

Lecture 8: February 22

*Lecturer: Horst Simon**Scribes: Jonathan Ellithorpe*

Disclaimer: *These notes have not been subjected to the usual scrutiny reserved for formal publications. They may be distributed outside this class only with the permission of the Instructor.*

8.1 Parallelism and Locality in Simulation

- Many real world problems exhibit dependencies with spacial locality. This allows us to partition the space and operate in parallel on those spaces.
- Sometimes there are long range dependencies (i.e. gravitational forces) that force all-to-all interactions between objects. In general these interactions can be simplified.
- Scientific models can discretize continuous environments to allow for parallelism. Generally time steps only depend on adjacent time steps.
- Multilevel parallism:
 - Can exploit parallelism and multiple levels of abstraction
 - Example: Circuits. Can be simulated at a high level in a graph with edges as wires. Can simulate low level (physics) as well, in parallel.

8.2 Basic Types of Simulation

1. Discrete Event Systems
 - Time and space are discrete
2. Particle Systems
 - Important special case of lumped systems
3. Lumped Variables depending on continuous parameters
 - Local objects move together as time moves forward
4. Continuous variables depending on continuous parameters

A given simulation may combine many of these techniques at multiple levels. The following is an example of circuit simulation broken down into multiple levels (highest level of abstraction to lowest):

1. ISA (instructions)
2. Cycle level (functional units)
3. Register Transfer Level

4. Gate Level (DQ flip flop, setup time, etc)
5. Switch level (MOSFET configurations)
6. Circuit level (resistor, capacitor)
7. Device Level (electrons)

8.3 Classic Problem: Sharks and Fishes

- Interaction between elements with various forces between elements
- 6 "flavors" of this problem, with different rules of interaction and with increasing complexity
 1. Just Fish. Fish move continuously only as a result of surrounding current and Newton's Laws.
 2. Just Fish. Fish move continuously according to gravitation between them and surrounding current.
 3. Just Fish. Fish play "game of life" on square grid
 4. Just Fish. Fish move randomly on 2D grid, with at most 1 fish per grid point.
 5. Sharks and Fish. Move randomly on 2D grid, w/ at most 1 animal per grid point. Includes rules for fish attracting sharks, eating breeding, etc.
 6. Sharks and Fish. Move continuously on 2D space. Subject to Newton's Laws.

8.4 Discrete Event Systems

System Definition:

- Finite set of variables
- Variable values at time step t called state
- Depending on the current value of variables and the transition functions we can compute the new value of the variables that describe the system.
- Also called "state machine"

Two types of systems:

1. Synchronous
 - Compute next state at predefined time steps
 - Example 1: Game of Life:
 - (a) Can divide the space into domains, assign each domain to a processor, and the processors will need to communicate their boundary state to each other at each time step.
 - (b) How do we divide up the space?
 - (c) Goal 1: minimize processor communication == maximize volume to surface area of your domains. (Surface area proportional to communication).
 - (d) Goal 2: Load balance give equal work to each processor

- Example 2: Circuit Simulation:
 - (a) Circuit is a graph
 - i. Nodes: Cluster of elements
 - ii. Edges: Wires
 - (b) Partitioning the graph
 - i. Minimize edge crossing (and thus communication)
 - ii. Evenly distribute subgraphs to nodes (load balancing per processor)
 - iii. NP-Hard in general
- Synchronous systems may waste time because the inputs may not be changing between each time step

2. Asynchronous

- Compute next state when inputs change
- Also called event driven
- More efficient than Synchronous systems but harder to parallize - When to schedule receives?
- Scheduling Asynchronous Systems:
 - Conservative:
 - * Only act on the most recent complete state of the system that you are aware of (you may have a more recent partial state, but do not act on this because incomplete).
 - Aggressive:
 - * Predict based on partial information, backtrack if prediction was wrong
 - * More complicated

8.5 Particle Systems

System Definition:

- Finite number of particles
- Moving in space governed by physics
- Time is continuous

Examples: Galaxies, electron beam, atoms in space.

Forces:

- External Force
- Nearby Force
- Far Field Force - Gravity, electrostatics, radioactivity

8.5.1 Exploiting Parallelism in Nearby Forces

- Requires communication between nearby particles
- Forces communication between processors
- Can divide up the space to maximize volume to surface area keep track of information across boundaries.
- Also need to load balance.
 - Can have irregular particle density, so must divide up the space to balance the load and minimize communication simultaneously
 - Example space partitioning: Quad-tree in 2D, Oct-tree in 3D
 - May have to re-partition space as particle density changes
- We can approximate a limit to the range of the nearby force, thus limiting how much space near the boundaries are in need of processor-processor communication.

8.5.2 Exploiting Parallelism in Far Field Forces

- One particle affects all other particles
- Simplest algorithm is order n^2
- More clever algorithms exist
- Particle Mesh Methods
 - Move particles to nearest grid point
 - Becomes PDE problem that's easy to solve on a regular mesh
 - More accurate the finer the grid points.
 - Forces interpolated at particles
- Tree decomposition
 - Find center of mass of group of particles, then simplify as single large particle
 - Use tree- each node is approximation of all nodes below.
 - Several Algorithms available:
 - * Barnes-Hut
 - * Fast multipole method
 - * Anderson's method

8.6 Lumped Systems (ODEs)

System Definition:

- Each variable is a function of time (or some other continuous parameter).

Solving ODEs:

- For these problems the matrices are generally sparse
- Each component only depends on a few others
- Given set of ODEs we have two questions:
 - Compute the values of the variables at some time t
 - * Explicit Methods
 - i.e. Forward Euler's method
 - Use slope of all variables at current time to progress a tiny time step forward.
 - Pro: Sparse matrix vector multiply
 - Con: Stability problems: may have to take really small time steps to get anything even remotely accurate.
 - * Implicit Methods
 - Use slope of all variables at next time step to backwards solve for the next time step value
 - Pro: Larger time steps possible
 - Con: More difficult problem: must do sparse solve at each time step.
 - Compute modes of vibration
 - * Eigensolvers - Solution schemes reduce either to sparse-matrix multiplication, or solving sparse linear systems.
 - * Implicit Methods
 - * Direct Solvers - Based on Gaussian elimination and LU Decompositions
 - * Iterative Solvers - Most have sparse-matrix-vector multiplication in the kernel
- All these problems reduce to sparse matrix problems (thus, Sparse Matrix Vector multiply is key to solving these problems)
- Two Key issues: locality and load balancing
- Use CSR (compressed sparse row) matrix representation to represent the sparse matrix for efficiency.
 - Key idea: store only nonzero values of the matrix in memory
- Parallelizing Sparse Matrix vector multiplication
 - Divide up the rows for each processor.
 - Each processor needs all of x ($y = y + A*x$)
 - Allows each processor to be responsible for a particular section of the y vector