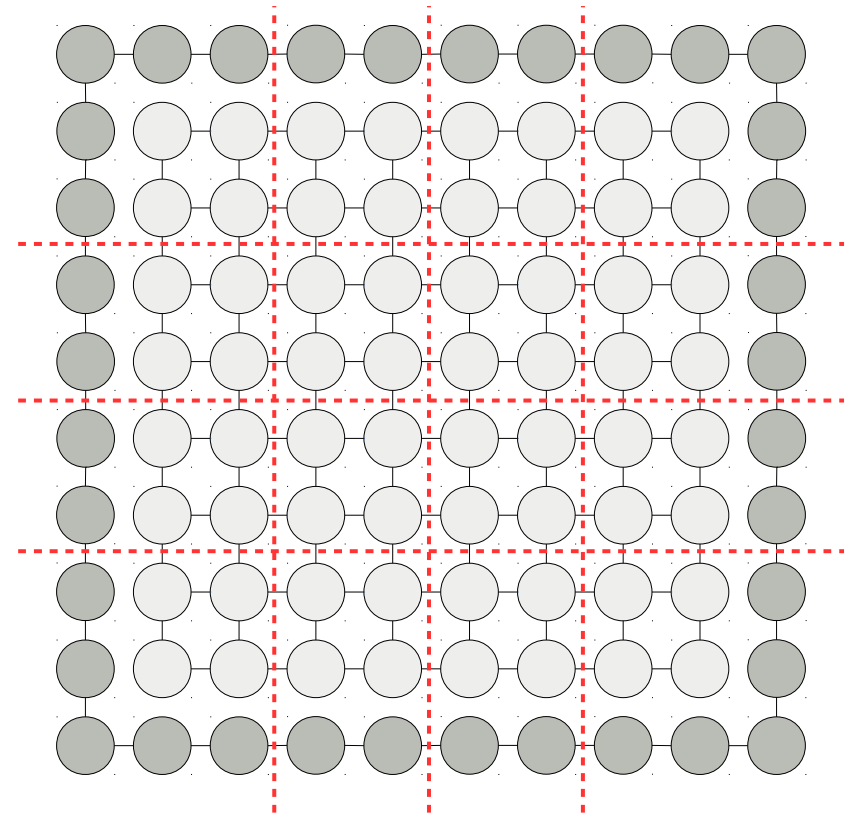


Computation:

$$\Theta(N^2/P)$$

Communication (N by N mesh):

$$\Theta(N)$$



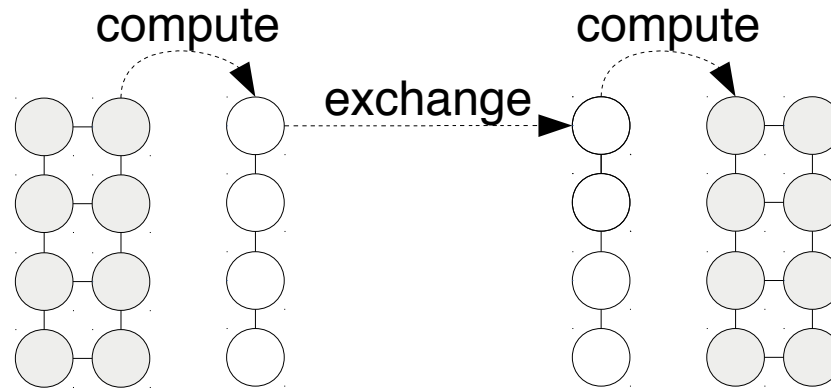
Computation:

$$\Theta(N/\sqrt{P} * N/\sqrt{P}) = \Theta(N^2/P)$$

Communication (N by N mesh):

$$\Theta(N/\sqrt{P})$$

# Implementation: Ghost Cells



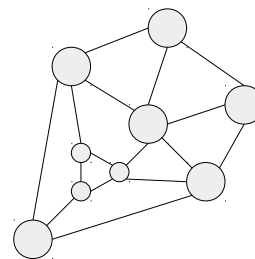
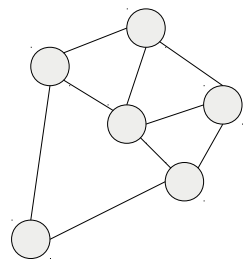
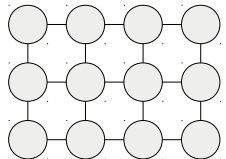
1. Compute on local cells
2. Compute on ghost cells
3. Exchange ghost cells
4. If not converged GOTO 1

This is usually combined in a clever implementation  
Communication is local

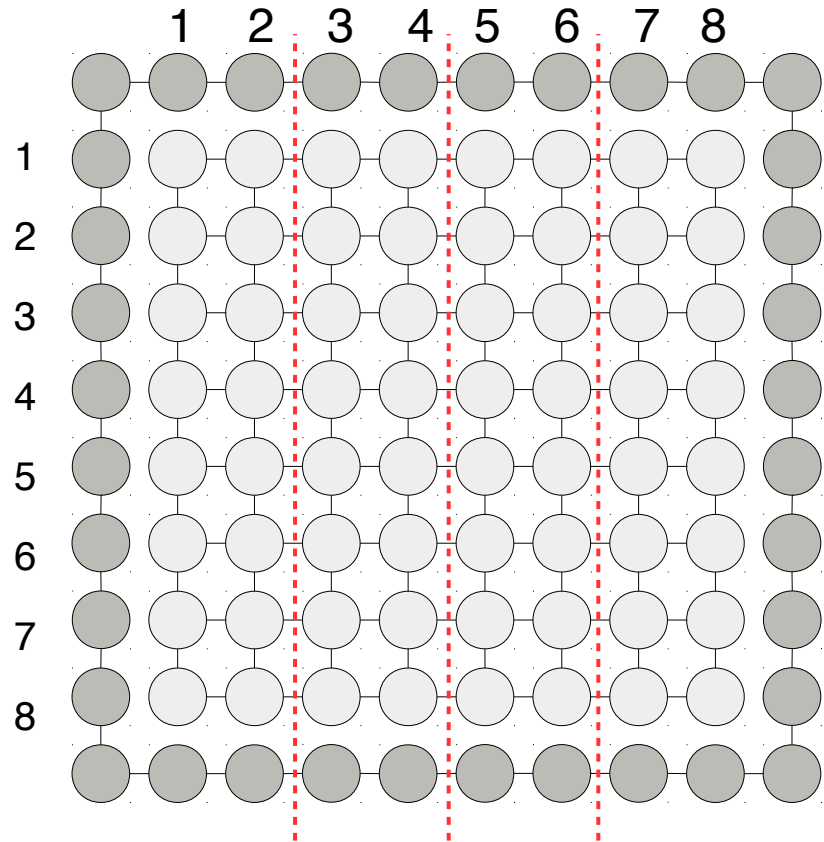


# Details: Divisibility, Numerics, Mesh Refinement

- Divisibility
  - More complex math (no simple way to pad to  $N+k$ )
    - We have to tolerate slight imbalance
  - Still want square processor grid
    - Might need to leave processors off for good prime factors
- Numerical issues
  - Convergence is a more complicated math problem
    - Need continuous boundary conditions etc.
  - More complicated PDEs and local solvers are a necessity
- Mesh structure
  - It does not always make sense to have uniform mesh
  - The mesh might change as computation proceeds



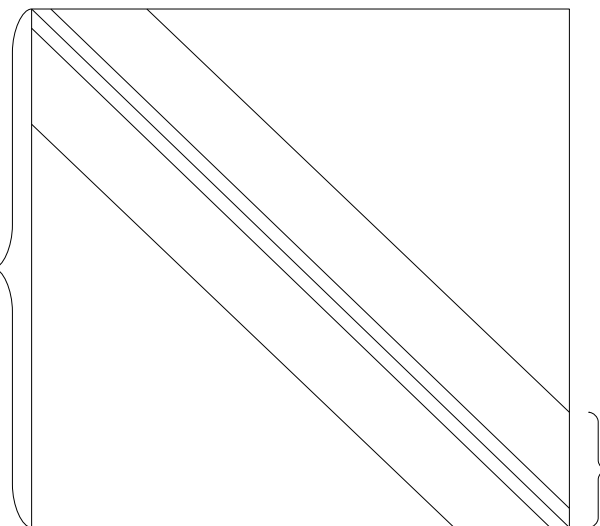
# Mesh and Its Adjacency Matrix



	1,1	2,1	3,1	4,1	5,1	6,1, 7,1	8,1	1,2	2,2	3,2
1,1	-4	1	0	0	0	0	0	1	0	0
2,1	1	-4	1	0	0	0	0	0	1	0
3,1	0	1	-4	1	0	0	0	0	0	1
4,1	0	0	1	-4	1	0	0	0	0	0

Adjacency matrix is sparse:

$N^2$



N

Natural ordering  
(other orderings possible:  
red-black, nested dissection,  
Cuthill-McKee, ...)