CS 267 Applications of Parallel Computers

Lecture 21:

Hierarchical Methods for the N-Body problem - I

James Demmel

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

Outline

- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force on N bodies takes $O(N^2)$ work
- **How to reduce the number of particles in the force sum**
  - We must settle for an approximate answer (say 2 decimal digits, or perhaps 16 …)
- **Basic Data Structures: Quad Trees and Oct Trees**
- **The Barnes-Hut Algorithm (BH)**
  - An O(N log N) approximate algorithm for the N-Body problem
- **The Fast Multipole Method (FMM)**
  - An O(N) approximate algorithm for the N-Body problem
- **Parallelizing BH, FMM and other algorithms**
- **Example applications**

Particle Simulation

```
t = 0
while t < t_final
    for i = 1 to n              … n = number of particles
        compute f(i) = force on particle i
    for i = 1 to n
        move particle i under force f(i) for time dt    … using F=ma
    compute interesting properties of particles (energy, etc.)
    t = t + dt
end while
```

° f(i) = external_force + nearest_neighbor_force + N-Body_force
  - External_force is usually embarrassingly parallel and costs O(N) for all particles
    - external current in Sharks and Fish
  - Nearest_neighbor_force requires interacting with a few neighbors, so still O(N)
    - van der Waals, bouncing balls
  - N-Body_force (gravity or electrostatics) requires all-to-all interactions
    - $f(i) = \sum_{k \neq i} f(i,k)$ … f(i,k) = force on i from k
    - $f(i,k) = c*v/||v||^3$ in 3 dimensions or $f(i,k) = c*v/||v||^2$ in 2 dimensions
      - v = vector from particle i to particle k , c = product of masses or charges
      - ||v|| = length of v
    - Obvious algorithm costs $O(N^2)$, but we can do better...

Applications

° Astrophysics and Celestial Mechanics
  - Intel Delta = 1992 supercomputer, 512 Intel i860s
  - 17 million particles, 600 time steps, 24 hours elapsed time
    - M. Warren and J. Salmon
    - Gordon Bell Prize at Supercomputing 92
  - Sustained 5.2 Gflops = 44K Flops/particle/time step
  - 1% accuracy
  - Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer

° Plasma Simulation

° Molecular Dynamics

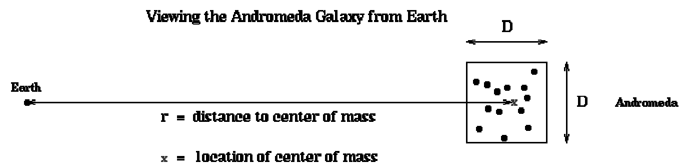
° Electron-Beam Lithography Device Simulation

° Fluid Dynamics (vortex method)

° Good sequential algorithms too!

Reducing the number of particles in the force sum

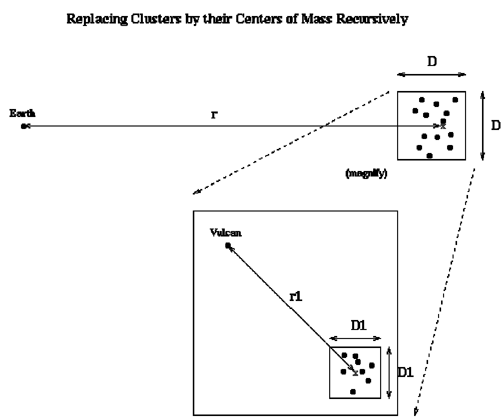
- All later divide and conquer algorithms use same intuition
- Consider computing force on earth due to all celestial bodies
  - Look at night sky, # terms in force sum >= number of visible stars
  - Oops! One "star" is really the Andromeda galaxy, which contains billions of real stars
    - Seems like a lot more work than we thought …
- Don't worry, ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass
  - D = size of box containing Andromeda , r = distance of CM to Earth
  - Require that D/r be "small enough"



  - Idea not new: Newton approximated earth and falling apple by CMs

What is new: Using points at CM recursively

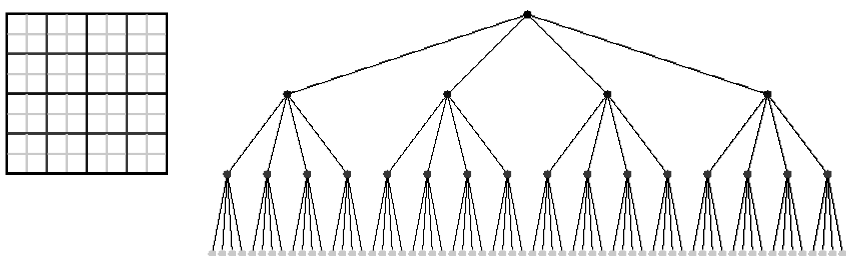
- From Andromeda's point of view, Milky Way is also a point mass
- Within Andromeda, picture repeats itself
  - As long as D1/r1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
  - Boxes nest in boxes recursively



Quad Trees

- Data structure to subdivide the plane
  - Nodes can contain coordinates of center of box, side length
  - Eventually also coordinates of CM, total mass, etc.
- In a complete quad tree, each nonleaf node has 4 children

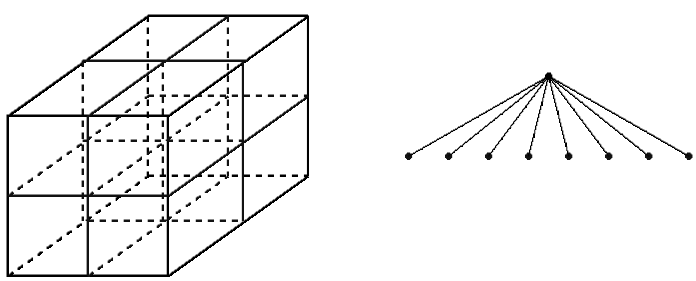
A Complete Quadtree with 4 Levels



Oct Trees

- Similar Data Structure to subdivide space

2 Levels of an Octree



Using Quad Trees and Oct Trees

° All our algorithms begin by constructing a tree to hold all the particles

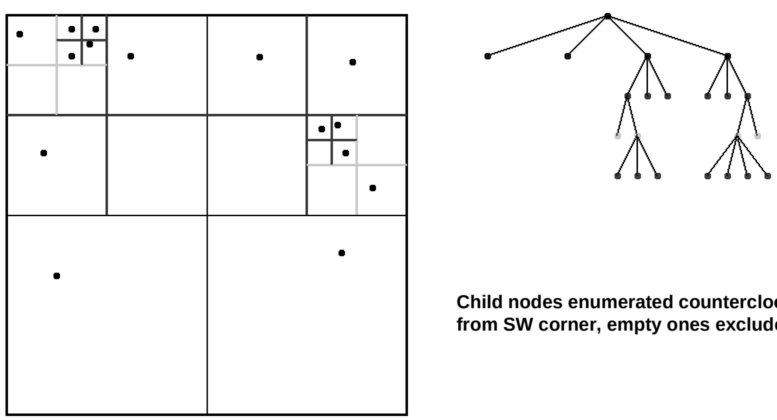
° Interesting cases have nonuniformly distributed particles

- In a complete tree most nodes would be empty, a waste of space and time

° Adaptive Quad (Oct) Tree only subdivides space where particles are located

Example of an Adaptive Quad Tree

Adaptive quadtree where no square contains more than 1 particle



Child nodes enumerated counterclockwise from SW corner, empty ones excluded

Adaptive Quad Tree Algorithm (Oct Tree analogous)

```
Procedure QuadTreeBuild
   QuadTree = {emtpy}
    for j = 1 to N                        … loop over all N particles
        QuadTreeInsert(j, root)        … insert particle j in QuadTree
    endfor
    …   At this point, the QuadTree may have some empty leaves,
    …   whose siblings are not empty
    Traverse the QuadTree eliminating empty leaves  … via, say Breadth First Search
```

```
Procedure QuadTreeInsert(j, n)
    … Try to insert particle j at node n in QuadTree
    … By construction, each leaf will contain either 1 or 0 particles
    if the subtree rooted at n contains more than 1 particle   … n is an internal node
       determine which child c of node n contains particle j
       QuadTreeInsert(j, c)
    else if the subtree rooted at  n contains 1 particle   …  n is a leaf
       add n's 4 children to the QuadTree
       move the particle already in n into the child containing it
       let c be the child of n containing j
       QuadTreeInsert(j, c)
    else  … the subtree rooted at n is an empty leaf
       store particle j in node n
    end
```

- Cost <= N * maximum cost of QuadTreeInsert = O(N * maximum depth of QuadTree)
- Uniform distribution => depth of QuadTree = O(log N), so Cost = O(N log N)
- Arbitrary distribution => depth of Quad Tree = O(b) = O(# bits in particle coords), so Cost = O(bN)

Barnes-Hut Algorithm

- **"A Hierarchical O(n log n) force calculation algorithm", J. Barnes and P. Hut, Nature, v. 324 (1986), many later papers**
- **Good for low accuracy calculations:**

  RMS error $= (\Sigma_k \|\text{approx } f(k) - \text{true } f(k)\|^2 / \|\text{true } f(k)\|^2 / N)^{1/2}$

  $\sim$ 1%

  (other measures better if some true f(k) ~ 0)
- **High Level Algorithm (in 2D, for simplicity)**

1) Build the QuadTree using QuadTreeBuild
   … already described, cost = O(N log N) or O(b N)
2) For each node = subsquare in the QuadTree, compute the CM and total mass (TM) of all the particles it contains
   … "post order traversal" of QuadTree, cost = O(N log N) or O(b N)
3) For each particle, traverse the QuadTree to compute the force on it, using the CM and TM of "distant" subsquares
   … core of algorithm
   … cost depends on accuracy desired but still O(N log N) or O(bN)

Step 2 of BH: compute CM and total mass of each node

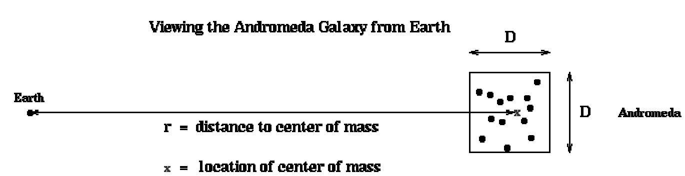
```
… Compute the CM = Center of Mass and TM = Total Mass of all the particles
… in each node of the QuadTree
( TM, CM ) = Compute_Mass( root )

function ( TM, CM ) = Compute_Mass( n )    … compute the CM and TM of node n
    if n contains 1 particle
        … the TM and CM are identical to the particle's mass and location
        store (TM, CM) at n
        return (TM, CM)
    else      … "post order traversal": process parent after all children
        for all children c(j) of n  … j = 1,2,3,4
            ( TM(j), CM(j) ) = Compute_Mass( c(j) )
        endfor
        TM = TM(1) + TM(2) + TM(3) + TM(4)
             … the total mass is the sum of the children's masses
        CM = ( TM(1)*CM(1) + TM(2)*CM(2) + TM(3)*CM(3) + TM(4)*CM(4) ) / TM
             … the CM is the mass-weighted sum of the children's centers of mass
        store ( TM, CM ) at n
        return ( TM, CM )
    end if
```

Cost = O(# nodes in QuadTree)

= O(N log N) or O(b N)

Step 3 of BH: compute force on each particle

- For each node = square, can approximate force on particles outside the node due to particles inside node by using the node's CM and TM
- This will be accurate enough if the node if "far away enough" from the particle
- For each particle, use as few nodes as possible to compute force, subject to accuracy constraint
- Need criterion to decide if a node is far enough from a particle
  - D = side length of node
  - r = distance from particle to CM of node
  - θ = user supplied error tolerance < 1
  - Use CM and TM to approximate force of node on box if D/r < θ



Computing force on a particle due to a node

- Suppose node n, with CM and TM, and particle k, satisfy D/r < θ
- Let (x_k, y_k, z_k) be coordinates of k, m its mass
- Let (x_{CM}, y_{CM}, z_{CM}) be coordinates of CM
- $r = ((x_k - x_{CM})^2 + (y_k - y_{CM})^2 + (z_k - z_{CM})^2)^{1/2}$
- G = gravitational constant
- Force on k ~ $G * m * TM * \left[\frac{x_{CM} - x_k}{r^3}, \frac{y_{CM} - y_k}{r^3}, \frac{z_{CM} - z_k}{r^3} \right]$

Details of Step 3 of BH

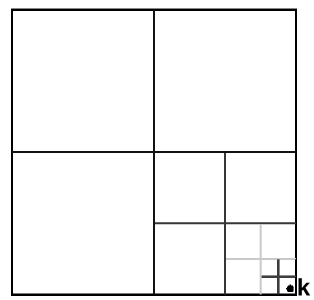
```
… for each particle, traverse the QuadTree to compute the force on it
for k = 1 to N
   f(k) = TreeForce( k, root )
            … compute force on particle k due to all particles inside root
endfor
```

```
function f = TreeForce( k, n )
   … compute force on particle k due to all particles inside node n
   f = 0
   if n contains one particle  … evaluate directly
      f = force computed using formula on last slide
   else
      r = distance from particle k to CM of particles in n
      D = size of n
      if  D/r  < θ    … ok to approximate by CM and TM
          compute f using formula from last slide
      else            … need to look inside node
          for all children c of n
               f = f + TreeForce ( k, c )
          end for
      end if
   end if
```

Analysis of Step 3 of BH

- **Correctness follows from recursive accumulation of force from each subtree**
  - **Each particle is accounted for exactly once, whether it is in a leaf or other node**
- **Complexity analysis**
  - **Cost of TreeForce(k, root) = O(depth in QuadTree of leaf containing k)**
  - **Proof by Example (for θ>1):**
    - **For each undivided node = square, (except one containing k), D/r < 1 < θ**
    - **There are 3 nodes at each level of the QuadTree**
    - **There is O(1) work per node**
    - **Cost = O(level of k)**
  - **Total cost = O(Σ_k level of k) = O(N log N)**
    - **Strongly depends on θ**

Sample Barnes-Hut Force calculation
For particle in lower right corner
Assuming theta > 1



Fast Multiple Method (FMM)

- **"A fast algorithm for particle simulation", L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, many later papers**
- **Greengard won 1987 ACM Dissertation Award**
- **Differences from Barnes-Hut**
  - **FMM computes the *potential* at every point, not just the force**
  - **FMM uses more information in each box than the CM and TM, so it is both more accurate and more expensive**
  - **In compensation, FMM accesses a fixed set of boxes at every level, independent of D/r**
  - **BH uses fixed information (CM and TM) in every box, but # boxes increases with accuracy. FMM uses a fixed # boxes, but the amount of information per box increase with accuracy.**
- **FMM uses two kinds of expansions**
  - **Outer expansions represent potential outside node due to particles inside, analogous to (CM,TM)**
  - **Inner expansions represent potential inside node due to particles outside; *Computing this for every leaf node is the computational goal of FMM***
- **First review potential, then return to FMM**

Gravitational/Electrostatic Potential

- Force on particle at (x,y,z) due to one at origin = $-(x,y,z)/r^3$
- Instead of force, consider potential $\phi(x,y,z) = -1/r$
  - potential satisfies 3D Poisson equation

    $d^2\phi/dx^2 + d^2\phi/dy^2 + d^2\phi/dz^2 = 0$
- Force = -grad $\phi(x,y,z) = -(d\phi/dx , d\phi/dy , d\phi/dz)$
- FMM will compute a compact express for $\phi(x,y,z)$, which can be evaluated and/or differentiated at any point
- For simplicity, present algorithm in 2D instead of 3D
  - force = $-(x,y)/r^2 = -z / |z|^2$ where $z = x + \iota y$ (complex number)
  - potential = $\log |z|$
  - potential satisfies 2D Poisson equation

    $d^2\phi/dx^2 + d^2\phi/dy^2 = 0$
  - equivalent to gravity between "infinite parallel wires" instead of point masses

2D Multipole Expansion (Taylor expansion in 1/z)

$\phi(z)$ = potential due to $z_{k,\ k=1,\ldots,n}$

$= \Sigma_k\ m_k * \log |z - z_k|$

$= \mathrm{Real}(\ \Sigma_k\ m_k * \log (z - z_k)\ )$

… since $\log z = \log |z|e^{i\theta} = \log |z| + i\theta$

… drop Real() from now on

$= \Sigma_k\ m_k * [\ \log(z) + \log (1 - z_k/z)\ ]$

… how logarithms work

$= M * \log(z) + \Sigma_k\ m_k * \log (1 - z_k/z)$

… where $M = \Sigma_k\ m_k$

$= M * \log(z) + \Sigma_k\ m_k * \Sigma_{\ e>=1}\ (z_k/z)^e$

… Taylor expansion converges if $|z_k/z| < 1$

$= M * \log(z) + \Sigma_{\ e>=1}\ z^{-e}\ \Sigma_k\ m_k\ z_k^e$

… swap order of summation

$= M * \log(z) + \Sigma_{\ e>=1}\ z^{-e}\ \alpha_e$

… where $\alpha_e = \Sigma_k\ m_k\ z_k^e$

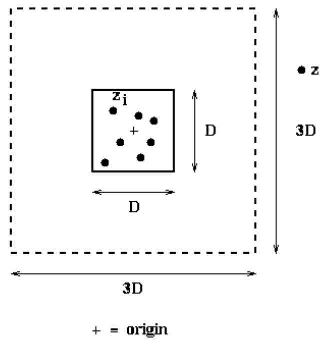
$= M * \log(z) + \Sigma_{\ r>=e>=1}\ z^{-e}\ \alpha_e + \mathrm{error}(\ r\ )$

… where $\mathrm{error}(\ r\ ) = \Sigma_{\ r<e} z^{-e}\ \alpha_e$

Error in Truncated 2D Multipole Expansion

- error(r) = $\Sigma_{r<e} z^{-e} \alpha_e = \Sigma_{r<e} z^{-e} \Sigma_k m_k z_k^e$
- error(r) = O($\{\max_k |z_k| / |z|\}^{r+1}$)   … bounded by geometric sum
- Suppose 1 > c >= $\max_k |z_k| / |z|$
  - error(r) = O(c^{r+1})
  - Suppose all the particles z_k lie inside a D-by-D square centered at the origin
  - Suppose z is outside a 3D-by-3D square centered at the origin
  - 1/c = 1.5*D/(D/sqrt(2)) ~ 2.12
  - Each extra term in expansion increases accuracy about 1 bit
  - 24 terms enough for single, 53 terms for double
- In 3D, can use spherical harmonics or other expansions

Multipole error outside larger box
is bounded by O((3/sqrt(2))^(-p)) ~ O(2.12^(-p))



Outer(n) and Outer Expansion

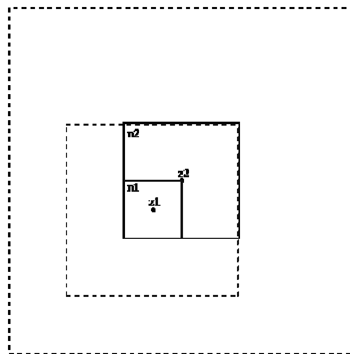
$\phi(z) \sim M * \log(z) + \Sigma_{r>=e>=1} z^{-e} \alpha_e$

- Outer(n) = (M, α_1 , α_2 , … , α_r)
  - Stores data for evaluating potential $\phi(z)$ outside node n due to particles inside n
  - Used in place of (TM,CM) in Barnes-Hut
- Will be computed for each node in QuadTree
- Cost of evaluating $\phi(z)$ is O(r), independent of the number of particles inside n

Outer_shift: converting Outer(n_1) to Outer(n_2)

- As in step 2 of BH, we want to compute Outer(n) cheaply from Outer(c) for all children of n
- How to combine outer expansions around different points?
  - $\phi_k(z) \sim M_k * \log(z-z_k) + \Sigma_{r>=e>=1} (z-z_k)^{-e} \alpha_{ek}$ expands around z_k , k=1,2
  - First step: make them expansions around same point
- n_1 is a subsquare of n_2
- z_k = center(n_k) for k=1,2
- Outer(n_1) expansion accurate outside blue dashed square, so also accurate outside black dashed square
- So there is an Outer(n_2) expansion with different α_k and center z_2 which represents the same potential as Outer(n_1) outside the black dashed box

Using Outer_Shift to convert Outer(n1) to Outer(n2)



CS267 L21 N-Body Problem I.23

Demmel Sp 1999

Outer_shift: continued

- Given

$$\phi_1(z) = M_1 * \log(z-z_1) + \Sigma_{r>=e>=1} (z-z_1)^{-e} \alpha_{e1}$$

- Solve for M_2 and α_{e2} in

$$\phi_1(z) \sim \phi_2(z) = M_2 * \log(z-z_2) + \Sigma_{r>=e>=1} (z-z_1)^{-e} \alpha_{e2}$$

- Get $M_2 = M_1$ and each α_{e2} is a linear combination of the α_{e1}
  - multiply r-vector of α_{e1} values by a fixed r-by-r matrix to get α_{e2}
- (M_2 , α_{12} , … , α_{r2}) = Outer_shift(Outer(n_1) , z_2)
  = desired Outer(n_2)

CS267 L21 N-Body Problem I.24

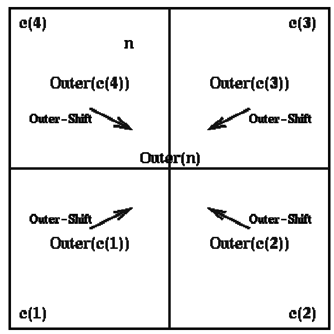
Demmel Sp 1999

FMM: compute Outer(n) for each node n in QuadTree

… Compute the Outer(n) for each node of the QuadTree
outer = Build_ Outer(root)

```
function ( M, α1,…,αr ) = Build_ Outer( n )    … compute outer expansion of node n
    if n if a leaf  … it contains 1 (or a few) particles
        compute and return Outer(n) = ( M, α1,…,αr ) directly from its definition as a sum
    else      … "post order traversal": process parent after all children
        Outer(n) = 0
        for all children c(k) of n  … k = 1,2,3,4
            Outer( c(k) ) = Build_ Outer( c(k) )
            Outer(n) = Outer(n) +
                Outer_shift( Outer(c(k)) , center(n))
                … just add component by component
        endfor
        return Outer(n)
end if
```

Inner Loop of Build_Outer



Cost = O(# nodes in QuadTree)
= O(N log N) or O(b N)
same as for Barnes-Hut

Inner(n) and Inner Expansion

- Outer(n) used to evaluate potential outside n due to particles inside
- Inner(n) will be used to evaluate potential inside n due to particles outside
  - $\sum_{0<=k<=r} \beta_k * (z-z_c)^k$
  - z_c = center of n, a D-by-D box
  - Inner(n) = (β_0 , β_1 , … , β_r , z_c)
  - Particles outside lie outside 3D-by-3D box centered at z_c
- Output of FMM will include Inner(n) for each leaf node n in QuadTree
- Need to show how to convert Outer to Inner expansions

Summary of FMM

(1) Build the QuadTree
(2) Call Build_Outer(root), to compute outer expansions of each node n in the QuadTree
(3) Traverse the QuadTree from top to bottom, computing Inner(n) for each node n in QuadTree
    … still need to show how to convert outer to inner expansions
(4) For each leaf node n, add contributions of nearest particles directly into Inner(n)
    … since Inner(n) only includes potential from distant nodes