Parallel Algorithms

Lecture 9: Simulations

Ronald Veldema

Introduction

- Most large scale scientific computing performed is some form of simulation
  - Simulation can always use
    - A few more objects to simulate
    - Smaller timesteps
    - More timesteps
    - More precision
    - Real time behaviour
      - Computational steering

Grand Challenges

- "Grand Challenge applications are fundamental problems in science and engineering with broad economic and scientific impact. They are generally considered intractable without the use of state-of-the-art massively parallel computers. "

Grand challenge: modelling the Sun

- Why ?
  - Solar flare prediction, improve general physics, etc
  - There exist a number of theoretic models for the Sun
  - There are numerous observations (xray/visual/magnetic)
- Which model is (most) correct?
  - Processes not well understood
  - Simulation is the only way to tell…
    - 3D and O(1024^3), 60TByte memory, 4000FP's per grid point
    - Multi teraflop range computing…

"fixed time step"

- A simulation system
  - Integration over time
  - Each time step, all individuals in the simuation are updated by advancing 'simulated time' by a constant delta

"variable time step"

- Integration over time
- Whenever the simulation becomes 'interesting' take smaller time steps

Monte Carlo Simulations

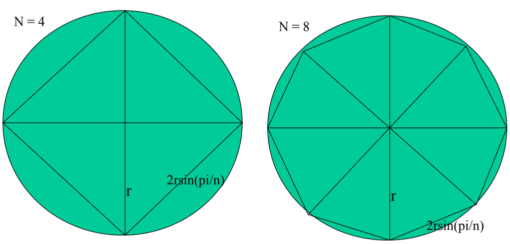
- Start with 'random' or 'reasonable' initial conditions
  - For example, place simulated individuals somewhere in a grid
- Pick a random individual
  - Move in random way
  - Check if movement is allowed
    - If allowed, update the whole system to take movement into account
    - If not allowed, take back movement as if it didn't happen
- http://sic.epfl.ch/SA/publications/SCR95/7-95-21a.html

Finite Element Methods (1)

- What is a finite element ?
  - Take a continuum model
    - discretize.
    - Limit size of continuum
  - Each element of discretized continuum is a Finite element
  - Useful if
    - Global continuum system is too complex
      - Break it down into 'primitive elements'
      - Simulate the primitive elements seperately (divide & conquer style)
      - Sum the effects of the individual parts somehow to approximate the continuum

Finite Element Methods (2)

- Example: compute PI using the finite element method (d=2*r, L=pi*d, pi=L/d)
- N = 8, pi = 3.06   N = 32, pi = 3.13



Bio Computing (1)

- Has large computational requirements
  - DNA sequence alignment
  - Protein database search
  - Molecule matching (see if molecule X can be attached to molecule Y)

Bio Computing (2)

- DNA sequence alignment
  - DNA scanning machines deliver chunks of dna strings
    - We want the large complete string, not the fragments
  - Dna scans deliver **large** amounts of DNA fragments
  - DNA encoded as string of base pairs (A, C, T, G)
  - Human has 48 chromosomes, $*3*10^9$ bases

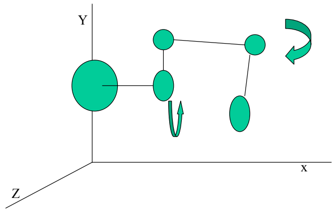
Bio Computing (3)

- DNA sequence alignment example

Have string
ACTGAGCTTCAC
And string
CACAGAGTATC
Head-tail match, thus make a larger string.
- use probability that it's the correct match before making the decision to merge
- potentially large numbers of possible matches to consider
- 3 Gbytes of input * N times for maintaining probable matches….

Bio Computing (4)

- Protein Folding problem
  - Given a sequence of amino-acid molecules, find the least energy 3D configuration
    - C3OH3CHCOGCS3…..



Bio Computing (4a)

- When able to predict the correct stable folding of an arbitrary protein
  - Can see if it ‘fits’ inside another molecule
    - If fit then possible medicin (protein blocker for other protein)
  - See if surface properties equal to other molecule in 3D
  - etc

Bio Computing (5)

- Protein Folding
  - Partially embarasingly parallel
    - All posible folding can be tried in parallel
      - Misses cut-offs
      - Testing if a state is possible is non trivial...

Bio Computing (6)

```
SequentialFindMinimumEnergyConfiguration(mol)
{
  Queue = empty
  Min_energy=energy(mol)
  Min_config=mol
  Put mol in queue
 while not queue is empty
        m = queue.get();
        for I=0 to #joints in m
              m’=twist joint I in m
              if(m’ is valid configuration)
                    put m’ in queue
                    if  energy(m’) < min_energy
                          min_energy = energy(m’)
                          min_config = m’
```

Bio Computing (7)

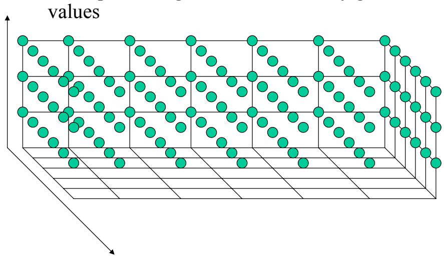
```
ParallelFindMinimumEnergyConfiguration(molecule mol)
Queue = empty
Min_energy=energy(mol)
Min_config=mol
Put mol in queue
Parallel while not queue is empty
      m = queue.get();
      for I=0 to #joints in m
            m’ = twist joint I in m
            if (m’ is valid configuration)
                  put m’ in queue
                  if  energy(m’) < min_energy
                        min_energy = energy(m’)
                        min_config = m’
```

Atmosphere modelling (1)

- Simulate wind, clouds, precipitation, etc that influence wind & weather
- Uses basic physics (mechanics, fluid property formulas)
  - Conservation of mass, energy and momentum
  - Hydrostatic approximation
  - Gas state equations
    - presure = density * temperature * height

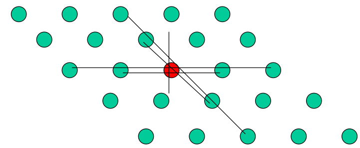
Atmosphere modelling (2)

- First try, put everything on a 3D grid
- Each grid point = 1 task
  - Note: points in grid don't move, they get different values



Atmosphere Modelling (3)

- Every grid point
  - Communicates with 11 others
  - Most communication is horizontal



Atmosphere modelling

- Agglomeration
  - Each grid point = 1 task
    - Nx * Ny * Nz tasks
      - Too many
  - Most communication is horizontal, thus agglomerate mostly horizontally
- Load imbalances
  - At night no radiation in physics model
  - Clouds only at threshold humidity
- Question: is this a finite element simulation ?

Particle Simulation: particle - particle method (1)

- Accumulate forces by finding the force *F(i,j)* of particle *j* on particle *i,*
- Integrate the equations of motion (which includes the accumulated forces), and
- Update the time counter.
- Repeat for the next time step.

Particle Simulation: Particle – Particle (2)

- Particle of mass M1 attracts other particle with mass M2 with:
  - $F = (G * M1 * M2 / r^3) * r$
    - G = gravitational constant
    - R = distance
  - Newton: F = MA, A = F/M
  - V = V + A
  - Pos = Pos + V, for each time step

Particle Simulation: Particle – Particle (3)

- With N particles: N-1 times the operations
- $O(N^2)$ complexity
- When particles are far apart, use large timesteps
  - When closeby, use smaller timestep

Particle Simulation: Barnes-Hut

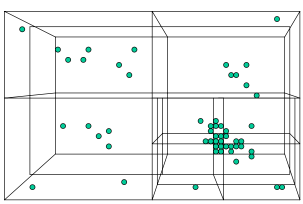
- Observation:
  - With the particle-particle method
    - particles that are far away deliver almost the same forces to a particle
    - If really far away, particles that are far away can be summarized into a ‘super-particle’

Particle Simulation: Barnes-Hut

1. Build a octtree
2. For each subcube in the octtree, compute the center of mass and total mass for all the particles it contains,
3. For each particle, traverse the tree to compute the force on it.

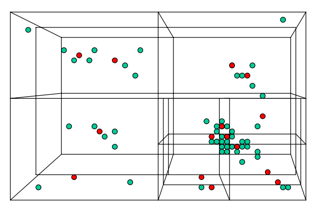
Particle Simulation: Barnes-Hut step 1: build octtree

- Step 1 communicates



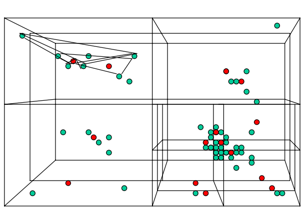
Particle Simulation: Barnes-Hut step 2:compute center of pass

- Within sub-cube, particle particle method. Compute forces to outside particles using center of pass of other subcubes.



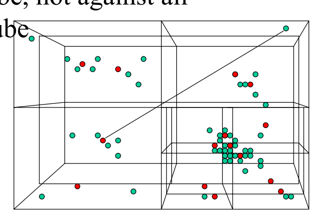
Particle Simulation: Barnes-Hut step 3: compute pairwise forces within cube

- Step 3 has no communication



Particle Simulation: Barnes-Hut step 4: compute intercube forces

- Step 4 may communicate
- Each particle computes force against center of mass of other cube, not against all particles in other cube



Particle Simulation: Barnes-Hut step 5: move particles

- Update position, velocity, acceleration of each particle
  - Pos += Velocity
  - Velocity += Acceleration
  - Acceleration = Sum directed forces / Mass
  - Etc (same as particle-particle method)

Rendering

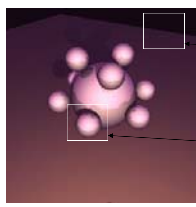
- Parallel raytracing
  - Simulate individual 'rays' of light
- Radiosity rendering
  - Simulate light as an amount of energy that is emitted by each surface

Raytracing (1)

- Shoot a ray from your eye towards each pixel of the screen
  - The ray's color is black
  - When hitting an object, bounce of it/into it etc
  - When hitting a light source, take over the light source's color
  - When returning from the recursion from a hit object, attenuate the ray's color

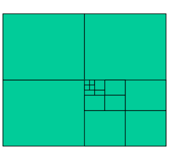
Raytracing (2)

- Note: each 'ray' of light is independent of all others
- Note: some rays are more computationally more complex than others



Raytracing (3)

- Assign to each processor a fixed partition of the screen ?
- Divide screen in fixed size squares ?
- Divide and conquer the screen ?



Radiosity (1)

- Divide objects of scene into patches
- Each patch receives some energy from all (visible) others
- Each patch emits some of its received energy back to all others

Radiosity (2)

```
For each patch P do
        P.energy = energy_from_lightsource(P);
        for each patch L do
                P.received_energy += L.energy *
                                     angle(P,L) *
                                     L.material
For each patch P do
        print on screen if P visible with color P.color * P.energy;
```

Radiosity (3)

- Create a par-for loop for each 'patch' ?
- Create a par-for loop for each patch'es 'gather energy' loop ?
- Can we optimize ?
  - Note: some patches not visible from others…
  - Note: energy transfer for some patches easier to compute then others (load imbalance !)