CS240A: Parallelism in CSE Applications

Tao Yang

Slides revised from James Demmel and Kathy Yelick

www.cs.berkeley.edu/~demmel/cs267_Spr11

Category of CSE Simulation Applications

- Discrete event systems
  - Time and space are discrete
- Particle systems
  - Important special case of lumped systems
- Ordinary Differentiation Equations (ODEs)
  - Location/entities are discrete, time is continuous
- Partial Differentiation Equations (PDEs)
  - Time and space are continuous

discrete

continuous

Basic Kinds of CSE Simulation

- Discrete event systems:
  - "Game of Life," Manufacturing systems, Finance, Circuits, Pacman
- Particle systems:
  - Billiard balls, Galaxies, Atoms, Circuits, Pinball …
- Ordinary Differential Equations (ODEs),
  - Lumped variables depending on continuous parameters
    - system is "lumped" because we are not computing the voltage/current at every point in space along a wire, just endpoints
  - Structural mechanics, Chemical kinetics, Circuits, Star Wars: The Force Unleashed
- Partial Differential Equations (PDEs)
  - Continuous variables depending on continuous parameters
  - Heat, Elasticity, Electrostatics, Finance, Circuits, Medical Image Analysis, Terminator 3: Rise of the Machines
- For more on simulation in games, see
  - www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD

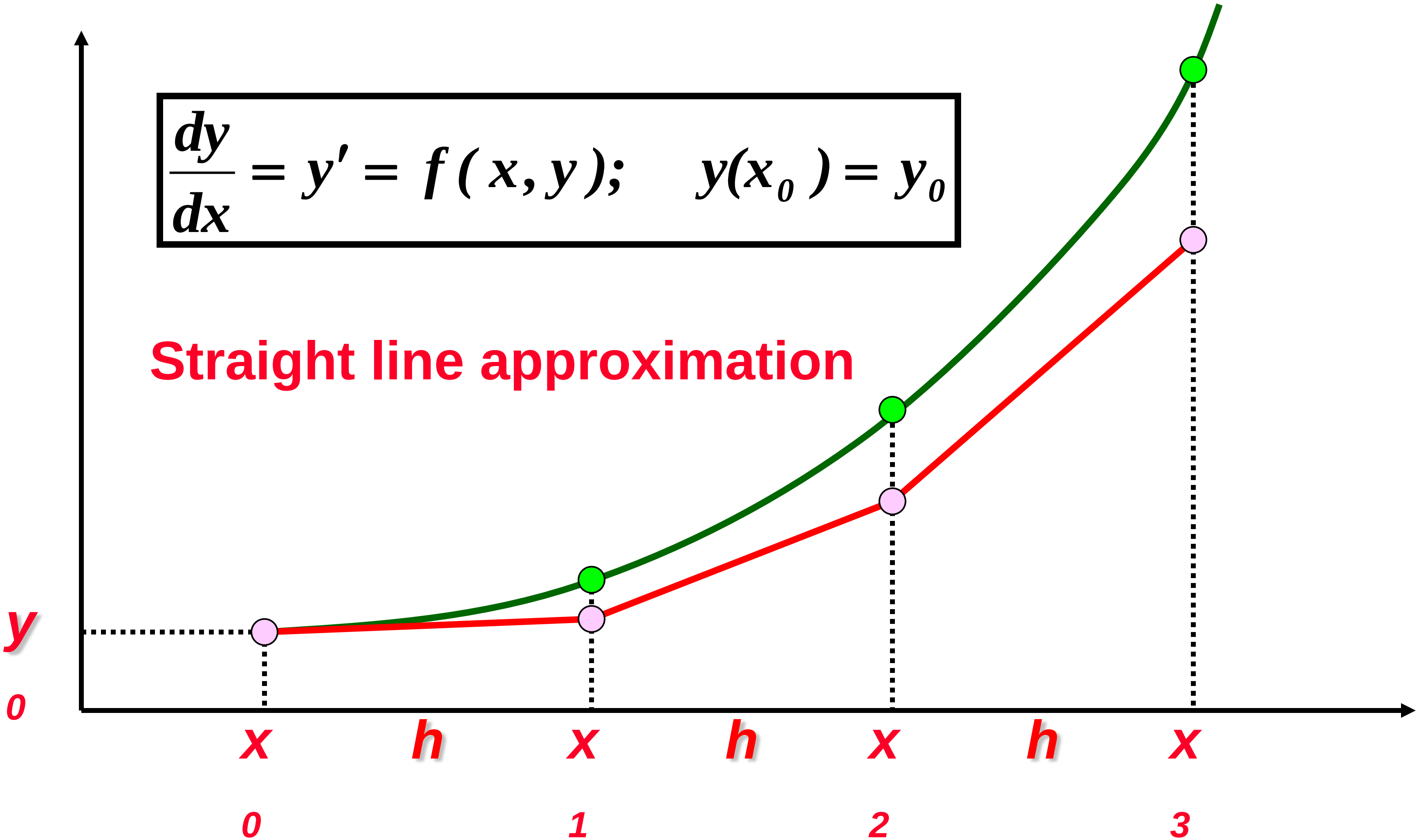
Table of Cotent

Finite-Difference Method for ODE/PDE

- Discretize domain of a function
- For each point in the discretized domain, name it with a variable, setup equations.
- The unknown values of those points form equations. Then solve these equations

Euler's method for ODE Initial-Value Problems

$$\frac{dy}{dx} = y' = f(x, y); \quad y(x_0) = y_0$$



Euler Method

Approximate: $$y'(x_0) \approx (y(x+\Delta h) - y(x_0)) / \Delta h$$

Then:

$$y_{\mathrm{n+1}} = y_{\mathrm{n}} + \Delta h\, y_{\mathrm{n}}' + O\left(\Delta h^2\right)$$

$$y_{\mathrm{n+1}} = y_{\mathrm{n}} + \Delta h\, \mathrm{f}(x_{\mathrm{n}}, y_{\mathrm{n}}) + O\left(\Delta h^2\right)$$

Thus starting from an initial value y_0

$$y_{\mathrm{n+1}} \approx y_{\mathrm{n}} + \Delta h\, \mathrm{f}(x_{\mathrm{n}}, y_{\mathrm{n}}) \quad \textit{with} \;\; O\left(\Delta h^2\right) \;\; \text{error}$$

Example

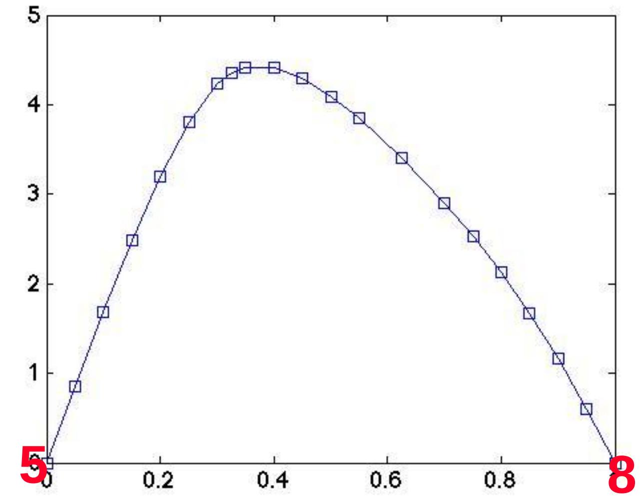
$$\frac{dy}{dx} = x + y \qquad y(0) = 1$$

$$y_{n+1} \approx y_n + \Delta h\, f(x_n, y_n) = y_n + \Delta h\,(x_n + y_n)$$

$$\Delta h = 0.02$$

x_n	y_n	y'_n	hy'_n	Exact Solution	Error
0	1.00000	1.00000	0.02000	1.00000	0.00000
0.02	1.02000	1.04000	0.02080	1.02040	-0.00040
0.04	1.04080	1.08080	0.02162	1.04162	-0.00082
0.06	1.06242	1.12242	0.02245	1.06367	-0.00126
0.08	1.08486	1.16486	0.02330	1.08657	-0.00171
0.1	1.10816	1.20816	0.02416	1.11034	-0.00218
0.12	1.13232	1.25232	0.02505	1.13499	-0.00267
0.14	1.15737	1.29737	0.02595	1.16055	-0.00318
0.16	1.18332	1.34332	0.02687	1.18702	-0.00370
0.18	1.21019	1.39019	0.02780	1.21443	-0.00425
0.2	1.23799	1.43799	0.02876	1.24281	-0.00482

ODE with boundary value



$$\frac{d^2u}{dr^2} + \frac{1}{r}\frac{d\ u}{dr} - \frac{u}{r^2} = 0$$

$$u(5) = 0.0038731'',$$

$$u(8) = 0.0030769''$$

Solution

Using the approximation of

$$\frac{d^2y}{dx^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta x)^2}$$

and

$$\frac{dy}{dx} \approx \frac{y_{i+1} - y_{i-1}}{2(\Delta x)}$$

Gives you

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta r)^2} + \frac{1}{r_i}\frac{u_{i+1} - u_{i-1}}{2(\Delta r)} - \frac{u_i}{r_i^{\,2}} = 0$$

$$\left(-\frac{1}{2r_i(\Delta r)} + \frac{1}{(\Delta r)^2}\right)u_{i-1} + \left(-\frac{2}{(\Delta r)^2} - \frac{1}{r_i^{\,2}}\right)u_i + \left(\frac{1}{(\Delta r)^2} + \frac{1}{2r_i\Delta r}\right)u_{i+1} = 0$$

Solution Cont

Step 1 **At node** $i = 0, r_0 = a = 5$

$$u_0 = 0.0038731$$

Step 2 **At node** $i = 1,\ r_1 = r_0 + \Delta r = 5 + 0.6 = 5.6''$

$$\left(-\frac{1}{2(5.6)(0.6)} + \frac{1}{(0.6)^2}\right)u_0 + \left(-\frac{2}{(0.6)^2} - \frac{1}{(5.6)^2}\right)u_1 + \left(\frac{1}{0.6^2} + \frac{1}{2(5.6)(0.6)}\right)u_2 = 0$$

$$2.6290u_0 - 5.5874u_1 + 2.9266u_2 = 0$$

Step 3 **At node** $i = 2,\quad r_2 = r_1 + \Delta r = 5.6 + 0.6 = 6.2$

$$\left(-\frac{1}{2(6.2)(0.6)} + \frac{1}{0.6^2}\right)u_1 + \left(-\frac{2}{0.6^2} - \frac{1}{6.2^2}\right)u_2 + \left(\frac{1}{0.6^2} + \frac{1}{2(6.2)(0.6)}\right)u_3 = 0$$

$$2.6434u_1 - 5.5816u_2 + 2.9122u_3 = 0$$

Solution Cont

Step 4 At node $i = 3, \quad r_3 = r_2 + \Delta r = 6.2 + 0.6 = 6.8$

$$\left(-\frac{1}{2(6.8)(0.6)} + \frac{1}{0.6^2}\right)u_2 + \left(-\frac{2}{0.6^2} - \frac{1}{6.8^2}\right)u_3 + \left(\frac{1}{0.6^2} + \frac{1}{2(6.8)(0.6)}\right)u_4 = 0$$

$$2.6552u_2 - 5.5772u_3 + 2.9003u_4 = 0$$

Step 5 At node $i = 4, \quad r_4 = r_3 + \Delta r = 6.8 + 0.6 = 7.4$

$$\left(-\frac{1}{2(7.4)(0.6)} + \frac{1}{0.6^2}\right)u_3 + \left(-\frac{2}{0.6^2} - \frac{1}{(7.4)^2}\right)u_4 + \left(\frac{1}{0.6^2} + \frac{1}{2(7.4)(0.6)}\right)u_5 = 0$$

$$2.6651u_3 - 5.6062u_4 + 2.8903u_5 = 0$$

Step 6 At node $i = 5, \quad r_5 = r_4 + \Delta r = 7.4 + 0.6 = 8$

$$u_5 = u/_{r=b} = 0.0030769$$

Solving system of equations

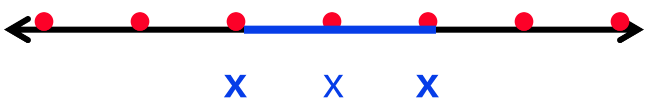
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 2.6290 & -5.5874 & 2.9266 & 0 & 0 & 0 \\ 0 & 2.6434 & -5.5816 & 2.9122 & 0 & 0 \\ 0 & 0 & 2.6552 & -5.5772 & 2.9003 & 0 \\ 0 & 0 & 0 & 2.6651 & -5.6062 & 2.8903 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{bmatrix} = \begin{bmatrix} 0.0038731 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.0030769 \end{bmatrix}$$

$$u_0 = 0.0038731 \qquad u_3 = 0.0032689$$

$$u_1 = 0.0036115 \qquad u_4 = 0.0031586$$

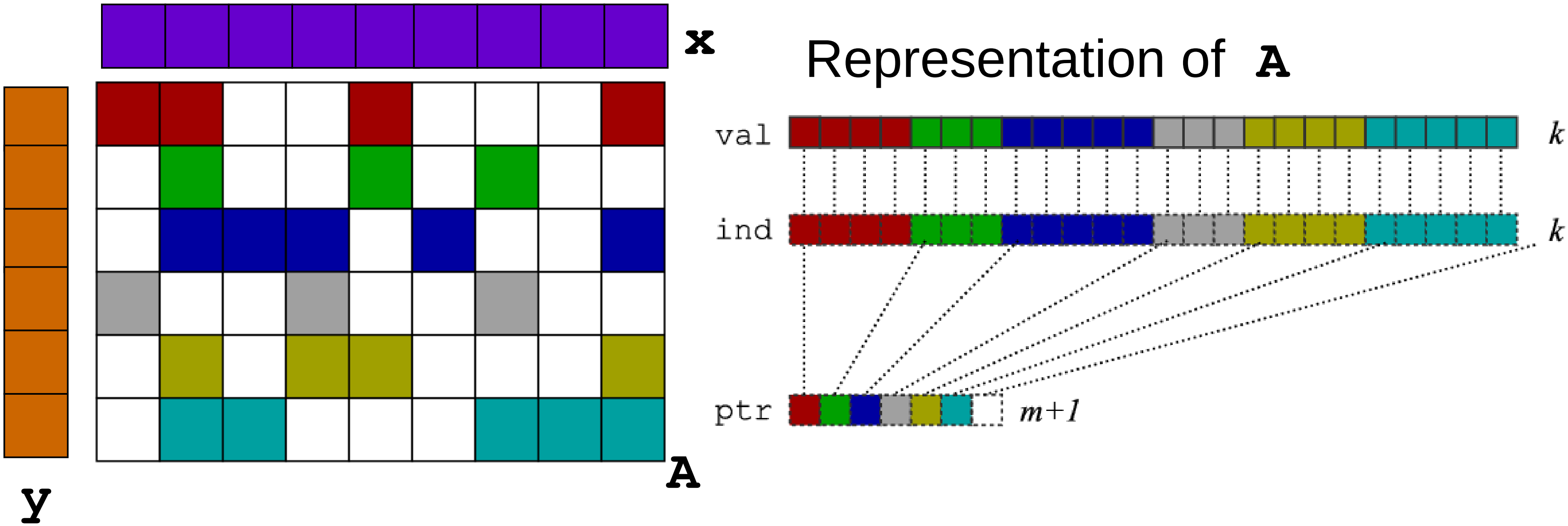
$$u_2 = 0.0034159 \qquad u_5 = 0.0030769$$

Graph and "stencil"



Compressed Sparse Row (CSR) Format

SpMV: y = y + A*x, only store, do arithmetic, on nonzero entries

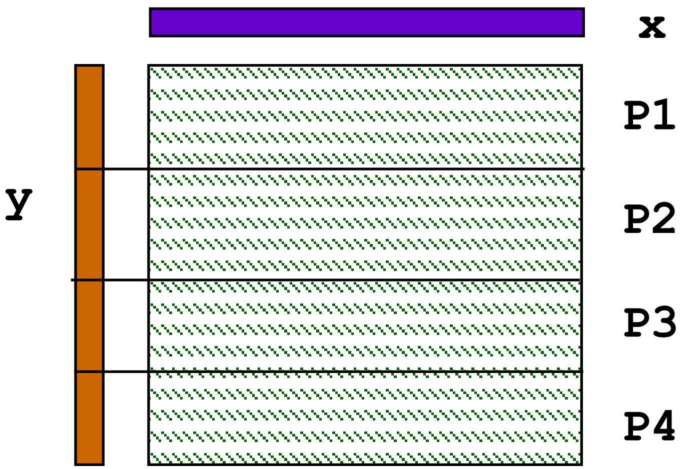


Matrix-vector multiply kernel: $y(i) \leftarrow y(i) + A(i,j) \times x(j)$

```
for each row i
  for k=ptr[i] to ptr[i+1]-1 do
      y[i] = y[i] + val[k]*x[ind[k]]
```

Parallel Sparse Matrix-vector multiplication

- y = A*x, where A is a sparse n x n matrix



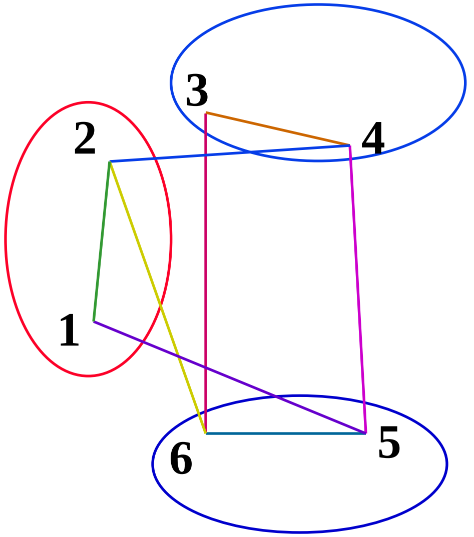
- Questions
  - which processors store
    - y[i], x[i], and A[i,j]
  - which processors compute
    - y[i] = sum (from 1 to n) A[i,j] * x[j]
      = (row i of A) * x … a sparse dot product
- Partitioning
  - Partition index set {1,…,n} = N1 ∪ N2 ∪ … ∪ Np.
  - For all i in Nk, Processor k stores y[i], x[i], and row i of A
  - For all i in Nk, Processor k computes y[i] = (row i of A) * x
    - "owner computes" rule: Processor k compute the y[i]s it owns.

May require communication

Matrix-processor mapping vs graph partitioning

- Relationship between matrix and graph

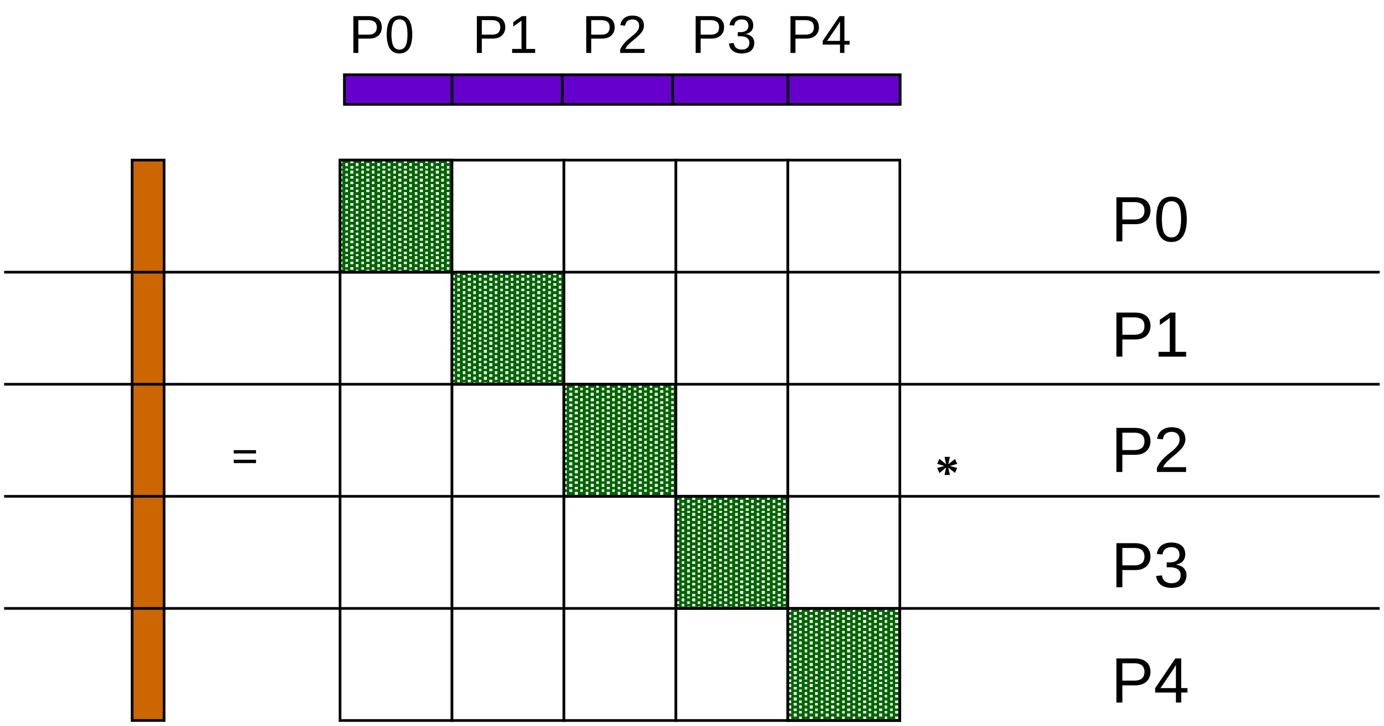
	1	2	3	4	5	6
1	1	1			1	
2	1	1		1		1
3			1	1		1
4		1	1	1	1	
5	1			1	1	1
6		1	1		1	1



- A “good” partition of the graph has
  - equal (weighted) number of nodes in each part (load and storage balance).
  - minimum number of edges crossing between (minimize communication).
- Reorder the rows/columns by putting all nodes in one partition together.

Matrix Reordering via Graph Partitioning

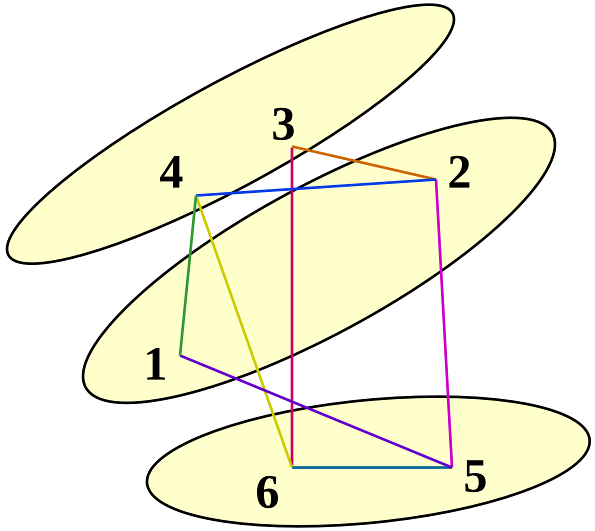
- "Ideal" matrix structure for parallelism: block diagonal
  - p (number of processors) blocks, can all be computed locally.
  - If no non-zeros outside these blocks, no communication needed
- Can we reorder the rows/columns to get close to this?
  - Most nonzeros in diagonal blocks, few outside



Graph Partitioning and Sparse Matrices

- Relationship between matrix and graph

	1	2	3	4	5	6
1	1			1	1	
2		1	1	1	1	
3		1	1			1
4	1	1		1		1
5	1	1			1	1
6			1	1	1	1



- Edges in the graph are nonzero in the matrix:
- If divided over 3 procs, there are 14 nonzeros outside the diagonal blocks, which represent the 7 (bidirectional) edges

Goals of Reordering

- Performance goals
  - balance load (how is load measured?).
    - Approx equal number of nonzeros (not necessarily rows)
  - balance storage (how much does each processor store?).
    - Approx equal number of nonzeros
  - minimize communication (how much is communicated?).
    - Minimize nonzeros outside diagonal blocks
    - Related optimization criterion is to move nonzeros near diagonal
  - improve register and cache re-use
    - Group nonzeros in small vertical blocks so source (x) elements loaded into cache or registers may be reused (temporal locality)
    - Group nonzeros in small horizontal blocks so nearby source (x) elements in the cache may be used (spatial locality)
- Other algorithms reorder for other reasons
  - Reduce # nonzeros in matrix after Gaussian elimination
  - Improve numerical stability

Table of Cotent

- ODE
- PDE
- Discrete Events and Particle Systems

Solving PDEs

- Finite element method
- Finite difference method (our focus)
  - Converts PDE into matrix equation
    - Linear system over discrete basis elements
  - Result is usually a sparse matrix

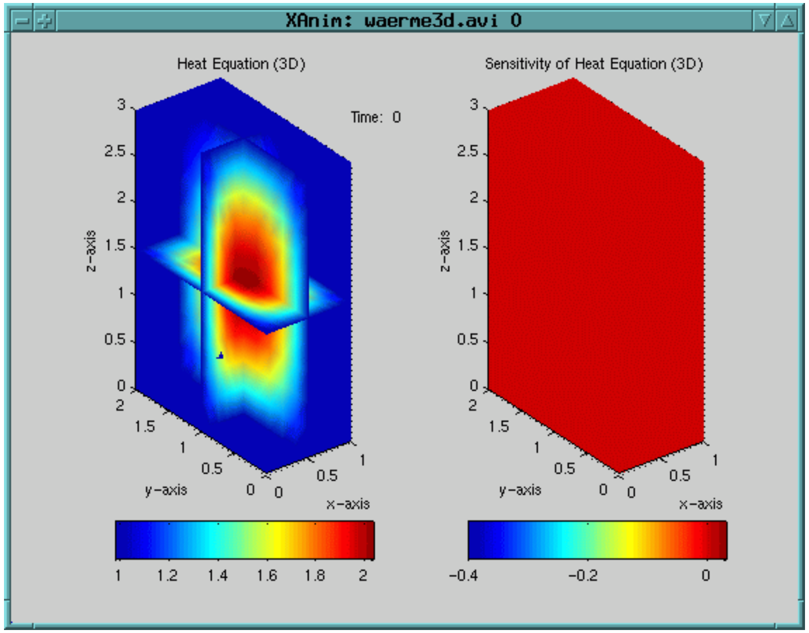
Class of Linear Second-order PDEs

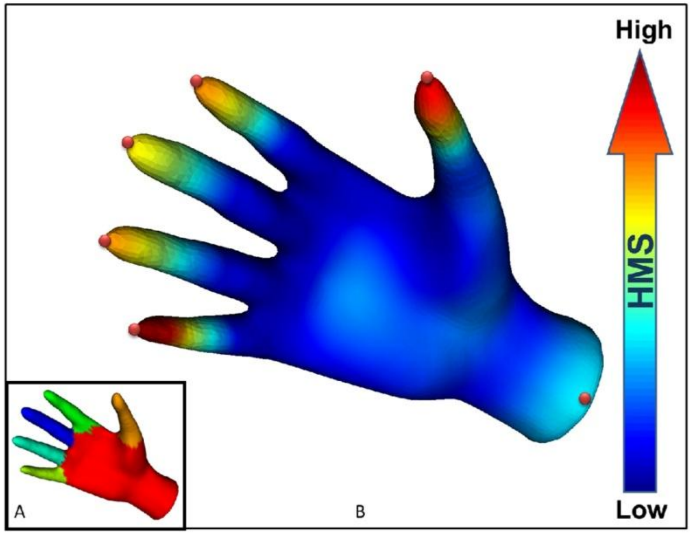
- Linear second-order PDEs are of the form

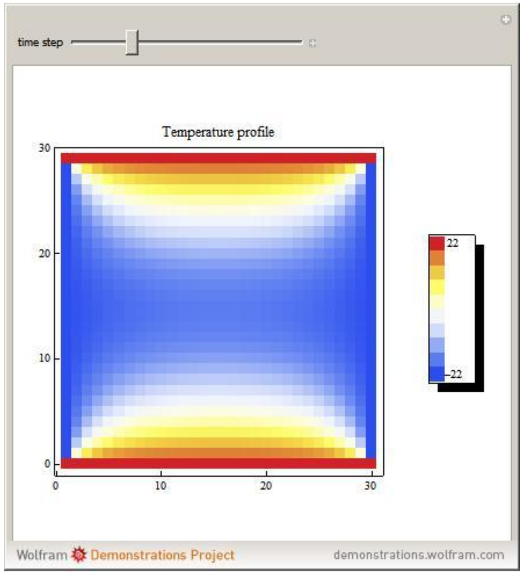
$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Eu_x + Fu_y + Gu = H$$

  where *A* - *H* are functions of *x* and *y* only
- Elliptic PDEs: $B^2 - AC < 0$
  (steady state heat equations)
- Parabolic PDEs: $B^2 - AC = 0$
  (heat transfer equations)
- Hyperbolic PDEs: $B^2 - AC > 0$
  (wave equations)

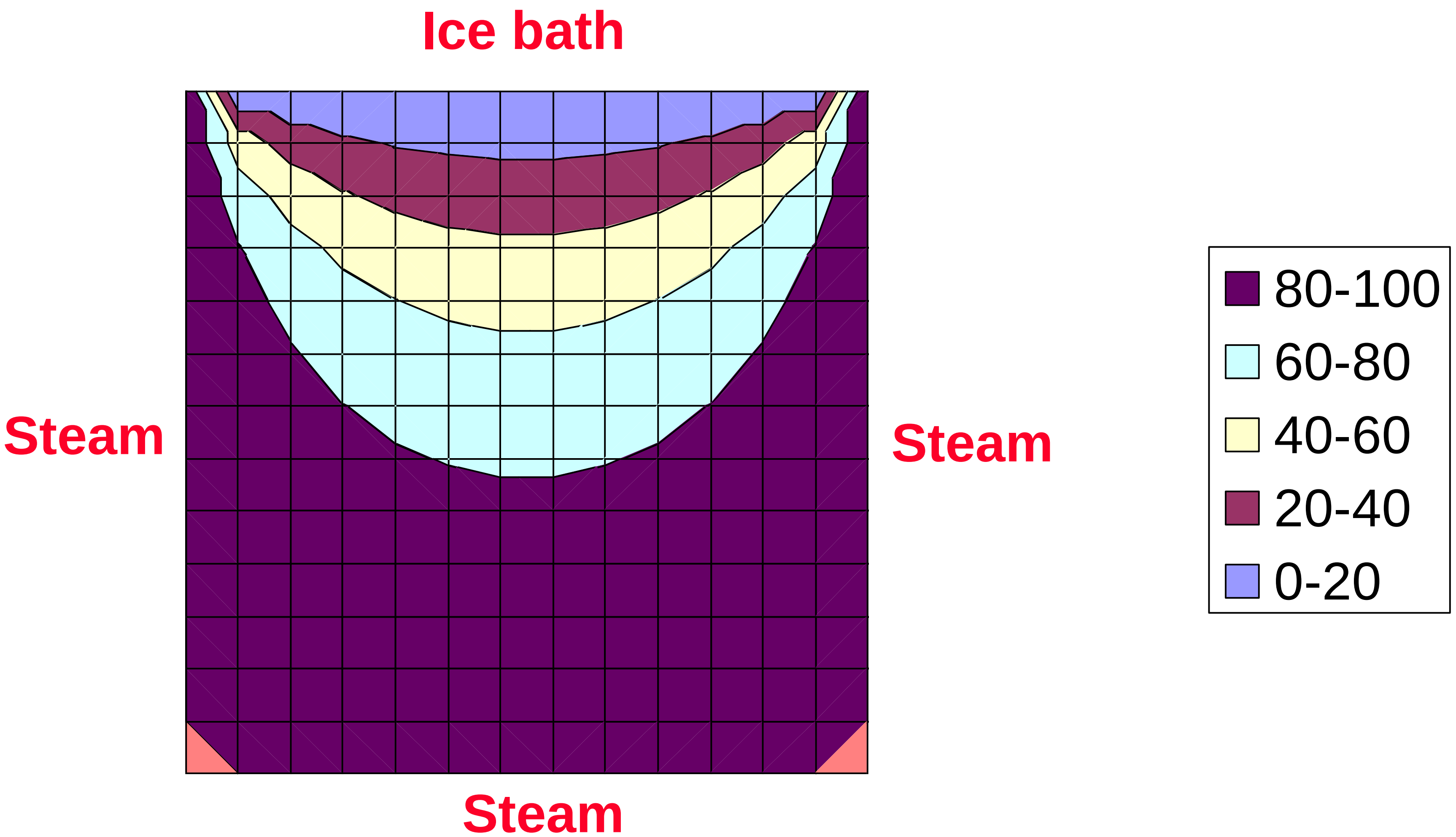
Various 2D/3D heat distribution







2D Steady State Heat Distribution



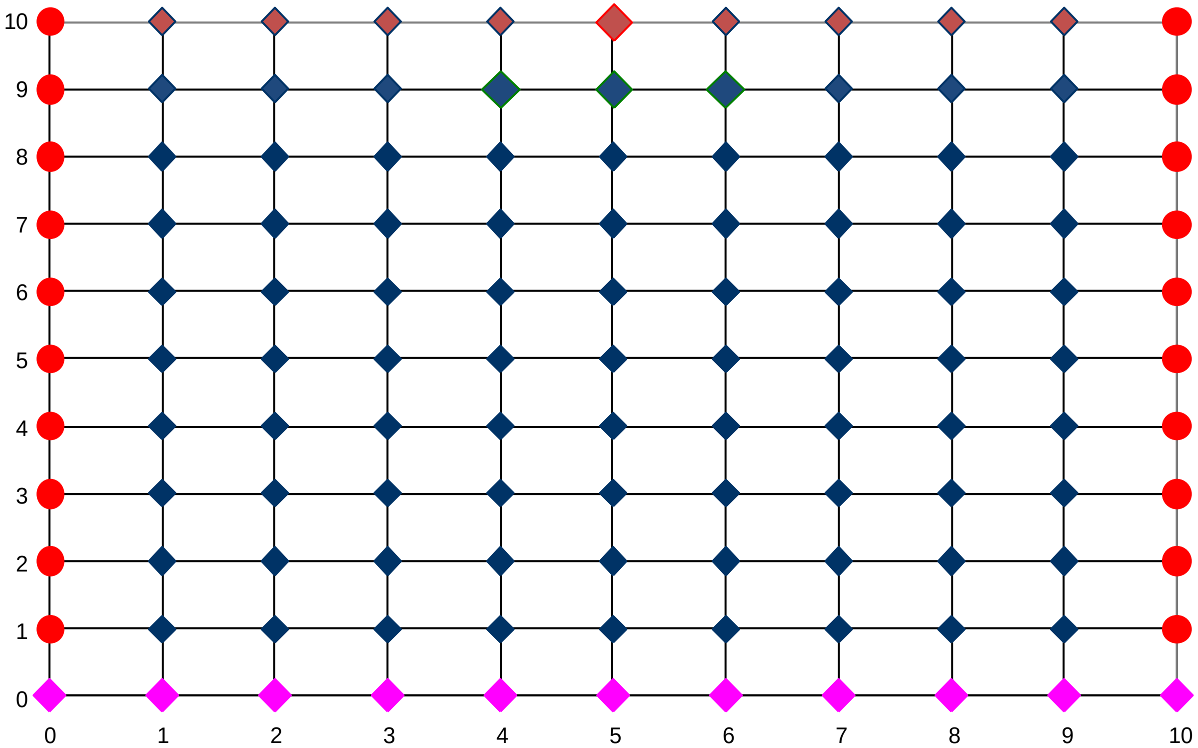
Solving the Heat Problem with PDE

- Underlying PDE is the Poisson equation

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

- This is an example of an elliptical PDE
- Will create a 2-D grid
- Each grid point represents value of state state solution at particular (*x, y*) location in plate

Discrete 2D grid space



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

Finite-difference

- **Assume f(x,y)=0**

$$\frac{1}{h^2}(u_{i-1,j} - 2u_{i,j} + u_{i+1,j})$$

- **Namely**

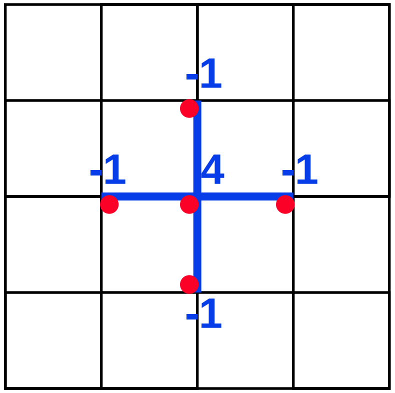
$$+\frac{1}{h^2}(u_{i-1,j+1} - 2u_{i,j+1} + u_{i+1,j+1}) = 0$$

$$4u_{i,j} - u_{i,j-1} - u_{i,j+1} - u_{i-1,j} - u_{i+1,j} = 0$$

Matrx vs. graph representation

$$L = \left(\begin{array}{ccc:ccc:ccc} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ \hdashline -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ \hdashline & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{array}\right)$$

Graph and "5 point stencil"



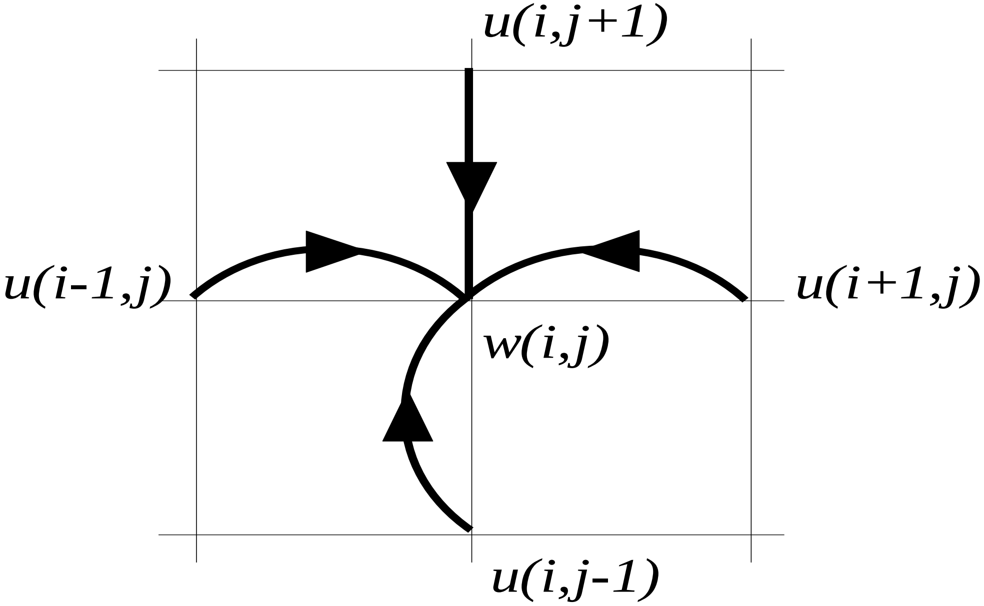
3D case is analogous
(7 point stencil)

Jacobi method for iterative solutions

Start with initial values.
Iteratively update variables based on equations

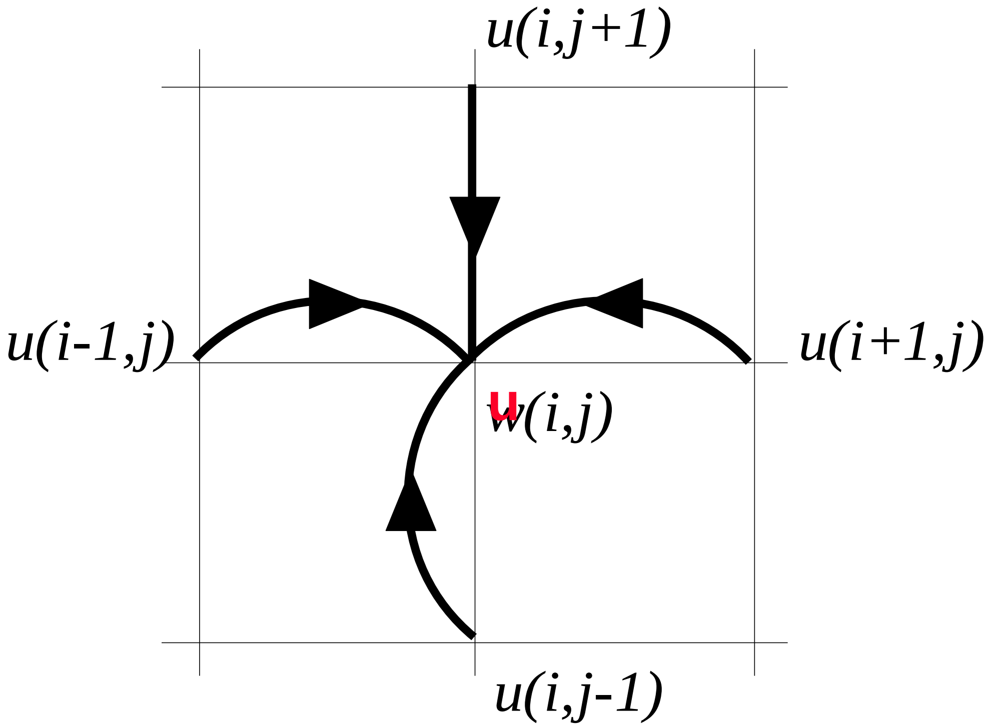
```
For i=1 to n
   for j= 1 to n
       w[i][j] = (u[i-1][j] + u[i+1][j] +
                u[i][j-1] + u[i][j+1]) / 4.0;

Swap w and u
```



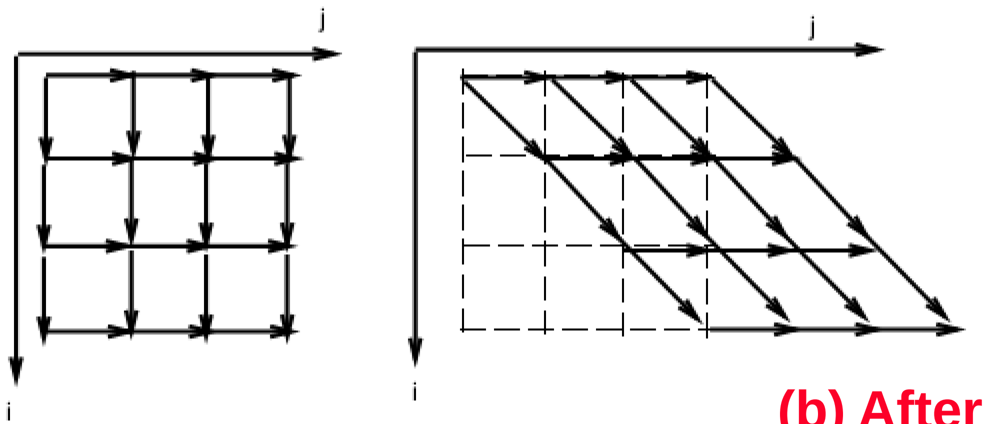
Gauss Seidel Iterative Method

```
For i = 1, n
For j = 1, n
    u[i][j] = (u[i-1][j] + u[i+1][j] +
                   u[i][j-1] + u[i][j+1]) / 4.0;
```

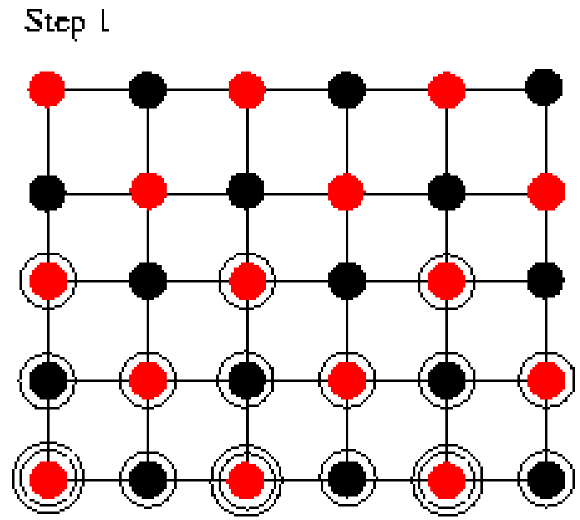


Gauss-Seidel method for equation solving

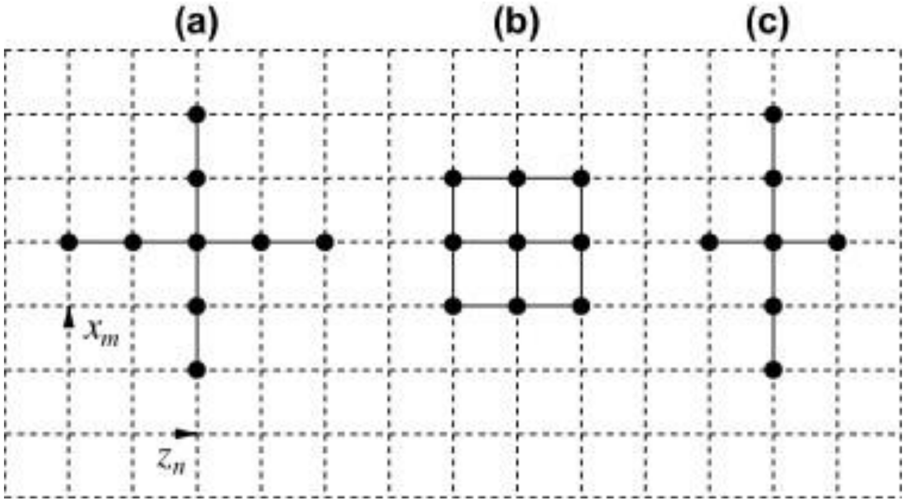
(a) 2D dependence graph

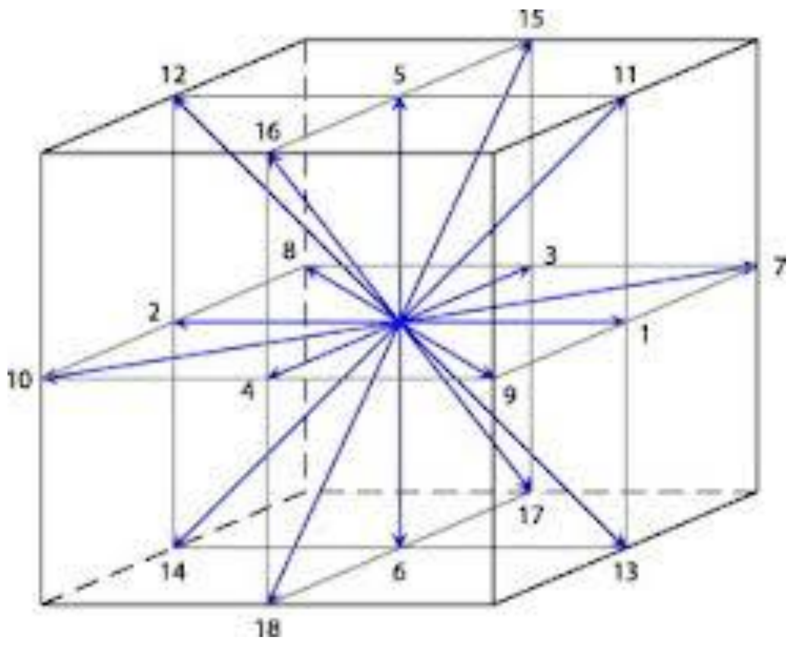


(b) After red/black variable reordering



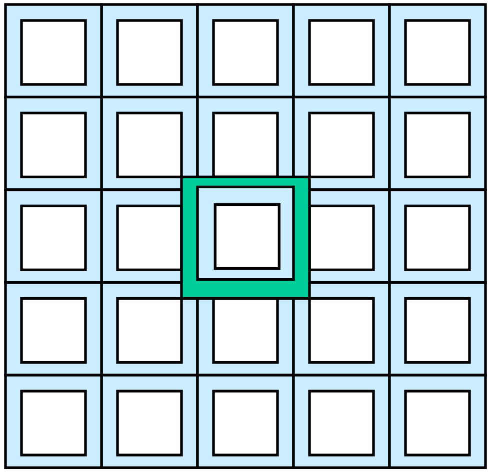
Different Dependence Patterns (Stencil)





Processor Partitioning in Regular meshes

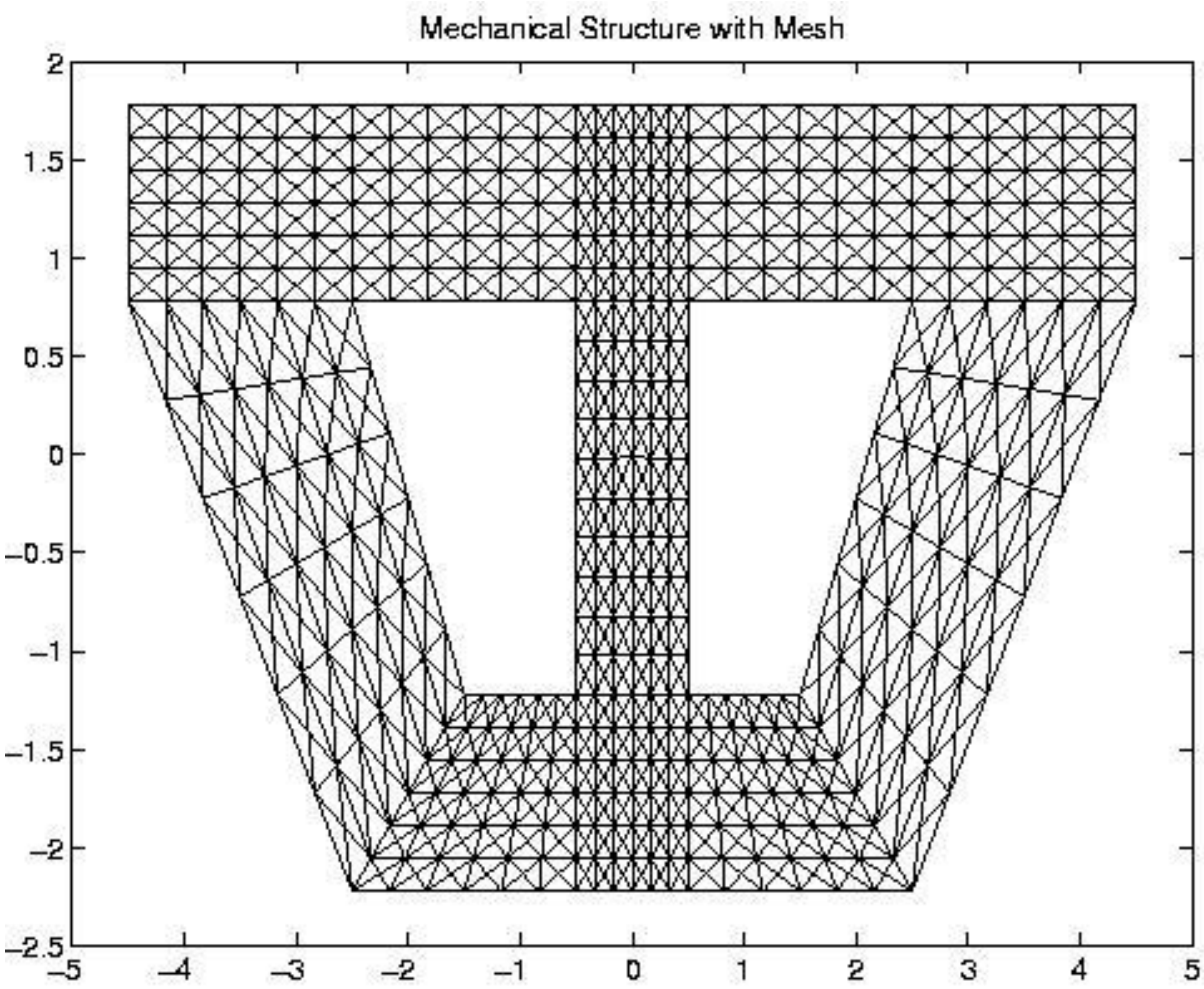
- Computing a Stencil on a regular mesh
  - need to communicate mesh points near boundary to neighboring processors.
    - Often done with ghost regions
  - Surface-to-volume ratio keeps communication down, but
    - Still may be problematic in practice



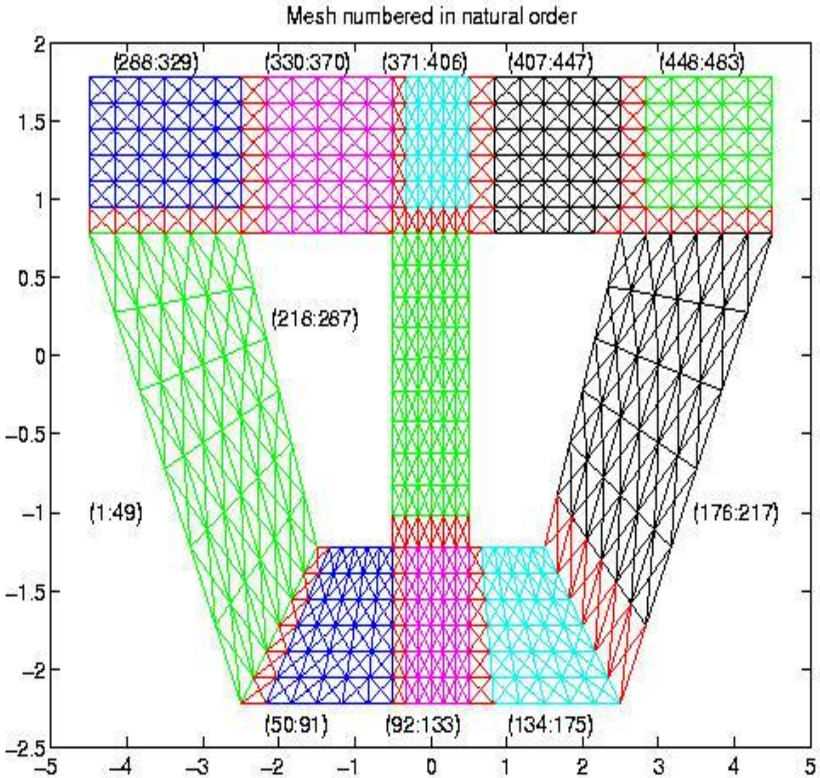
Implemented using "ghost" regions.

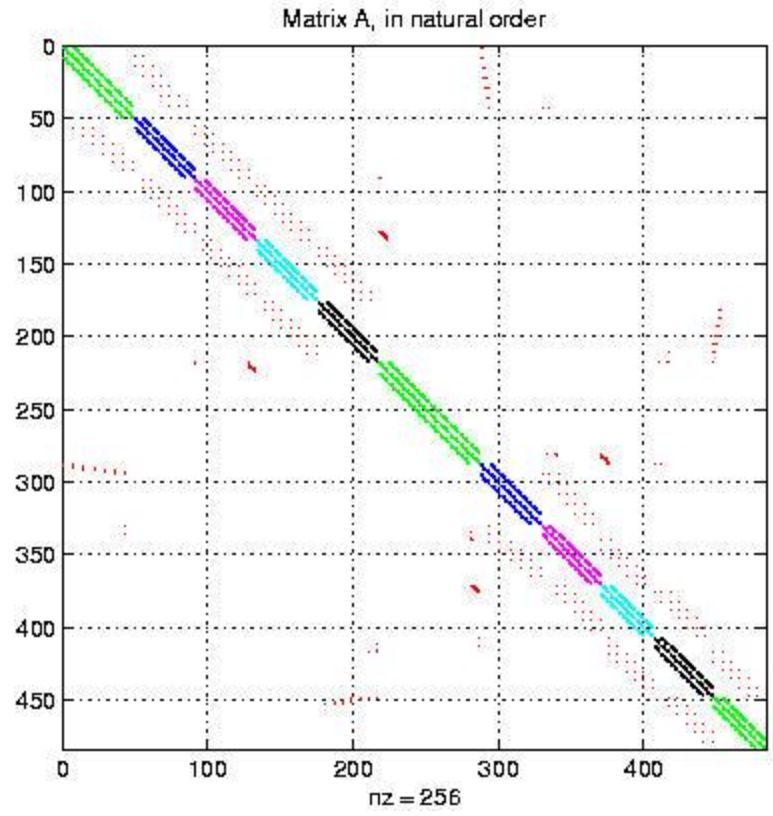
Adds memory overhead

Composite mesh from a mechanical structure



Converting the mesh to a matrix

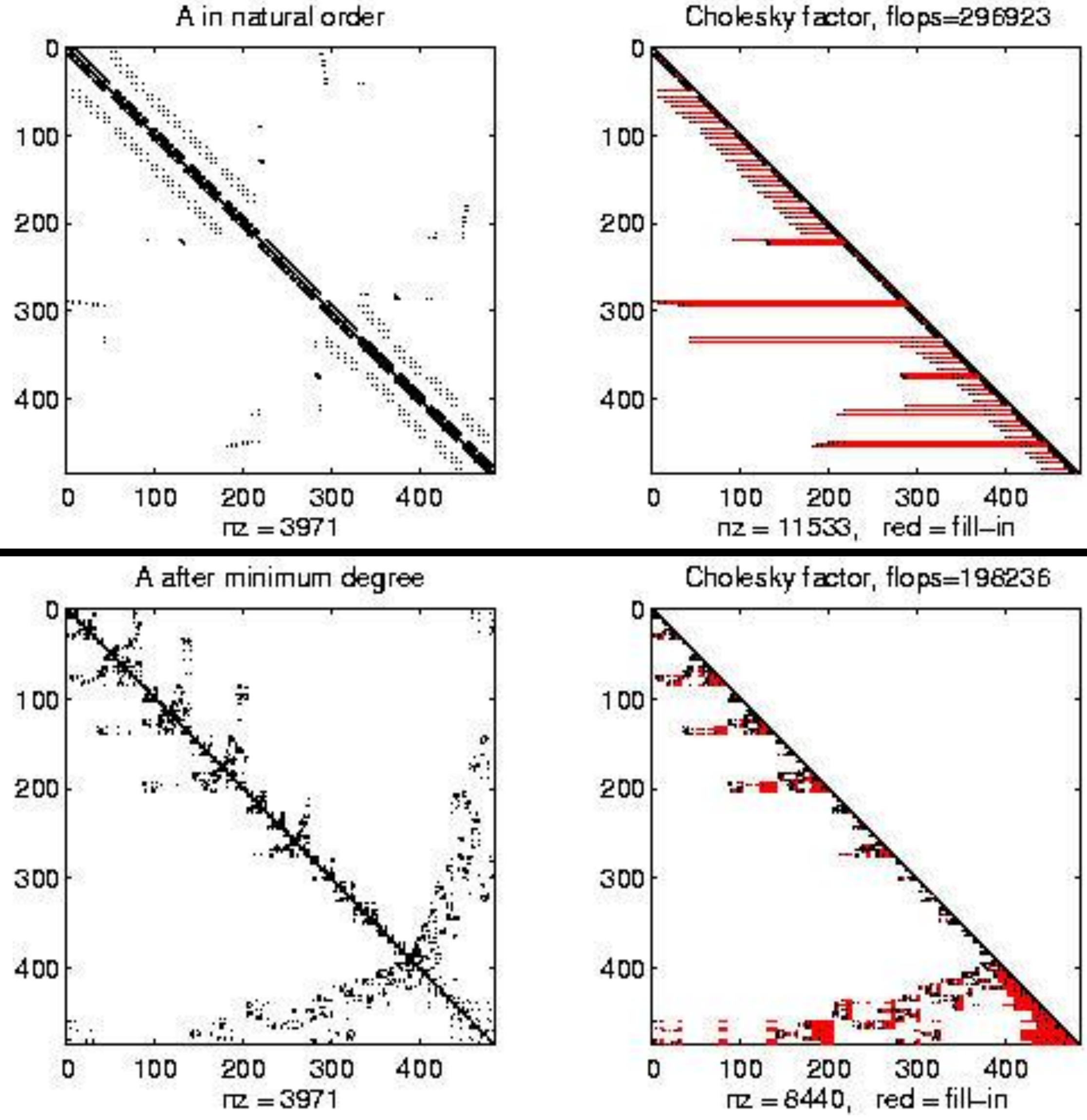




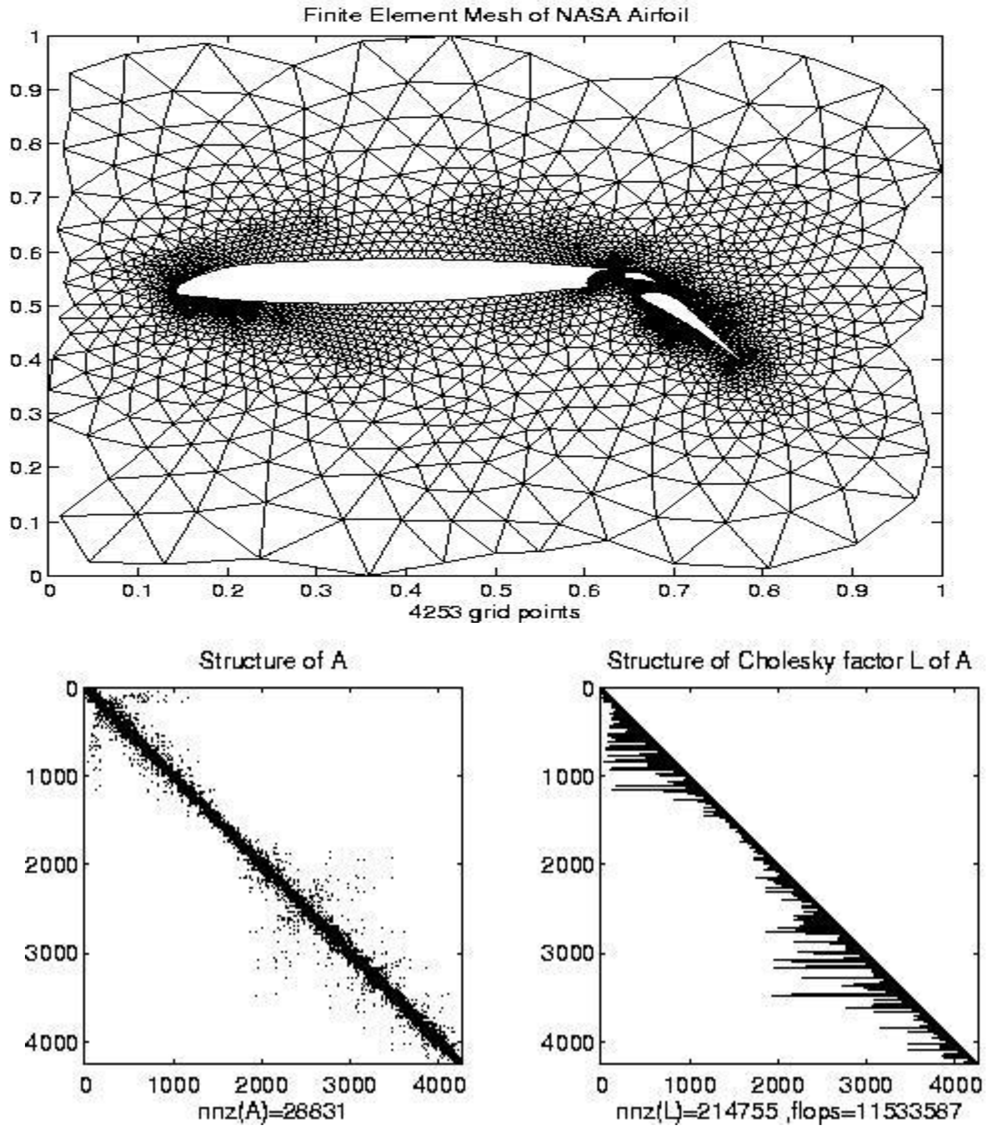
Example of Matrix Reordering Application

When performing Gaussian Elimination Zeros can be filled ☹

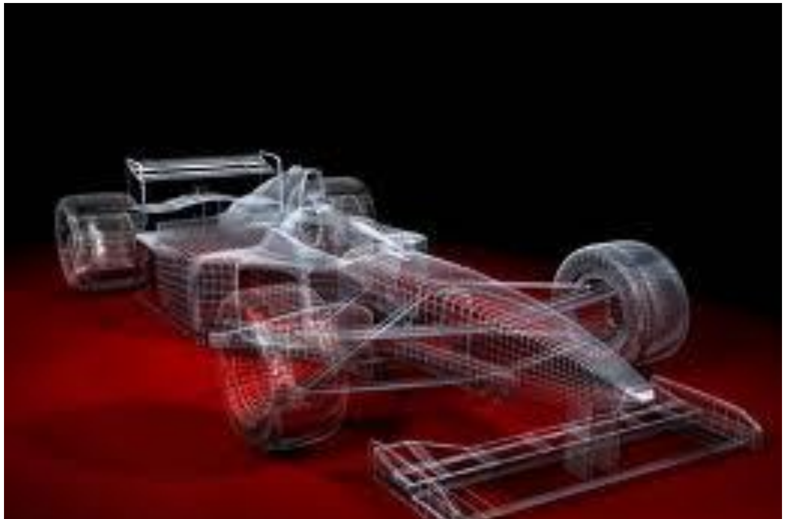
Matrix can be reordered to reduce this fill
But it's not the same ordering as for parallelism



Irregular mesh: NASA Airfoil in 2D (direct solution)



Irregular mesh and multigrid



Example of Prometheus meshes

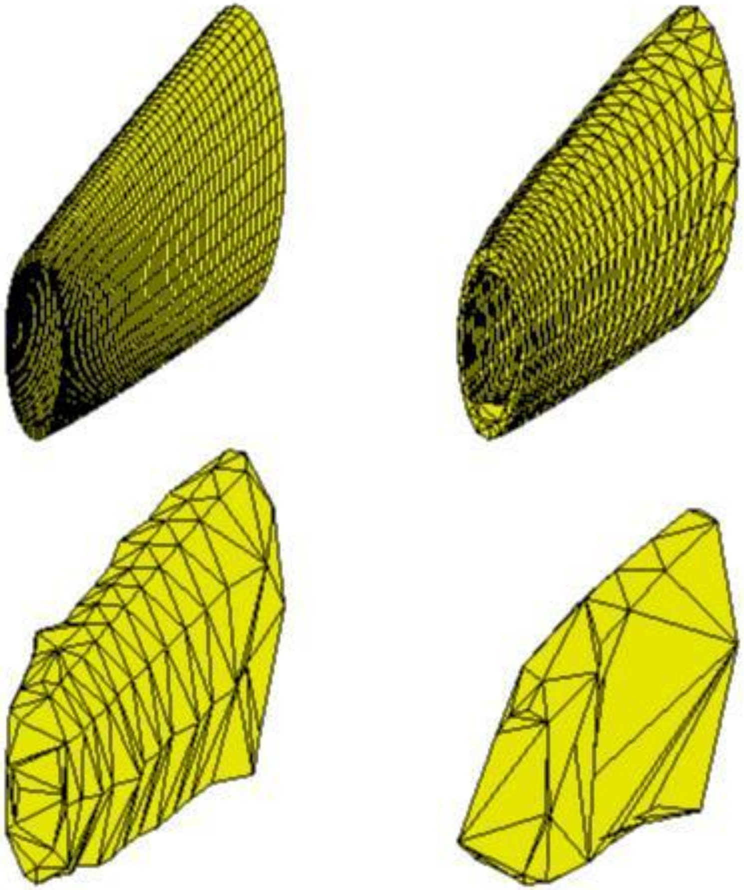


Figure 6: Sample input grid and coarse grids

Challenges of Irregular Meshes

- How to generate them in the first place
  - Start from geometric description of object
  - Triangle, a 2D mesh partitioner by Jonathan Shewchuk
  - 3D harder!
- How to partition them
  - ParMetis, a parallel graph partitioner
- How to design iterative solvers
  - PETSc, a Portable Extensible Toolkit for Scientific Computing
  - Prometheus, a multigrid solver for finite element problems on irregular meshes
- How to design direct solvers
  - SuperLU, parallel sparse Gaussian elimination

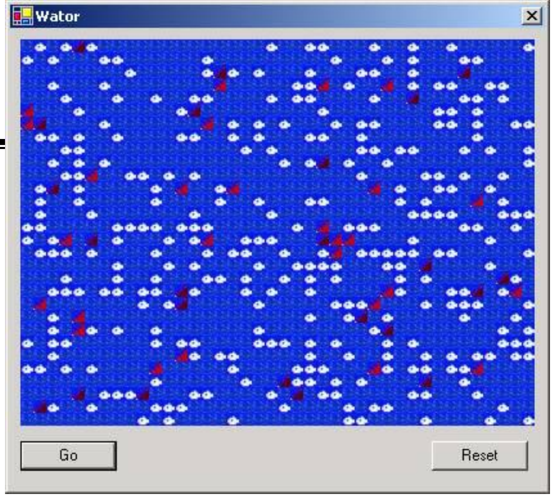
Table of Cotent

- ODE
- PDE
- Discrete Events and Particle Systems

Discrete Event Systems

- Systems are represented as:
  - finite set of variables.
  - the set of all variable values at a given time is called the state.
  - each variable is updated by computing a transition function depending on the other variables.
- System may be:
  - synchronous: at each discrete timestep evaluate all transition functions; also called a state machine.
  - asynchronous: transition functions are evaluated only if the inputs change, based on an "event" from another part of the system; also called event driven simulation.
- Example: The "game of life:"sharks and fish living in an ocean
  - breeding, eating, and death
  - forces in the ocean&between sea creatures

Parallelism in Game of Life



- The simulation is synchronous
  - use two copies of the grid (old and new).
  - the value of each new grid cell depends only on 9 cells (itself plus 8 neighbors) in old grid.
  - simulation proceeds in timesteps-- each cell is updated at every step.
- Easy to parallelize by dividing physical domain: *Domain Decomposition*

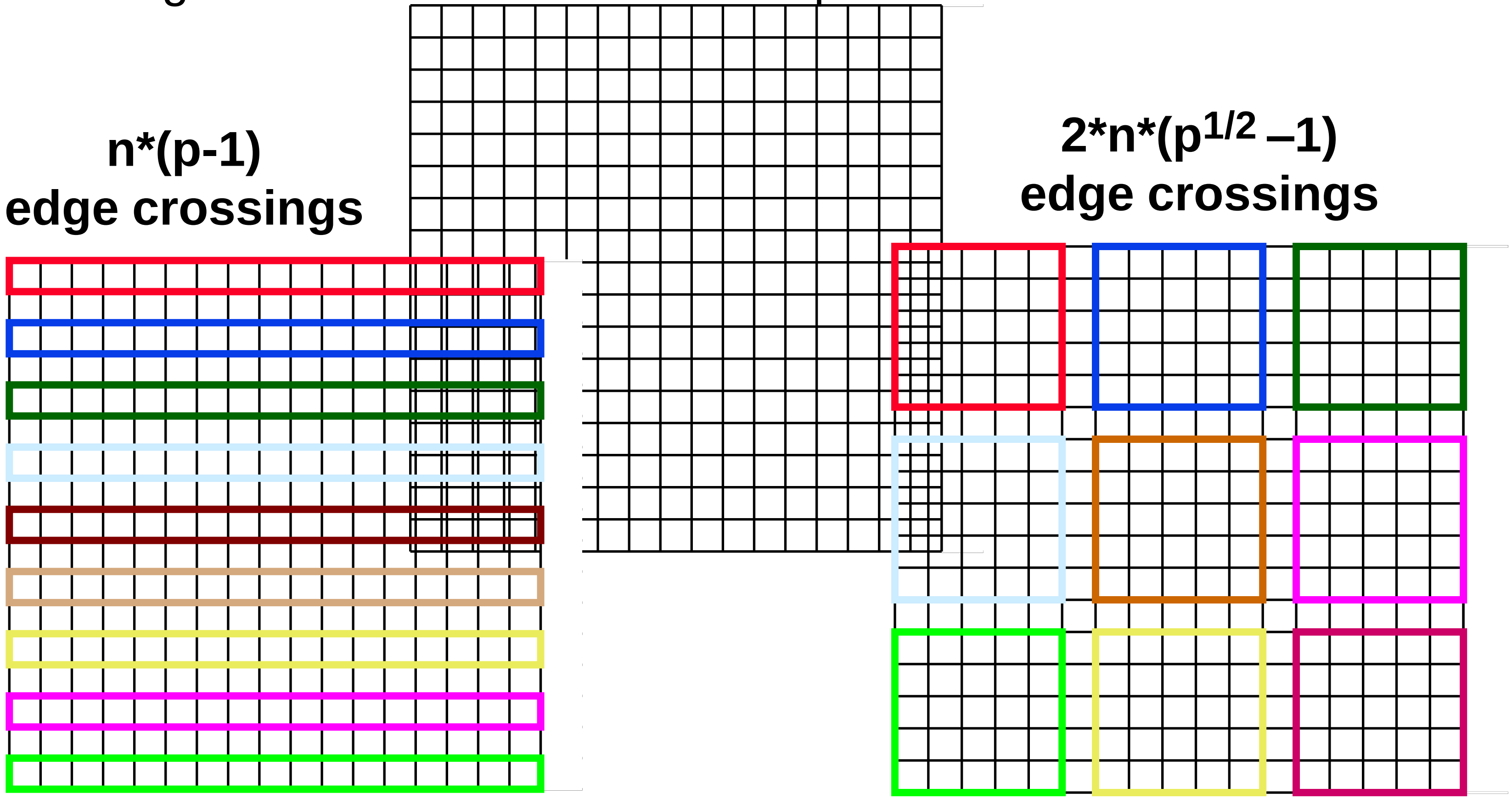
P1	P2	P3
P4	P5	P6
P7	P8	P9

Repeat
compute locally to update local system
barrier()
exchange state info with neighbors
until done simulating

- How to pick shapes of domains?

Regular Meshes (e.g. Game of Life)

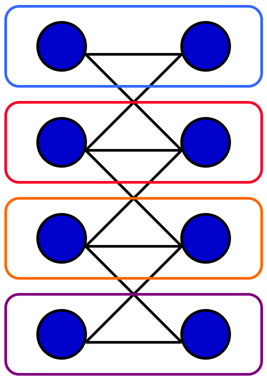
- Suppose graph is nxn mesh with connection NSEW neighbors
- Which partition has less communication? (n=18, p=9)
- Minimizing communication on mesh ≡ minimizing "surface to volume ratio" of partition



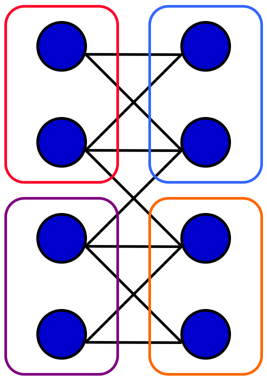
Synchronous Circuit Simulation

- Circuit is a graph made up of subcircuits connected by wires
  - Parallel algorithm is timing-driven or synchronous:
    - Evaluate all components at every timestep (determined by known circuit delay)
- Graph partitioning assigns subgraphs to processors
  - Goal 1 is to evenly distribute subgraphs to nodes (load balance).
  - Goal 2 is to minimize edge crossings (minimize communication).

better →



edge crossings = 6



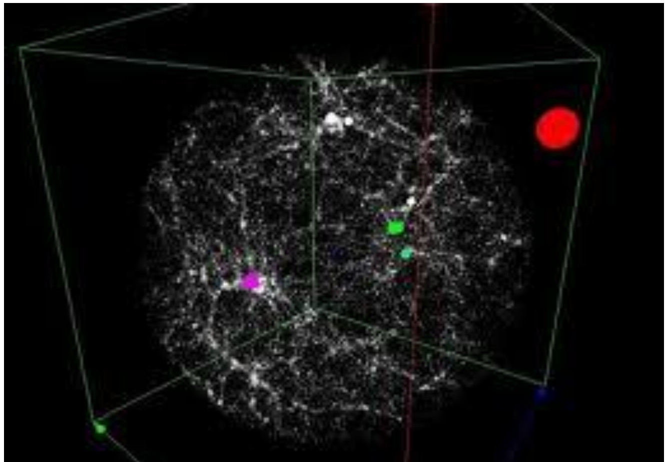
edge crossings = 10

Asynchronous Simulation

- Synchronous simulations may waste time:
  - Simulates even when the inputs do not change,.
- Asynchronous (event-driven) simulations update only when an event arrives from another component:
  - No global time steps, but individual events contain time stamp.
  - Example: Game of life in loosely connected ponds (don't simulate empty ponds).
  - Example: Circuit simulation with delays (events are gates changing).
  - Example: Traffic simulation (events are cars changing lanes, etc.).
- Asynchronous is more efficient, but harder to parallelize
  - In MPI, events are naturally implemented as messages, but how do you know when to execute a "receive"?

Particle Systems

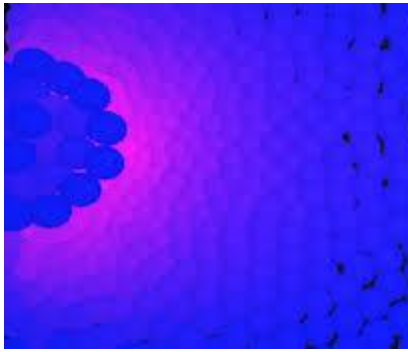
- A particle system has
  - a finite number of particles
  - moving in space according to Newton's Laws (i.e. $F = ma$)
  - time is continuous
- Examples
  - stars in space with laws of gravity
  - electron beam in semiconductor manufacturing
  - atoms in a molecule with electrostatic forces
  - neutrons in a fission reactor
  - cars on a freeway with Newton's laws plus model of driver and engine
  - balls in a pinball game



Forces in Particle Systems

- Force on each particle can be subdivided

force = external_force + nearby_force + far_field_force



- External force
  - ocean current to sharks and fish world
  - externally imposed electric field in electron beam
- Nearby force
  - sharks attracted to eat nearby fish
  - balls on a billiard table bounce off of each other
- Far-field force
  - fish attract other fish by gravity-like (1/r^2) force
  - gravity, electrostatics, radiosity in graphics

Example: Fish in an External Current

```
%    fishp = array of initial fish positions (stored as complex numbers)
%    fishv = array of initial fish velocities (stored as complex numbers)
%    fishm = array of masses of fish
%    tfinal = final time for simulation (0 = initial time)


dt = .01;   t = 0;
%  loop over time steps
     while t < tfinal,
        t = t + dt;
        fishp = fishp + dt*fishv;
        accel = current(fishp)./fishm;       % current depends on position
        fishv = fishv + dt*accel;
%       update time step (small enough to be accurate, but not too small)
        dt = min(.1*max(abs(fishv))/max(abs(accel)),1);
    end
```

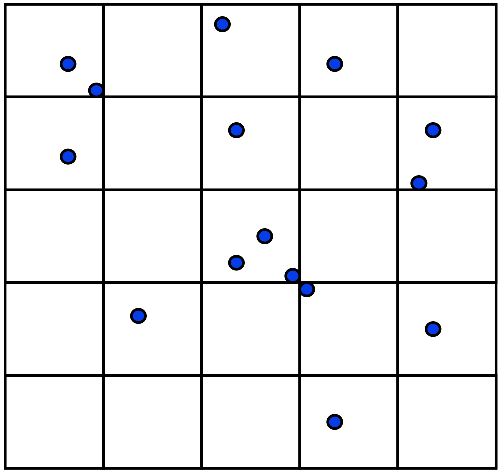
Parallelism in External Forces

- These are the simplest
- The force on each particle is independent
- Called "embarrassingly parallel"
  - Sometimes called "map reduce" by analogy

- Evenly distribute particles on processors
  - Any distribution works
  - Locality is not an issue, no communication
- For each particle on processor, apply the external force
  - May need to "reduce" (eg compute maximum) to compute time step, other data

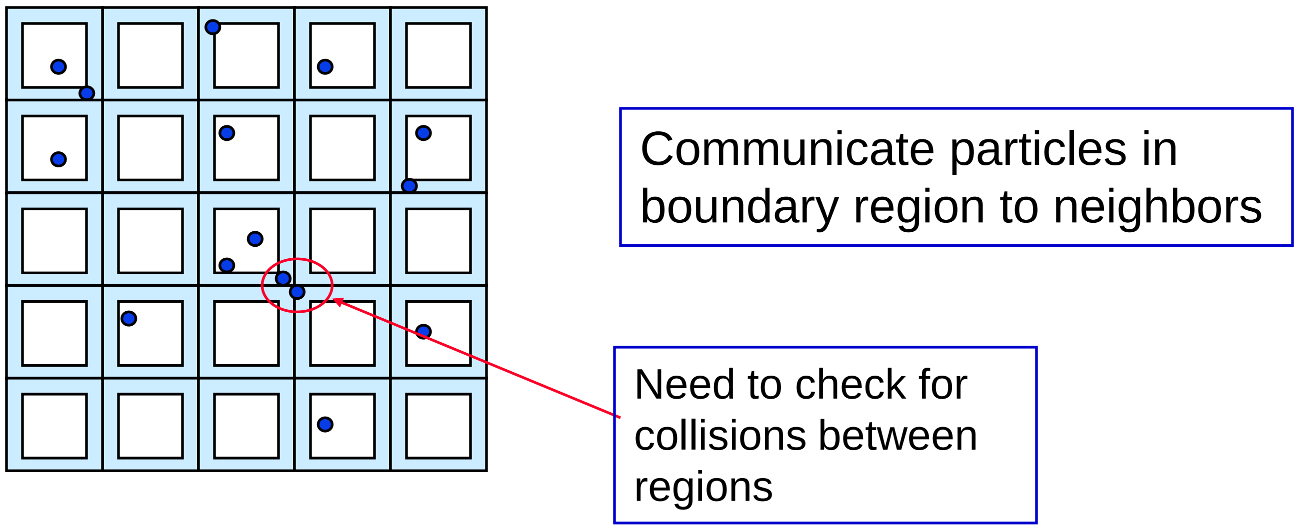
Parallelism in Nearby Forces

- Nearby forces require interaction and therefore communication.
- Force may depend on other nearby particles:
  - Example: collisions.
  - simplest algorithm is $O(n^2)$: look at all pairs to see if they collide.
- Usual parallel model is domain decomposition of physical region in which particles are located
  - O(n/p) particles per processor if evenly distributed.



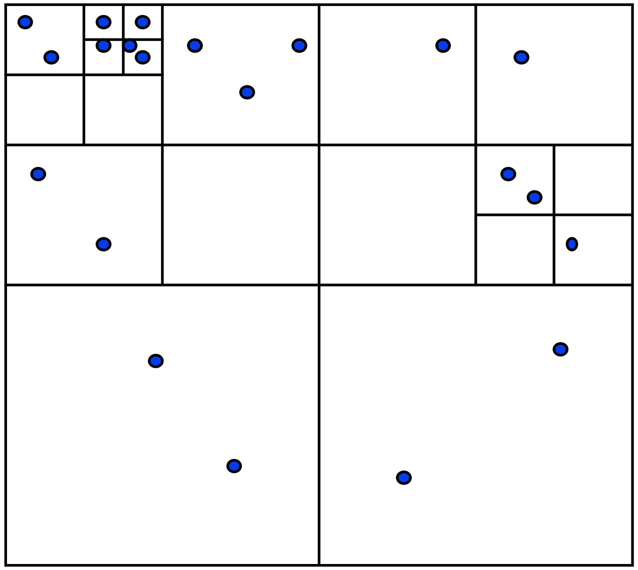
Parallelism in Nearby Forces

- Challenge 1: interactions of particles near processor boundary:
  - need to communicate particles near boundary to neighboring processors.
    - Region near boundary called "ghost zone"
  - Low surface to volume ratio means low communication.
    - Use squares, not slabs, to minimize ghost zone sizes



Parallelism in Nearby Forces

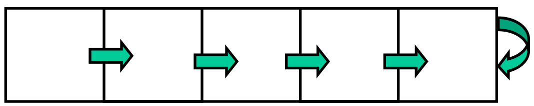
- Challenge 2: load imbalance, if particles cluster:
  - galaxies, electrons hitting a device wall.
- To reduce load imbalance, divide space unevenly.
  - Each region contains roughly equal number of particles.
  - Quad-tree in 2D, oct-tree in 3D.



Example: each square contains at most 3 particles

Parallelism in Far-Field Forces

- Far-field forces involve all-to-all interaction and therefore communication.
- Force depends on all other particles:
  - Examples: gravity, protein folding
  - Simplest algorithm is $O(n^2)$
  - Just decomposing space does not help since every particle needs to "visit" every other particle.



Implement by rotating particle sets.

- Keeps processors busy
- All processor eventually see all particles

- Use more clever algorithms to beat $O(n^2)$.

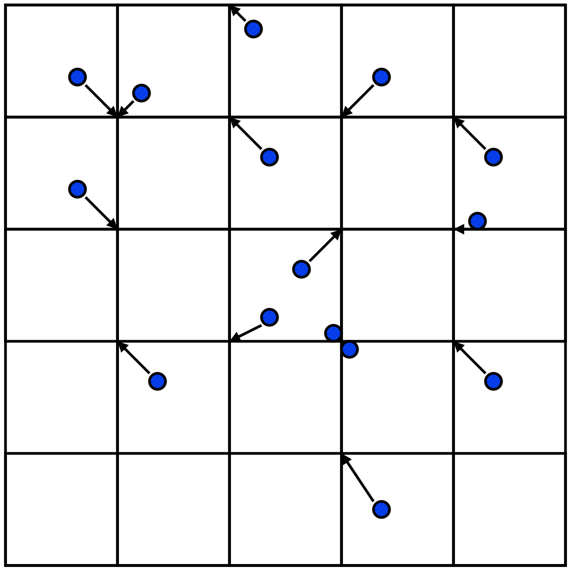
Far-field Forces: Particle-Mesh Methods

- Based on approximation:
  - Superimpose a regular mesh.
  - "Move" particles to nearest grid point.
- Exploit fact that the far-field force satisfies a PDE that is easy to solve on a regular mesh:
  - FFT, multigrid (described in future lectures)
  - Cost drops to O(n log n) or O(n) instead of $O(n^2)$
- Accuracy depends on the fineness of the grid is and the uniformity of the particle distribution.

1) Particles are moved to nearby mesh points (scatter)

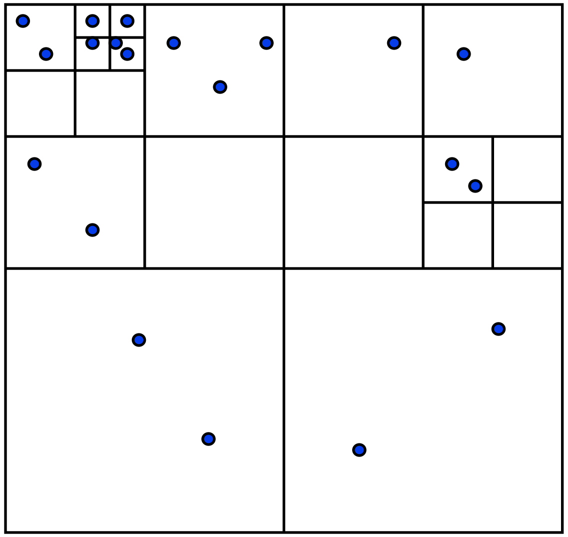
2) Solve mesh problem

3) Forces are interpolated at particles from mesh points (gather)



Far-field forces: Tree Decomposition

- Based on approximation.
  - Forces from group of far-away particles "simplified" -- resembles a single large particle.
  - Use tree; each node contains an approximation of descendants.
- Also O(n log n) or O(n) instead of $O(n^2)$.
- Several Algorithms
  - Barnes-Hut.
  - Fast multipole method (FMM) of Greengard/Rohklin.
  - Anderson's method.



- Discussed in later lecture.

Summary of Particle Methods

- Model contains discrete entities, namely, particles
- Time is continuous – must be discretized to solve

- Simulation follows particles through timesteps
  - Force = external _force + nearby_force + far_field_force
  - All-pairs algorithm is simple, but inefficient, $O(n^2)$
  - Particle-mesh methods approximates by moving particles to a regular mesh, where it is easier to compute forces
  - Tree-based algorithms approximate by treating set of particles as a group, when far away

- May think of this as a special case of a "lumped" system