CS 267 Applications of Parallel Computers

Lecture 10:

Sources of Parallelism and Locality

James Demmel

http://www.cs.berkeley.edu/~demmel/cs267_Spr99

Recap: Parallel Models and Machines

° **Machine models**
- shared memory
- distributed memory
- SIMD

Programming models
- threads
- message passing
- data parallel
- shared address space

° **Steps in creating a parallel program**
- decomposition
- assignment
- orchestration
- mapping

° **Performance in parallel programs**
- try to minimize performance loss from
  - load imbalance
  - communication
  - synchronization
  - extra work

Outline

° Simulation models

° A model problem: sharks and fish

° Discrete event systems

° Particle systems

° Lumped systems (Ordinary Differential Equations, ODEs)

° (Next time: Partial Different Equations, PDEs)

Simulation Models
and
A Simple Example

Sources of Parallelism and Locality in Simulation

° **Real world problems have parallelism and locality**
  - Many objects do not depend on other objects
  - Objects often depend more on nearby than distant objects
  - Dependence on distant objects often "simplifies"

° **Scientific models may introduce more parallelism**
  - When continuous problem is discretized, may limit effects to timesteps
  - Far-field effects may be ignored or approximated, if they have little effect

° **Many problems exhibit parallelism at multiple levels**
  - e.g., circuits can be simulated at many levels and within each there may be parallelism within and between subcircuits

Basic Kinds of Simulation

° **Discrete event systems**
  - e.g.,"Game of Life", timing level simulation for circuits

° **Particle systems**
  - e.g., billiard balls, semiconductor device simulation, galaxies

° **Lumped variables depending on continuous parameters**
  - ODEs, e.g., circuit simulation (Spice), structural mechanics, chemical kinetics

° **Continuous variables depending on continuous parameters**
  - PDEs, e.g., heat, elasticity, electrostatics

° **A given phenomenon can be modeled at multiple levels**

° **Many simulations combine these modeling techniques**

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 rule, to illustrate different phenomena

° **Available in Matlab, Threads, MPI, Split-C, Titanium, CMF, CMMD, pSather**
  - not all problems in all languages

° **www.cs.berkeley.edu/~demmel/cs267/Sharks_and_Fish**
  - being updated

Discrete Event Systems

Discrete Event Systems

° **Systems are represented as**
- finite set of variables
- each variable can take on a finite number of values
- 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 finite 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

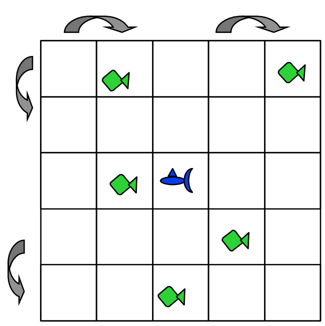
° **E.g., functional level circuit simulation**
- state is represented by a set of boolean variables (high & low voltages)
- set of logical rules defining state transitions (and, or, etc.)
- synchronous: only interested in state at clock ticks

Sharks and Fish as Discrete Event System

° **Ocean modeled as a 2D toroidal grid**

° **Each cell occupied by at most one sea creature**

° **S&F3, 4 and 5 are variations on this**



The Game of Life (Sharks and Fish 3)

° **Fish only, no sharks**

° **An new fish is born if**
- a cell is empty
- exactly 3 (of 8) neighbors contain fish

° **A fish dies (of overcrowding) if**
- cell contains a fish
- 4 or more neighboring cells are full

° **A fish dies (of loneliness) if**
- cell contains a fish
- less than 2 neighboring cells are full

° **Other configurations are stable**

Parallelism in Sharks and Fish

° **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, where each cell is updated at every timestep

° **Easy to parallelize using 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
```

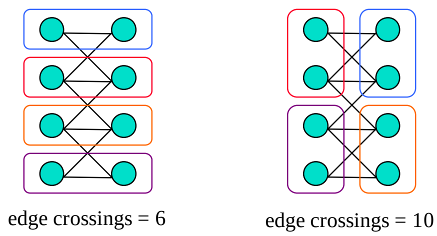
° **Locality is achieved by using large patches of the ocean**
- boundary values from neighboring patches are needed

° **If only cells next to occupied ones are visited (an optimization), then load balance is more difficult. The activities in this system are discrete events**

Parallelism in 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)
- parallel algorithm is synchronous
  - compute subcircuit outputs
  - propagate outputs to other circuits

° Graph partitioning assigns subgraphs to processors
- Determines parallelism and locality
- Want even distribution of nodes (load balance)
- With minimum edge crossing (minimize communication)
- Nodes and edges may both be weighted by cost
- NP-complete to partition optimally, but many good heuristics (later lectures)



Parallelism in Asynchronous Circuit Simulation

° Synchronous simulations may waste time
- simulate even when the inputs do not change, little internal activity
- activity varies across circuit

° Asynchronous simulations update only when an event arrives from another component
- no global timesteps, but events contain time stamp
- Ex: Circuit simulation with delays (events are gates changing)
- Ex: Traffic simulation (events are cars changing lanes, etc.)

Scheduling Asynchronous Circuit Simulation

° **Conservative:**
- Only simulate up to (and including) the minimum time stamp of inputs
- May need deadlock detection if there are cycles in graph, or else "null messages"
- Ex: Pthor circuit simulator in Splash1 from Stanford

° **Speculative:**
- Assume no new inputs will arrive and keep simulating, instead of waiting
- May need to backup if assumption wrong
- Ex: Parswec circuit simulator of Yelick/Wen
- Ex: Standard technique for CPUs to execute instructions

° **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

Particle Systems

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 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

° Reminder: many simulations combine techniques such as particle simulations with some discrete events (Ex Sharks and Fish)

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 (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 Wals forces in fluid (1/r^6)

° Far-field force
  - fish attract other fish by gravity-like (1/r^2) force (S&F 2)
  - gravity, electrostatics, radiosity
  - forces governed by elliptic PDE

Parallelism in External Forces

° These are the simplest

° The force on each particle is independent

° Called "embarrassingly parallel"

° Evenly distribute particles on processors

- Any distribution works
- Locality is not an issue, no communication

° For each particle on processor, apply the external force

Parallelism in Nearby Forces

° Nearby forces require interaction => communication

° Force may depend on other nearby particles

- Ex: collisions
- simplest algorithm is O(n^2): look at all pairs to see if they collide

° Usual parallel model is domain decomposition of physical domain

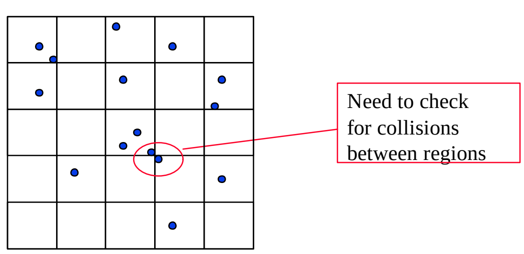
- O(n^2/p) particles per processor if evenly distributed

° Challenge 1: interactions of particles near processor boundary

- need to communicate particles near boundary to neighboring processors
- surface to volume effect means low communication
- Which communicates less: squares (as below) or slabs?

° Challenge 2: load imbalance, if particles cluster

- galaxies, electrons hitting a device wall

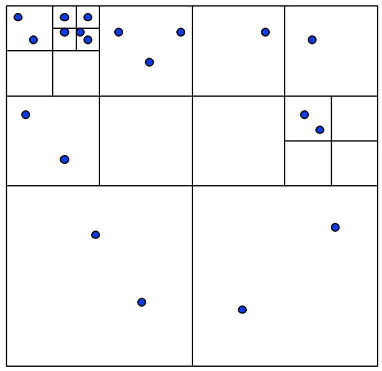


Load balance via Tree Decomposition

° 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 => communication

° Force depends on all other particles
- Ex: gravity
- Simplest algorithm is O(n^2) as in S&F 2, 4, 5
- Just decomposing space does not help since every particle apparently needs to "visit" every other particle

° Use more clever algorithms to beat O(n^2)

Far-field forces: Particle-Mesh Methods

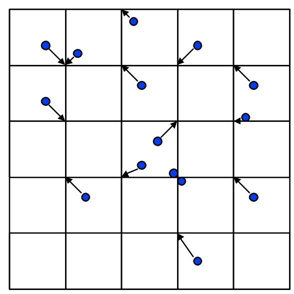
° Superimpose a regular mesh

° "Move" particles to nearest grid point

° Exploit fact that far-field satisfies a PDE that is easy to solve on a regular mesh

- FFT, Multigrid
- Wait for next lecture

° Accuracy depends on how fine the grid is and uniformity of particles



Far-field forces: Tree Decomposition

° Based on approximation

° O(n log n) or O(n) instead of O(n^2)

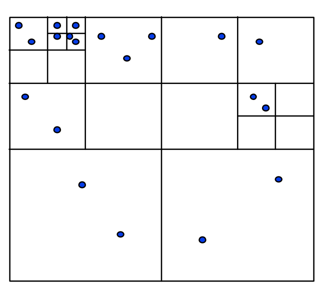
° Forces from group of far-away particles "simplifies"

- They resemble a single larger particle

° Use tree; each node contains an approximation of descendents

° Several Algorithms

- Barnes-Hut
- Fast Multipole Method (FMM) of Greengard/Rohklin
- Anderson
- Later lectures



Lumped Systems
ODEs

System of Lumped Variables

° **Many systems 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.

° **Form a system of Ordinary Differential Equations, ODEs**

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 time t

- Equations include
  - Kirchoff's current
  - Kirchoff's voltage
  - Ohm's law
  - Capacitance
  - Inductance

$$\begin{pmatrix} 0 & A & 0 \\ A' & 0 & -I \\ 0 & R & -I \\ 0 & -I & C*d/dt \\ 0 & L*d/dt & I \end{pmatrix} * \begin{pmatrix} v\_n \\ i\_b \\ v\_b \end{pmatrix} = \begin{pmatrix} 0 \\ S \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Write as single large system of ODEs (possibly with constraints)

Systems of Lumped Variables

- Another example is structural analysis in Civil Eng.
  - 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)
- The system in these case (and many) will be sparse
  - i.e., most array elements are 0
  - neither store nor compute on these 0's

Solving ODEs

° Explicit methods to compute solution(t)
  - Ex: Euler's method
  - Simple algorithm: sparse matrix vector multiply
  - May need to take very small timesteps, especially if system is stiff (i.e. can change rapidly)

° Implicit methods to compute solution(t)
  - Ex: Backward Euler's Method
  - Larger timesteps, especially for stiff problems
  - More difficult algorithm: solve a sparse linear system

° Computing modes of vibration
  - Finding eigenvalues and eigenvectors
  - Ex: do resonant modes of building match earthquakes?

° All these reduce to sparse matrix problems
  - Explicit: sparse matrix-vector multiplication
  - Implicit: solve a sparse linear system
    - direct solvers (Gaussian elimination)
    - iterative solvers (use sparse matrix-vector multiplication)
  - Eigenvalue/vector algorithms may also be explicit or implicit

Parallelism in Sparse Matrix-vector multiplication

° y = A*x, where A is sparse and n x n

° Questions
  - which processors store
    - y[i], x[i], and A[i,j]
  - which processors compute
    - x[i] = sum from 1 to n or A[i,j] * x[j]

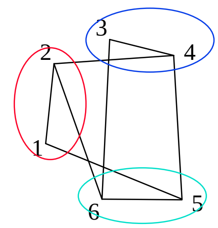
° Graph partitioning
  - Partition index set {1,…,n} = N1 u N2 u … u Np
  - for all i in Nk, store y[i], x[i], and row i of A on processor k
  - Processor k computes its own y[i]

° Constraints
  - balance load
  - balance storage
  - minimize communication

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



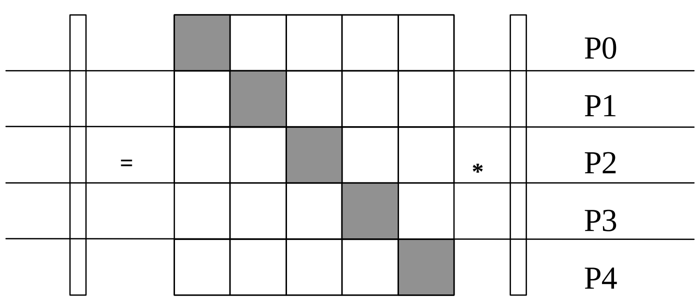
° **A "good" partition of the graph has**
- **equal number of (weighted) nodes in each part (load balance)**
- **minimum number of edges crossing between (minimize communication)**

° **Can reorder the rows/columns of the matrix by putting all the nodes in one partition together**

More on Matrix Reordering via Graph Partitioning

° **Goal is to reorder rows and columns to**
- **improve load balance**
- **decrease communication**

° **"Ideal" matrix structure for parallelism: (nearly) block diagonal**
- **p (number of processors) blocks**
- **few non-zeros outside these blocks, since these require communication**



What about implicit methods and eigenproblems?

° **Direct methods (Gaussian elimination)**
  - future lectures will consider both dense and sparse cases

° **Iterative solvers**
  - future lectures will discuss several of these
  - most have sparse-matrix-vector multiplication in kernel

° **Eigenproblems**
  - future lectures will discuss dense and sparse cases
  - depends on student interest