

# CS 267

## Sources of Parallelism and Locality in Simulation

### Lecture 5

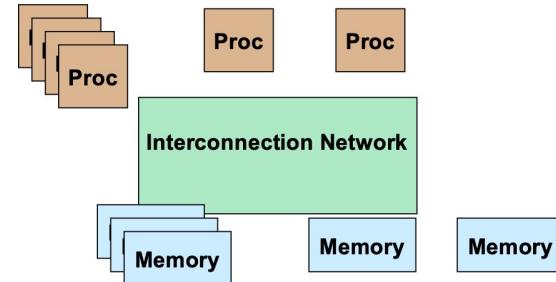
James Demmel

[www.cs.berkeley.edu/~demmel](http://www.cs.berkeley.edu/~demmel)

CS267 Lecture 5

1

### A generic parallel architecture

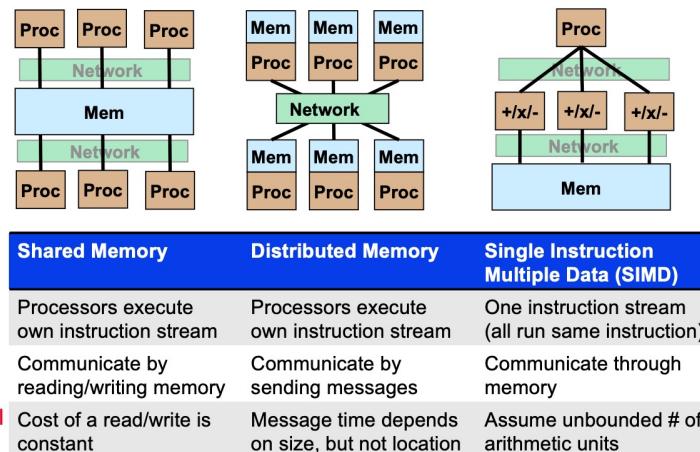


- Where is the memory physically located?
- Is it connected directly to processors?
- What is the connectivity of the network?
- How are the processors controlled?

CS267 - Lecture 5

2

### Parallel Machines and Programming



CS267 - Lecture 5

3

### Parallelism and Locality in Simulation

- Parallelism and data locality both critical to performance
  - Recall that moving data is the most expensive operation
- Real world problems have parallelism and locality:
  - Many objects operate independently of others.
  - Objects often depend much more on nearby than distant objects.
  - Dependence on distant objects can often be simplified.
    - Example of all three: particles moving under gravity
- Scientific models may introduce more parallelism:
  - When a continuous problem is discretized, time dependencies are generally limited to adjacent time steps.
    - Helps limit dependence to nearby objects (eg collisions)
  - Far-field effects may be ignored or approximated in many cases.
- Many problems exhibit parallelism at multiple levels

CS267 Lecture 5

4

## Basic Kinds of Simulation

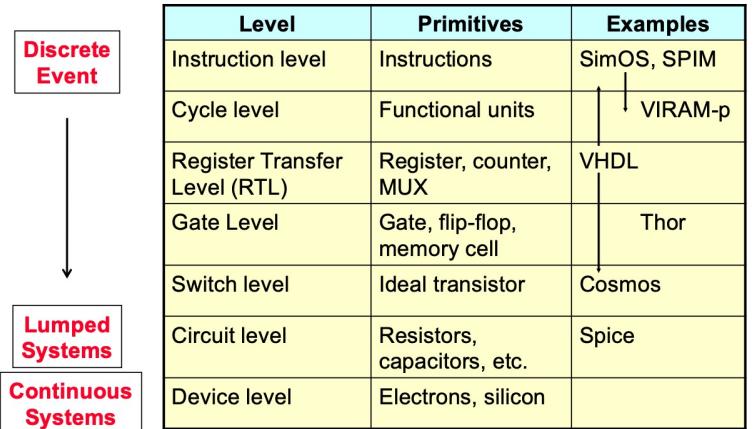
- Discrete event systems:
  - "Game of Life," Manufacturing systems, Finance, Circuits, Pacman, ...
- Particle systems:
  - Billiard balls, Galaxies, Atoms, Circuits, Pinball ...
- Lumped variables depending on continuous parameters
  - aka Ordinary Differential Equations (ODEs),
  - Structural mechanics, Chemical kinetics, Circuits, Star Wars: The Force Unleashed
- Continuous variables depending on continuous parameters
  - aka Partial Differential Equations (PDEs)
  - Heat, Elasticity, Electrostatics, Finance, Circuits, Medical Image Analysis, Terminator 3: Rise of the Machines
- A given phenomenon can be modeled at multiple levels.
- Many simulations combine more than one of these techniques.
- For more on simulation in games, see
  - [www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD](http://www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD)

CS267 Lecture 5

5

## Example: Circuit Simulation

- Circuits are simulated at many different levels



CS267 Lecture 5

6

## Outline

- Discrete event systems
  - Time and space are discrete
- Particle systems
  - Important special case of lumped systems
- Lumped systems (ODEs)
  - Location/entities are discrete, time is continuous
- Continuous systems (PDEs)
  - Time and space are continuous
  - Next lecture
- Identify common problems and solutions



CS267 Lecture 5

7

## A Model Problem: Sharks and Fish

- Illustration of parallel programming
  - Original version (discrete event only) proposed by Geoffrey Fox
  - Called WATOR
- Basic idea: sharks and fish living in an ocean
  - rules for movement (discrete and continuous)
  - breeding, eating, and death
  - forces in the ocean
  - forces between sea creatures
- 6 problems (S&F1 - S&F6)
  - Different sets of rules, to illustrate different phenomena
- Available in various languages (see class web page)
- Some homework based on these

CS267 Lecture 5

8

## **Sharks and Fish**

---

- **S&F 1.** Fish alone move continuously subject to an external current and Newton's laws.
- **S&F 2.** Fish alone move continuously subject to gravitational attraction and Newton's laws.
- **S&F 3.** Fish alone play the "Game of Life" on a square grid.
- **S&F 4.** Fish alone move randomly on a square grid, with at most one fish per grid point.
- **S&F 5.** Sharks and Fish both move randomly on a square grid, with at most one fish or shark per grid point, including rules for fish attracting sharks, eating, breeding and dying.
- **S&F 6.** Like Sharks and Fish 5, but continuous, subject to Newton's laws.

CS267 Lecture 5

9

# **Discrete Event Systems**

CS267 Lecture 5

10

## **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:”
  - Also known as Sharks and Fish #3:
  - Space divided into cells, rules govern cell contents at each step

CS267 Lecture 5

11

## **Parallelism in Game of Life (S&F 3)**

---

- The simulation is synchronous
  - use two copies of the grid (old and new), “ping-pong” between them
  - 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  
finish updates  
until done simulating

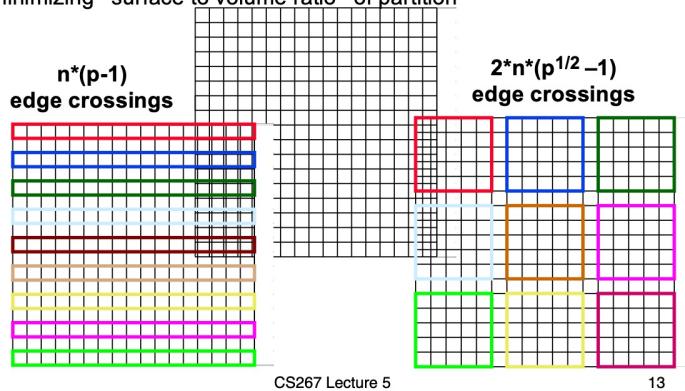
- Locality is achieved by using large patches of the ocean
  - Only boundary values from neighboring patches are needed.
- How to pick shapes of domains?

CS267 Lecture 5

12

## Regular Meshes (e.g. Game of Life)

- Suppose graph is  $n \times n$  mesh with connection NSEW neighbors
- Which partition has less communication? ( $n=18$ ,  $p=9$ )
- Minimizing communication on mesh  $\equiv$  minimizing “surface to volume ratio” of partition

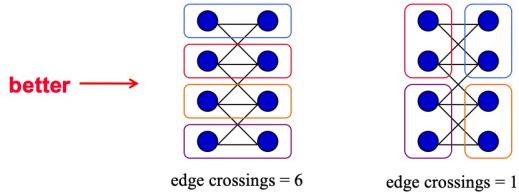


CS267 Lecture 5

13

## Synchronous Circuit Simulation

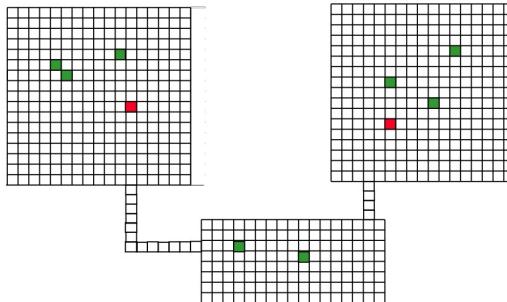
- Circuit is a **graph** made up of subcircuits connected by wires
  - Component simulations need to interact if they share a wire.
  - Data structure is (irregular) graph of subcircuits.
  - Parallel algorithm is timing-driven or **synchronous**:
    - Evaluate all components at every timestep (determined by known circuit delay)
- Graph partitioning** assigns subgraphs to processors
  - Determines parallelism and locality.
  - Goal 1 is to evenly distribute subgraphs to nodes (load balance).
  - Goal 2 is to minimize edge crossings (minimize communication).
  - Easy for meshes, NP-hard in general, so we will approximate (future lecture)



CS267 Lecture 5

14

## Sharks & Fish in Loosely Connected Ponds



- Parallelization: each processor gets a set of ponds with roughly equal total area
  - work is proportional to area, not number of creatures
- One pond can affect another (through streams) but infrequently

CS267 Lecture 5

15

## 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
  - On distributed memory, events are naturally implemented as messages between processors (eg using MPI), but how do you know when to execute a “receive”?

CS267 Lecture 5

16

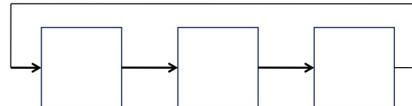
## Scheduling Asynchronous Circuit Simulation

- **Conservative:**
  - Only simulate up to (and including) the minimum time stamp of inputs.
  - Need deadlock detection if there are cycles in graph
    - Example on next slide
  - Example: Pthor circuit simulator in Splash1 from Stanford.
- **Speculative (or Optimistic):**
  - Assume no new inputs will arrive and keep simulating.
  - May need to backup if assumption wrong, using timestamps
  - Example: Timewarp [D. Jefferson], Parswec [Wen,Yelick].
- Optimizing load balance and locality is difficult:
  - Locality means putting tightly coupled subcircuit on one processor.
  - Since “active” part of circuit likely to be in a tightly coupled subcircuit, this may be bad for load balance.

CS267 Lecture 5

17

## Deadlock in Conservative Asynchronous Circuit Simulation

- Example: Sharks & Fish 3, with 3 processors simulating 3 ponds connected by streams along which fish can move

```
graph LR; P1[Processor 1] --> P2[Processor 2]; P2 --> P3[Processor 3]; P3 --> P1;
```
- Suppose all ponds simulated up to time  $t_0$ , but no fish move, so no messages sent from one proc to another
  - So no processor can simulate past time  $t_0$
- Fix: After waiting for an incoming message for a while, send out an “Are you stuck too?” message
  - If you ever receive such a message, pass it on
  - If you receive such a message that you sent, you have a deadlock cycle, so just take a step with latest input
- Can be a serial bottleneck

CS267 Lecture 5

18

## Summary of Discrete Event Simulations

- Model of the world is discrete
  - Both time and space
- Approaches
  - Decompose domain, i.e., set of objects
  - Run each component ahead using
    - **Synchronous:** communicate at end of each timestep
    - **Asynchronous:** communicate on-demand
      - **Conservative scheduling** – wait for inputs
        - need deadlock detection
      - **Speculative scheduling** – assume no inputs
        - roll back if necessary

CS267 Lecture 5

19

## Particle Systems

CS267 Lecture 5

21

## Particle Systems

- A particle system has
  - a finite number of particles
  - moving in space according to Newton's Laws (i.e.  $F = ma$ )
  - Time and positions are 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
- Reminder: many simulations combine techniques such as particle simulations with some discrete events (Ex Sharks and Fish)

CS267 Lecture 5

22

## Example S&F 1: 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)
% Algorithm: integrate using Euler's method with varying step size
% Initialize time step, iteration count, and array of times
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
```

CS267 Lecture 5

24

## Forces in Particle Systems

- Force on each particle can be subdivided  
 $\text{force} = \text{external\_force} + \text{nearby\_force} + \text{far\_field\_force}$
- External force
  - ocean current in sharks and fish world (S&F 1)
  - externally imposed electric field in electron beam
- Nearby force
  - sharks attracted to eat nearby fish (S&F 5)
  - balls on a billiard table bounce off of each other
  - Van der Waals forces in fluid ( $1/r^6$ ) ... how Gecko feet work?
- Far-field force
  - fish attract other fish by gravity-like ( $1/r^2$ ) force (S&F 2)
  - gravity, electrostatics, radiosity in graphics
  - forces governed by elliptic PDE

CS267 Lecture 5

23

## Parallelism in External Forces

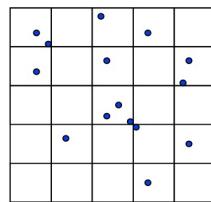
- These are the simplest
- The force on each particle is independent
- Called "embarrassingly parallel"
  - Sometimes called "map reduce" by analogy
- Evenly distribute particles on processors
  - Any distribution works
  - Locality is not an issue
- For each particle on processor, apply the external force
  - Also called "map" (eg absolute value)
  - May need to "reduce" (eg compute maximum) to compute time step, other data

CS267 Lecture 5

25

## 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.

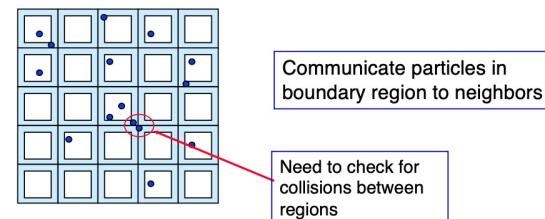


CS267 Lecture 5

26

## 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” or “halo”
  - **Low surface to volume ratio means low communication.**
  - Use squares, not slabs, to minimize ghost zone sizes

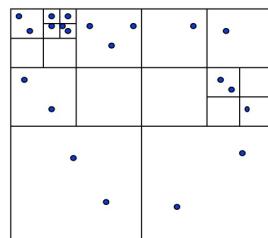


CS267 Lecture 5

27

## 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.



- May need to rebalance as particles move, hopefully seldom

CS267 Lecture 5

28

## 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  $O(n^2)$  as in S&F 2, 4, 5.
  - Just decomposing space does not help since every particle needs to “visit” every other particle.



Implement by rotating particle sets.

- Keeps processors busy
- All processors eventually see all particles

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

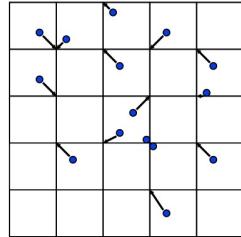
CS267 Lecture 5

29

## **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 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)

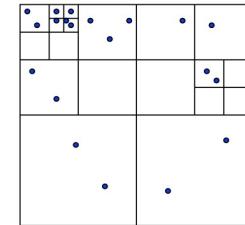


CS267 Lecture 5

30

## **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/Rokhlin.
  - Anderson’s method.
- Discussed in later lecture.



CS267 Lecture 5

31

## **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 approximate 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

CS267 Lecture 5

32

## **Lumped Systems: ODEs**

CS267 Lecture 5

33

## System of Lumped Variables

- Many systems are approximated by
  - System of “lumped” variables.
  - Each depends on continuous parameter (usually time).
- Example -- circuit:
  - approximate as graph.
    - wires are edges.
    - nodes are connections between 2 or more wires.
    - each edge has resistor, capacitor, inductor or voltage source.
  - system is “lumped” because we are not computing the voltage/current at every point in space along a wire, just endpoints.
  - Variables related by Ohm's Law, Kirchoff's Laws, etc.
- Forms a system of ordinary differential equations (ODEs).
  - Differentiated with respect to time
  - Variant: ODEs with some constraints
    - Also called DAEs, Differential Algebraic Equations

CS267 Lecture 5

34

## Structural Analysis Example

- Another example is structural analysis in civil engineering:
  - Variables are displacement of points in a building.
  - Newton's and Hook's (spring) laws apply.
  - Static modeling: exert force and determine displacement.
  - Dynamic modeling: apply continuous force (earthquake).
  - Eigenvalue problem: do the resonant modes of the building match an earthquake?



OpenSees project in CE at Berkeley looks at this section of 880, among others

CS267 Lecture 5

36

## Circuit Example

- State of the system is represented by
  - $v_n(t)$  node voltages
  - $i_b(t)$  branch currents
  - $v_b(t)$  branch voltages

} all at times t
- Equations include
  - Kirchoff's current
  - Kirchoff's voltage
  - Ohm's law
  - Capacitance
  - Inductance
$$\begin{pmatrix} 0 & A & 0 \\ A' & 0 & -I \\ 0 & R & -I \\ 0 & -I & C \cdot d/dt \\ 0 & L \cdot d/dt & I \end{pmatrix} * \begin{pmatrix} v_n \\ i_b \\ v_b \end{pmatrix} = \begin{pmatrix} 0 \\ S \\ 0 \\ 0 \\ 0 \end{pmatrix}$$
- A is sparse matrix, representing connections in circuit
  - One column per branch (edge), one row per node (vertex) with +1 and -1 in each column at rows indicating end points
- Write as single large system of ODEs or DAEs

CS267 Lecture 5

35

## Gaming Example

### **Star Wars – The Force Unleashed ...**

[www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD](http://www.cs.berkeley.edu/b-cam/Papers/Parker-2009-RTD)

CS267 Lecture 5

37

## Solving ODEs

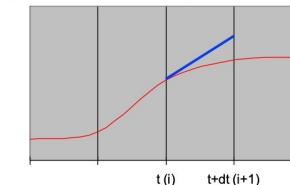
- In these examples, and most others, the matrices are sparse:
  - i.e., most array elements are 0.
  - neither store nor compute on these 0's.
  - Sparse because each component only depends on a few others
- Given a set of ODEs, two kinds of questions are:
  - Compute the values of the variables at some time t
    - Explicit methods
    - Implicit methods
  - Compute modes of vibration
    - Eigenvalue problems

CS267 Lecture 5

38

## Solving ODEs: Explicit Methods

- Assume ODE is  $x'(t) = f(x) = A \cdot x(t)$ , where A is a sparse matrix
  - Compute  $x(i^*dt) = x[i]$  at  $i=0,1,2,\dots$
  - ODE gives  $x'((i+1)^*dt) = \text{slope}$   
 $x[i+1] = x[i] + dt \cdot \text{slope}$
- Explicit methods, e.g., (Forward) Euler's method.
  - Approximate  $x'(t) = A \cdot x(t)$  by  $(x[i+1] - x[i]) / dt = A \cdot x[i]$ .
  - $x[i+1] = x[i] + dt \cdot A \cdot x[i]$ , i.e. sparse matrix-vector multiplication.
- Tradeoffs:
  - Simple algorithm: sparse matrix vector multiply.
  - Stability problems: May need to take very small time steps, especially if system is **stiff** (i.e. A has some large entries, so x can change rapidly).

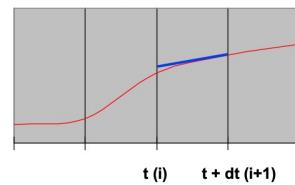


CS267 Lecture 5

39

## Solving ODEs: Implicit Methods

- Assume ODE is  $x'(t) = f(x) = A \cdot x(t)$ , where A is a sparse matrix
  - Compute  $x(i^*dt) = x[i]$  at  $i=0,1,2,\dots$
  - ODE gives  $x'((i+1)^*dt) = \text{slope}$   
 $x[i+1] = x[i] + dt \cdot \text{slope}$
- Use slope at  $x[i+1]$
- Implicit method, e.g., Backward Euler solve:
  - Approximate  $x'(t) = A \cdot x(t)$  by  $(x[i+1] - x[i]) / dt = A \cdot x[i+1]$ .
  - $(I - dt \cdot A) \cdot x[i+1] = x[i]$ , i.e. we need to solve a sparse linear system of equations.
- Trade-offs:
  - Larger timestep possible: especially for **stiff** problems
  - More difficult algorithm: need to solve a sparse linear system of equations at each step



CS267 Lecture 5

40

## Solving ODEs: Eigensolvers

- Computing modes of vibration: finding eigenvalues and eigenvectors.
- Seek solution of  $d^2x(t)/dt^2 = A \cdot x(t)$  of form  $x(t) = \sin(\omega * t) * x_0$ , where  $x_0$  is a constant vector
  - $\omega$  called the frequency of vibration
  - $x_0$  sometimes called a "mode shape"
- Plug in to get  $-\omega^2 * x_0 = A \cdot x_0$ , so that  $-\omega^2$  is an eigenvalue and  $x_0$  is an eigenvector of A.
- Solution schemes reduce either to sparse-matrix multiplications, or solving sparse linear systems.

CS267 Lecture 5

41

## Summary of ODE Methods

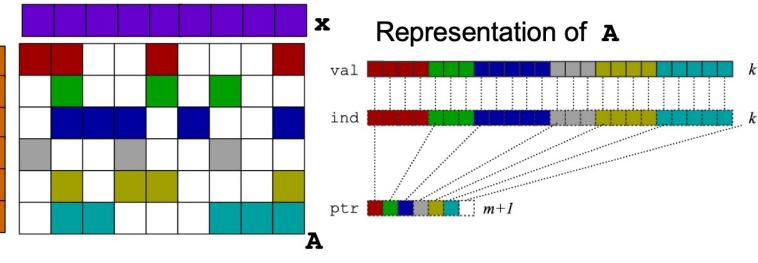
- Explicit methods for ODEs need sparse-matrix-vector mult.
- Implicit methods for ODEs need to solve linear systems
- Direct methods (Gaussian elimination)
  - Called LU Decomposition, because we factor  $A = L \cdot U$ .
  - Future lectures will consider both dense and sparse cases.
  - More complicated than sparse-matrix vector multiplication.
- Iterative solvers
  - Will discuss several of these in future.
    - Jacobi, Successive over-relaxation (SOR), Conjugate Gradient (CG), Multigrid,...
  - Most have sparse-matrix-vector multiplication in kernel.
- Eigenproblems
  - Future lectures will discuss dense and sparse cases.
  - Also depend on sparse-matrix-vector multiplication, direct methods.

CS267 Lecture 5

42

## SpMV in Compressed Sparse Row (CSR) Format

SpMV:  $y = y + A^*x$ , only store, do arithmetic, on nonzero entries  
 CSR format is simplest one of many possible data structures for A



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]]
```

CS267 Lecture 5

43

## Parallel Sparse Matrix-vector multiplication

- $y = A^*x$ , where A is a sparse  $n \times n$  matrix
- Questions
  - which processors store
    - $y[i]$ ,  $x[i]$ , and  $A[i,j]$
  - which processors compute
    - $y[i] = \text{sum}(\text{from } 1 \text{ to } n) A[i,j] * x[j]$
    - $= (\text{row } i \text{ of } A) * x$  ... a sparse dot product
- Partitioning
  - Partition index set  $\{1, \dots, n\} = N_1 \cup N_2 \cup \dots \cup N_p$ .
  - For all  $i \in N_k$ , Processor  $k$  stores  $y[i]$ ,  $x[i]$ , and row  $i$  of  $A$ .
  - For all  $i \in N_k$ , Processor  $k$  computes  $y[i] = (\text{row } i \text{ of } A)^* x$ 
    - "owner computes" rule: Processor  $k$  computes the  $y[i]$ s it owns.

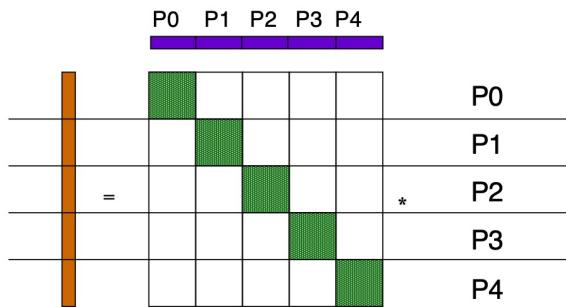


CS267 Lecture 5

44

## 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



CS267 Lecture 5

45

## Goals of Reordering

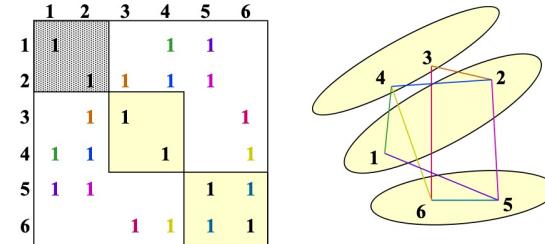
- Performance goals
  - balance load (how is load measured?).
    - Approx equal number of nonzeros (not necessarily rows)
  - balance storage (how much does each processor store?).
    - Approx equal number of nonzeros
  - minimize communication (how much is communicated?).
    - Minimize nonzeros outside 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 rows/columns for other reasons
  - Reduce # nonzeros in matrix after Gaussian elimination
  - Improve numerical stability

CS267 Lecture 5

46

## Graph Partitioning and Sparse Matrices

- Relationship between matrix and graph



- Edges in the graph are nonzero in the matrix: here the matrix is symmetric (edges are unordered) and weights are equal (1)
- If divided over 3 procs, there are 14 nonzeros outside the diagonal blocks, which represent the 7 (bidirectional) edges

CS267 Lecture 5

47

## Summary: Common Problems

- Load Balancing
  - Statically - Graph partitioning
    - Discrete event simulation
    - Sparse matrix vector multiplication
  - Dynamically – if load changes significantly during job
- Linear algebra
  - Solving linear systems (sparse and dense)
  - Eigenvalue problems will use similar techniques
- Fast Particle Methods
  - $O(n \log n)$  instead of  $O(n^2)$

CS267 Lecture 5

48

## What do commercial and CSE applications have in common?

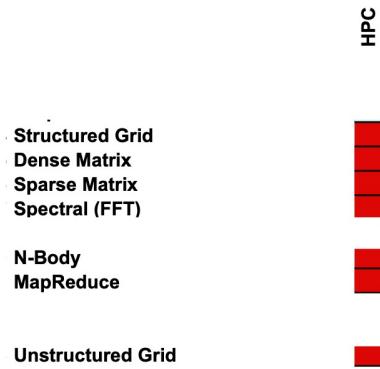
### **Motif/Dwarf: Common Computational Methods**



49

## What do commercial and CSE applications have in common?

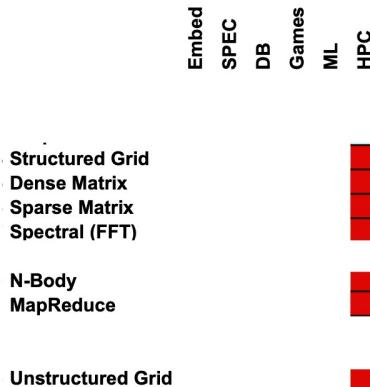
### Motif/Dwarf: Common Computational Methods



50

## What do commercial and CSE applications have in common?

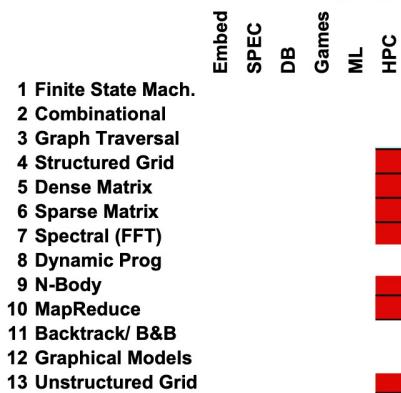
### Motif/Dwarf: Common Computational Methods



51

## What do commercial and CSE applications have in common?

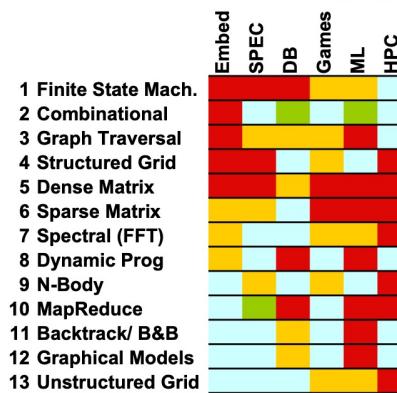
### Motif/Dwarf: Common Computational Methods



52

## What do commercial and CSE applications have in common?

### Motif/Dwarf: Common Computational Methods (Red Hot → Blue Cool)



53