CS601: Software Development for Scientific Computing

Autumn 2024

Week15 : N-Body Problems

Course Progress..

- **Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)**

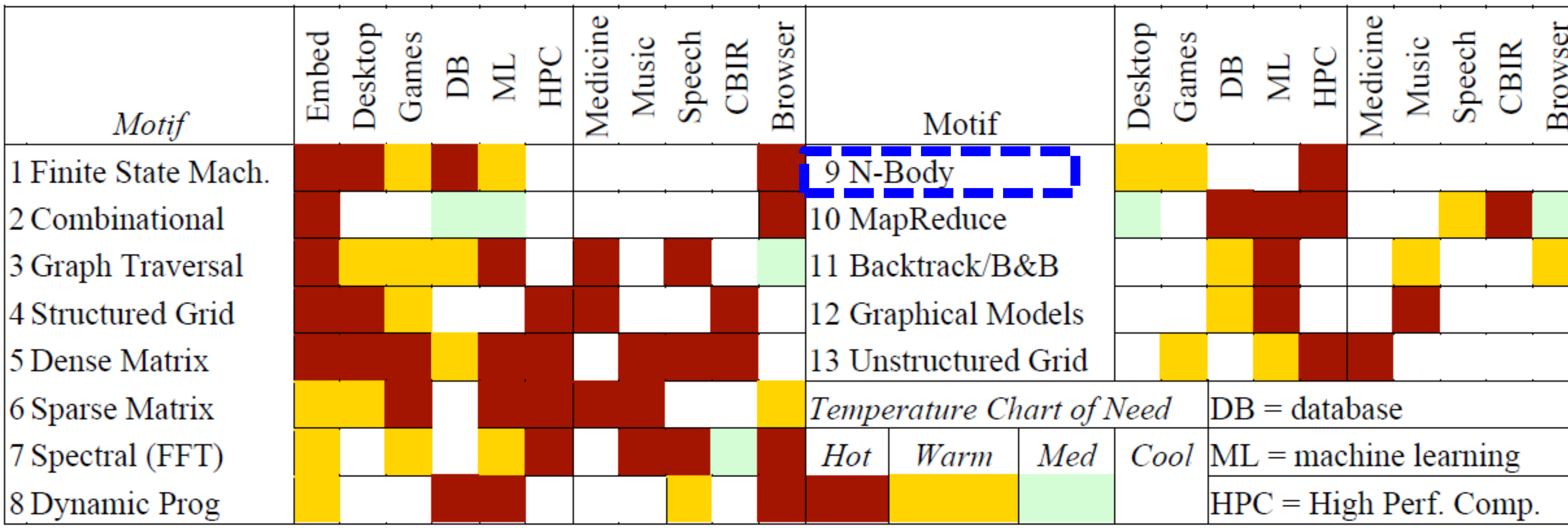


Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

Next..

Particle (Simulation) Methods

- N-Body Simulation – Problem

System of N-bodies (e.g. galaxies, stars, atoms, light rays etc.) interacting with each other continuously

  - Problem:
    - Compute force acting on a body due to all other bodies in the system
    - Determine position, velocity, at various times for each body
  - Objective:
    - Determine the (approximate) evolution of a system of bodies interacting with each other simultaneously

Particle (Simulation) Methods

- N-Body Simulation - Examples
  - Astrophysical simulation: E.g. each body is a star/galaxy https://commons.wikimedia.org/w/index.php?title=File%3AGalaxy_collision.ogv
  - Graphics: E.g. each body is a ray of light emanating from the light source. https://www.fxguide.com/fxfeatured/brave-new-hair/
    - Here each body is a point on a strand of hair

N-Body Simulation

- All-pairs Method
  - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
  - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
  - Examples:
    - Barnes-Hut
    - Fast Multipole Method

N-Body Simulation

- Three fundamental simulation approaches
  - Particle-Particle (PP)
  - Particle-Mesh (PM)
  - Particle-Particle-Particle-Mesh (P3M)
- Hybrid approaches
  - Nested Grid Particle Scheme
  - Tree Codes
  - Tree Code Particle Mesh (TPM)
- Self Consistent Field (SCF), Smoothed-Particle Hydrodynamics (SPH), Symplectic etc.

Particle-Particle method

- Simplest. Adopts an all-pairs approach.
- State of the system at time `t` given by particle positions $x_i(t)$ and velocity $v_i(t)$ for `i=1` to `N`

$$\{x_i(t), v_i(t); i = 1, N\}$$

  – Steps:

    1. Compute forces
    2. Integrate equations of motion
    3. Update time counter
       Each iteration updates $x_i(t)$ and $v_i(t)$ to compute $x_i(t + \Delta t)$ and $v_i(t + \Delta t)$

Particle-Particle Method

1. Compute forces

```
//initialize forces
for i=1 to N
  F_i = 0

//Accumulate forces
for i=1 to N-1
  for j=i+1 to N
      F_i = F_i + F_ij
      F_j = F_j - F_ij
```

F_{ij} is the force on particle `i` due to particle `j`

Typically: $F_i = F_{external} + F_{nearest\_neighbor} + F_{N\text{-}Body}$

Particle-Particle Method

2. Integrate equations of motion

```
for i=1 to N
```

$$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\,\Delta t$$ `//using a=F/m and v=u+at`

$$x_i^{new} = x_i^{old} + v_i\,\Delta t$$

3. Update time counter

$$t^{new} = t^{old} + \Delta t$$

Particle-Particle Method

```
t=0
while(t<t^final) {
//initialize forces
      for i=1 to N
        F_i = 0
//Accumulate forces
      for i=1 to N-1
        for j=i+1 to N
          F[i] = F[i] + F_ij
          F[j] = F[j] - F_ij
//Integrate equations of motion
      for i=1 to N
```

$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\Delta t$ //using a=F/m and v=u+at

$x_i^{new} = x_i^{old} + v_i\,\Delta t$

// Update time counter

t = t + Δt

}

Particle-Particle Method

- Costs (CPU operations)?

```
t=0
while(t<t^final) {
//initialize forces
        for i=1 to N
          F_i = 0
//Accumulate forces
        for i=1 to N-1
          for j=i+1 to N
            F[i] = F[i] + F_ij
            F[j] = F[j] - F_ij
//Integrate equations of motion
        for i=1 to N
```

$v_i^{new} = v_i^{old} + \frac{F_i}{m_i}\,\Delta t$ //using a=F/m and v=u+at

$x_i^{new} = x_i^{old} + v_i\,\Delta t$

// Update time counter

t = t + Δt

}

Particle-Particle Method

- Experimental results (then):
  - 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 1992
  - Sustained 5.2 Gigaflops = 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*

Particle-Particle Method

- Experimental results (now):

  Vortex particle simulation of turbulence

  – Cluster of 256 NVIDIA GeForce 8800 GPUs

  – 16.8 million particles

    - T. Hamada, R. Yokota, K. Nitadori. T. Narumi, K. Yasoki et al
    - *Gordon Bell Prize for Price/Performance at Supercomputing 2009*

  – Sustained 20 Teraflops, or $8/Gigaflop

Particle-Particle (PP) Method

- Discussion
  - Simple/trivial to program
  - High computational cost
    - Useful when number of particles are small (few thousands) and
    - We are interested in close-range dynamics when the particles in the range contribute *significantly* to forces
    - Constant time step must be replaced with variable time steps and numerical integration schemes for close-range interactions

N-Body Simulation

- All-pairs Method
  - Naïve approach. Compute *all pair-wise interactions*
- Hierarchical Methods
  - Optimize. Reduce the number of pair-wise force calculations. How? dependence on ‘distant’ particle(s) can be *compressed*
  - Examples:
    - Barnes-Hut
    - Fast Multipole Method

Tree Codes

$$F_i = F_{external} + F_{nearest\_neighbor} + F_{N\text{-}Body}$$

- $F_{external}$ can be computed for each body independently. O(N)
- $F_{nearest\_neighbor}$ involve computations corresponding to few nearest neighbors. O(N)
- $F_{N\text{-}Body}$ require all-to-all computations. Most expensive. O(N^2) if computed using all-pairs approach.

```
for(i = 1 to N)
```

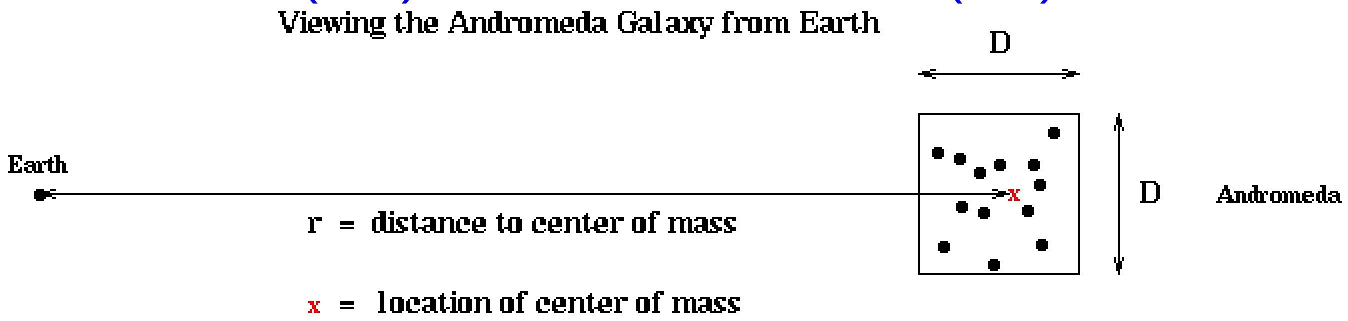
$$F_i = \sum_{i \neq j} F_{ij}$$ F_{ij}= force on i from j

F_{ij} =c*v/||v||^3 in 3D, F_{ij} = c*v/||v||^2 in 2D
v = vector from particle i to particle j , ||v|| = length of v, c = product of masses or charges

We can do better.

Tree Codes: Divide-Conquer Approach

- Consider computing force on earth due to all celestial bodies
  - Look at the night sky. Number of terms in $\sum_{i \neq j} F_{ij}$ is greater than the number of visible stars
  - One "star" could really be the Andromeda galaxy, which contains billions of real stars. *Seems like a lot more work than we thought …*
  - Idea: Ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass (TM)



  - Require that `D/r` be "small enough" (`D` = size of box containing Andromeda , r = distance of CM to Earth).

Idea is not new. Newton approximated earth and falling apple by CM

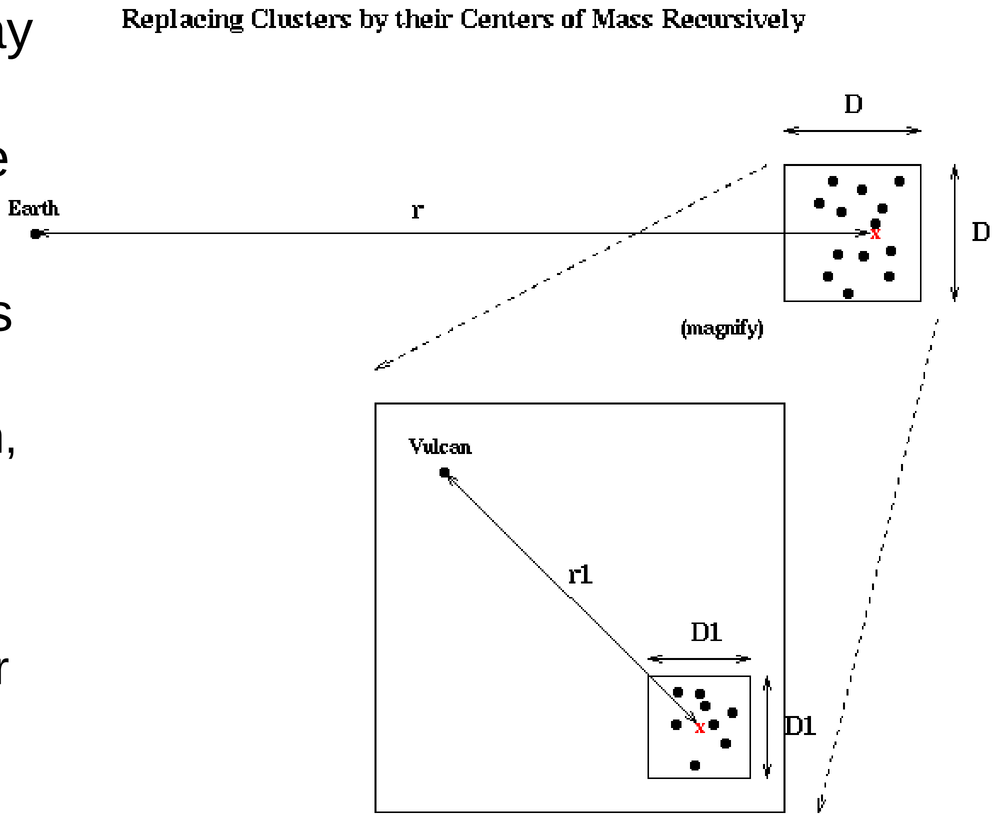
Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Tree Codes: Divide-Conquer Approach

- New idea: recursively divide the box.

– If you are in Andromeda, Milky Way (the galaxy we are part of) could appear like a white dot. So, can be approximated by 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
  - If you are on Vulcan, another solar system in Andromeda can be a white dot.
  - Boxes nest in boxes recursively



Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

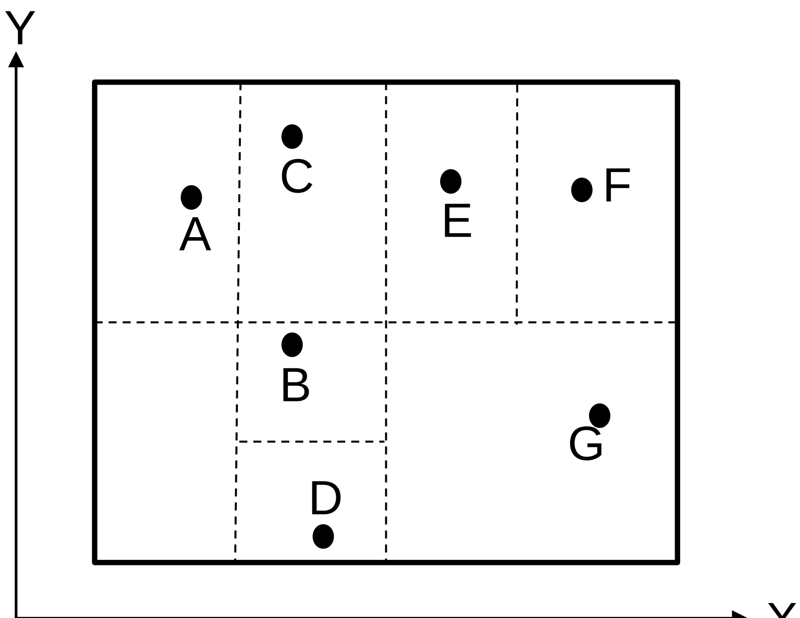
Tree Codes: Divide-Conquer Approach

- Data structures needed:
  - Quad-trees
  - Octrees

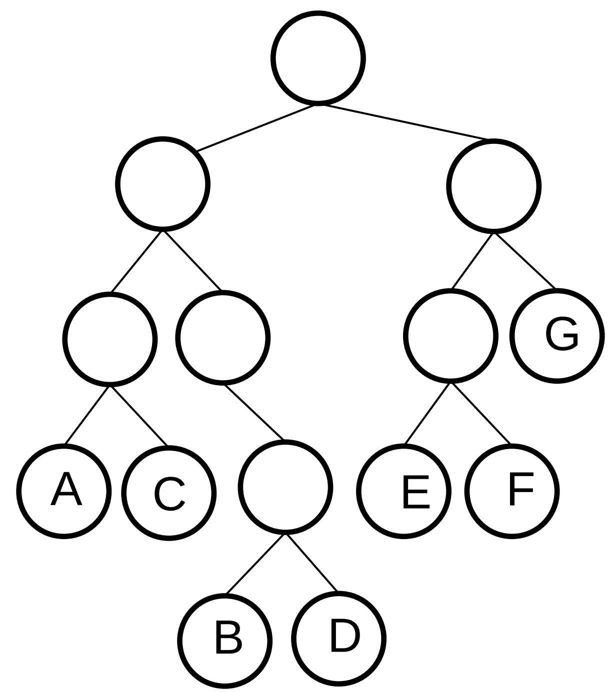
Background – metric trees

e.g. K-dimensional (kd-), Vantage Point (vp-), quad-trees, octrees, ball-trees

2-dimensional space of points



Binary kd-tree, 1 point /leaf cell



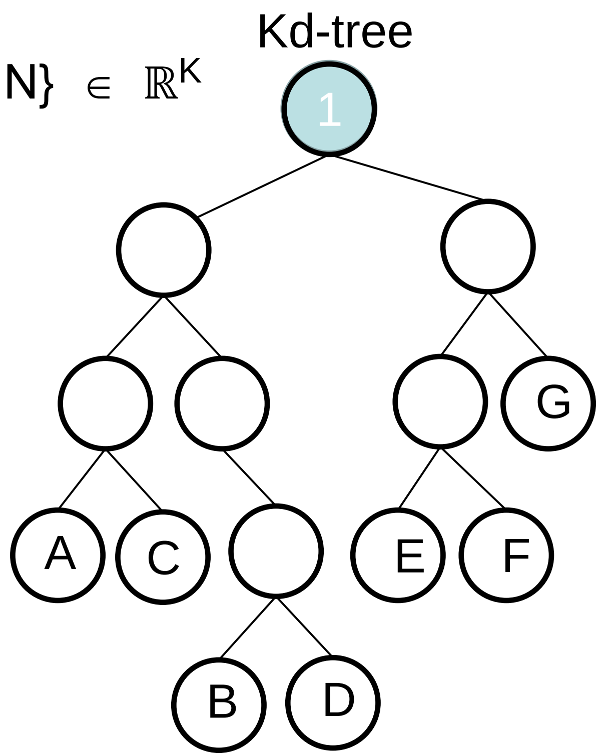
Background - metric trees

Typical use: traverse the tree (often repeatedly), truncate the traversal at some intermediate node if <u>a domain-specific criteria</u> is not met.

E.g. Does the distance from CM to me < D/r?

Input points = {1, 2, … , N} $\in \mathbb{R}^K$

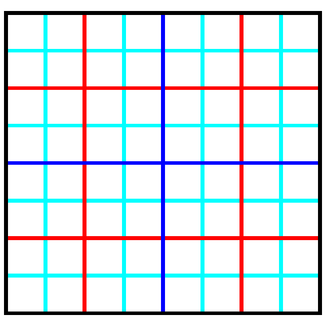
Cost ???

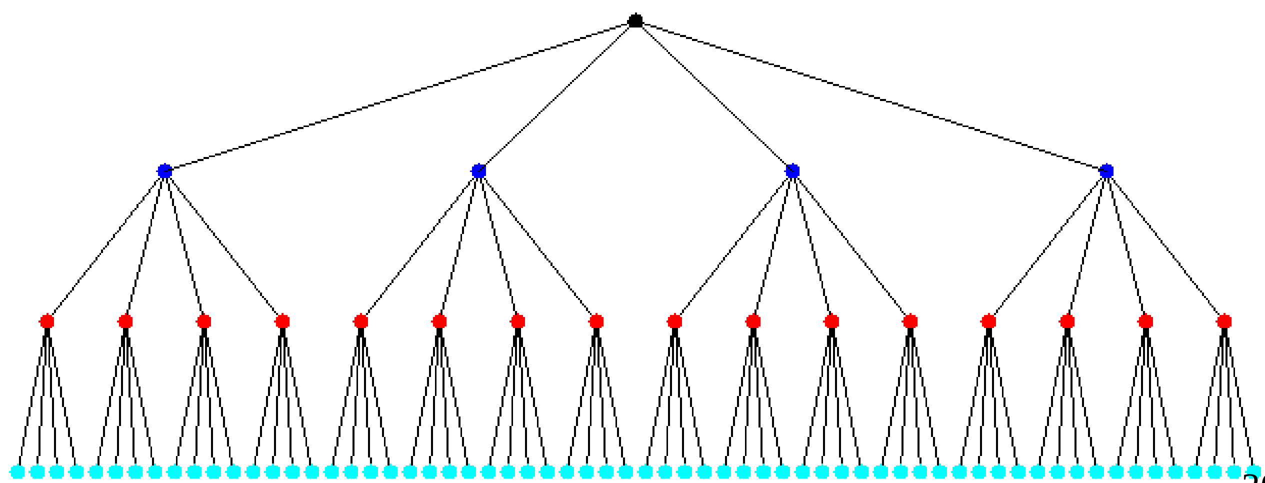


Quad Tree

- 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 non-leaf node has 4 children

A Complete Quadtree with 4 Levels

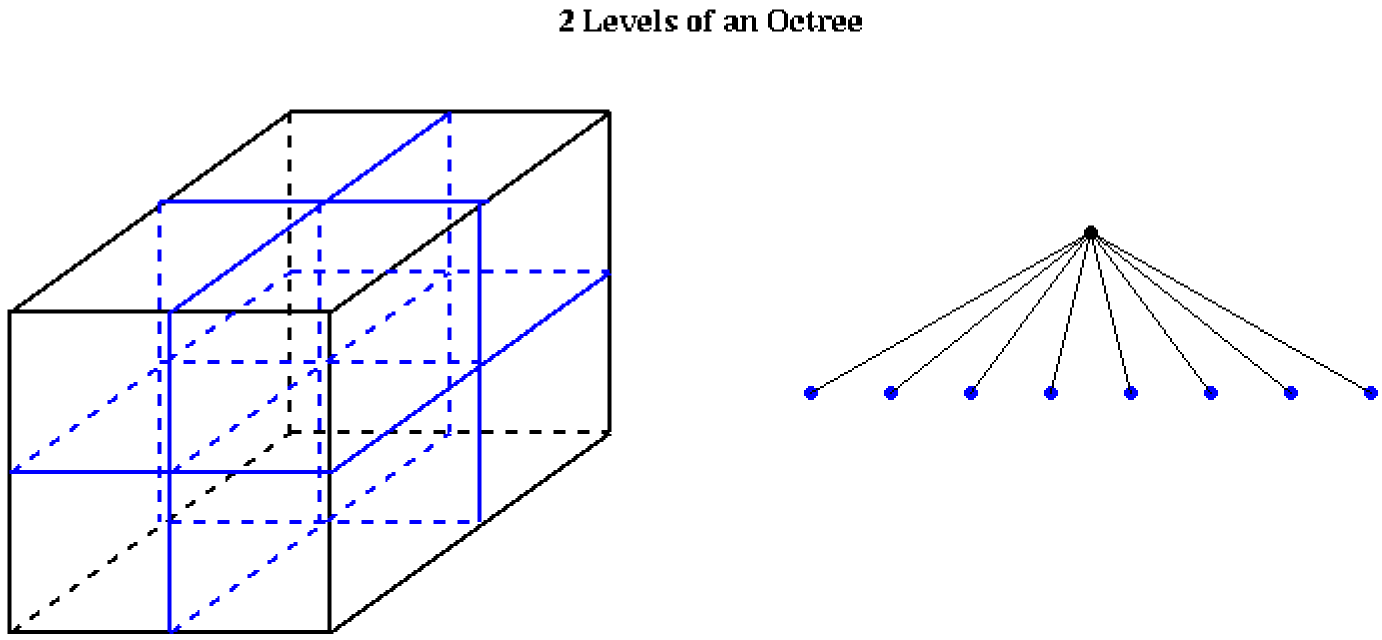




Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Octree or Oct Tree

- Similar data structure for subdividing 3D space



Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

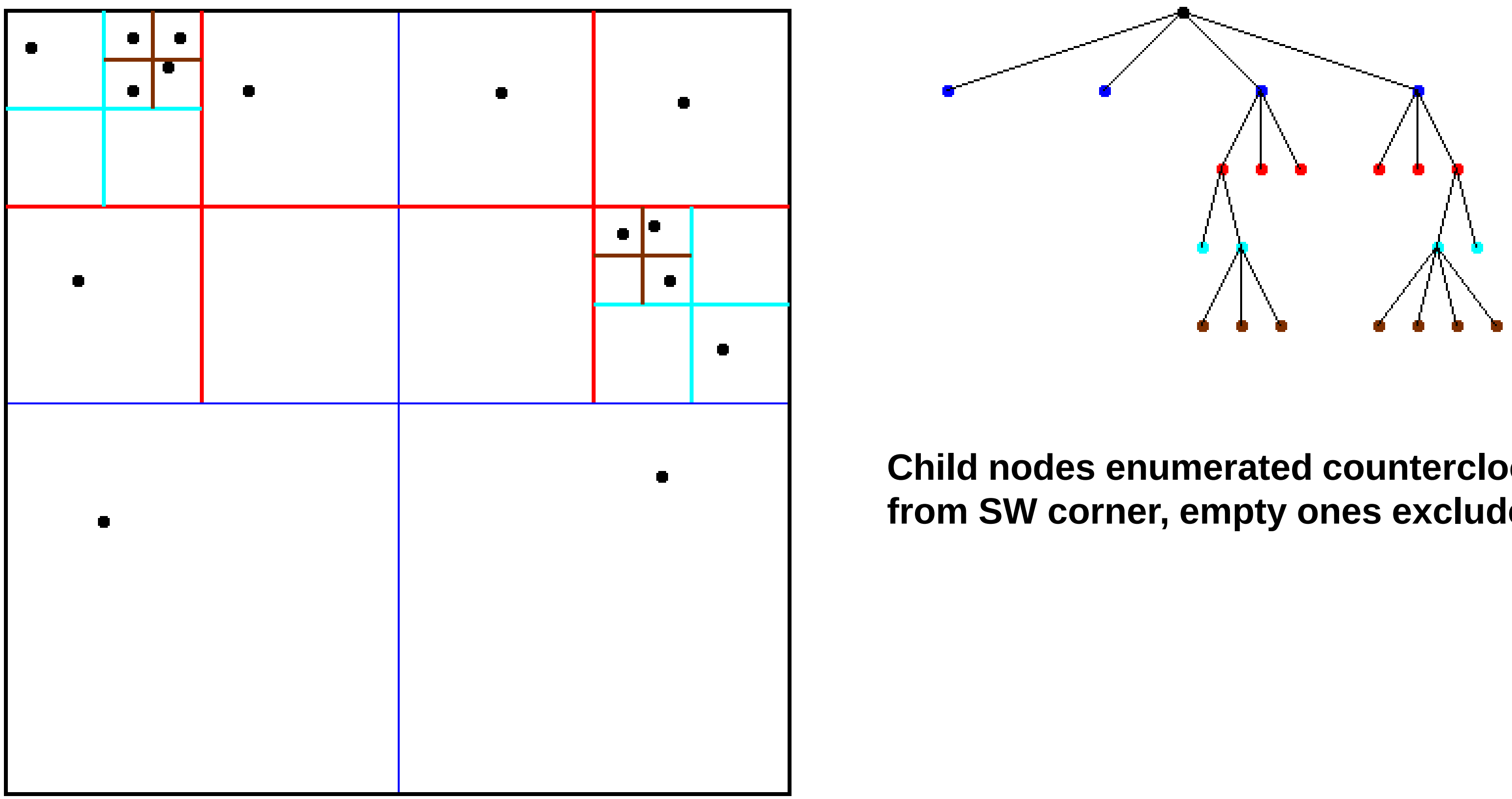
Using Quad Tree and Octree

- 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
- For each particle, traverse the tree to compute force on it

Slide contents based on: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Using Quad Tree and Octree

Adaptive quadtree where no square contains more than 1 particle

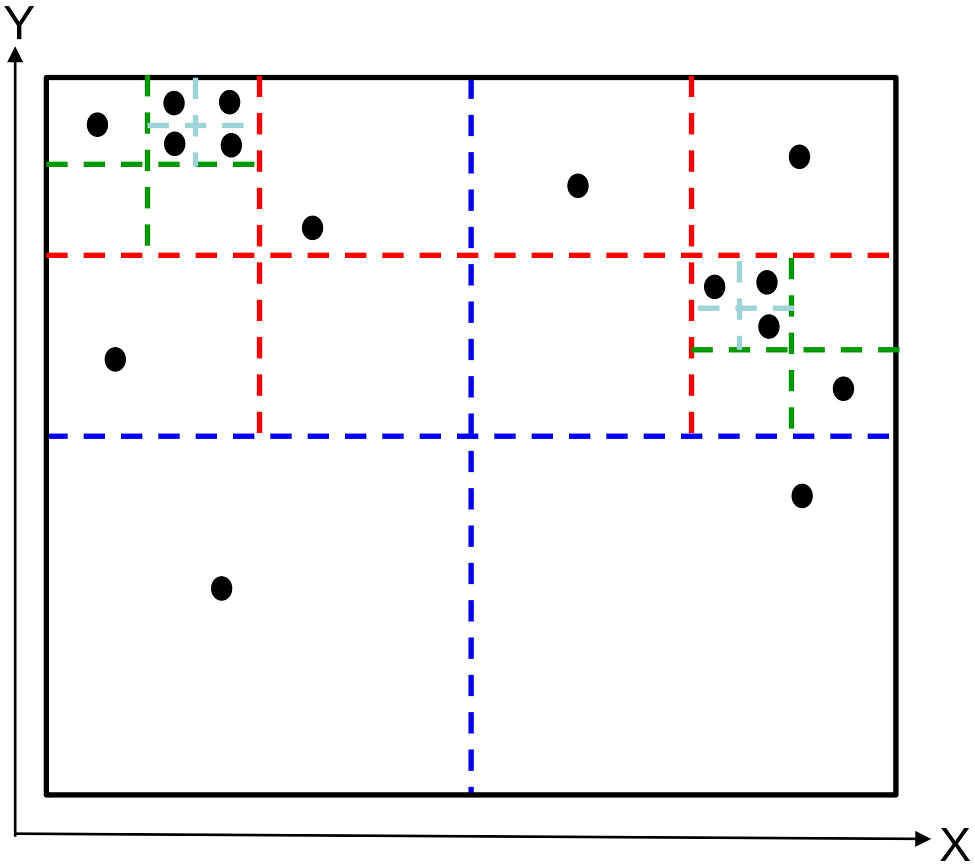


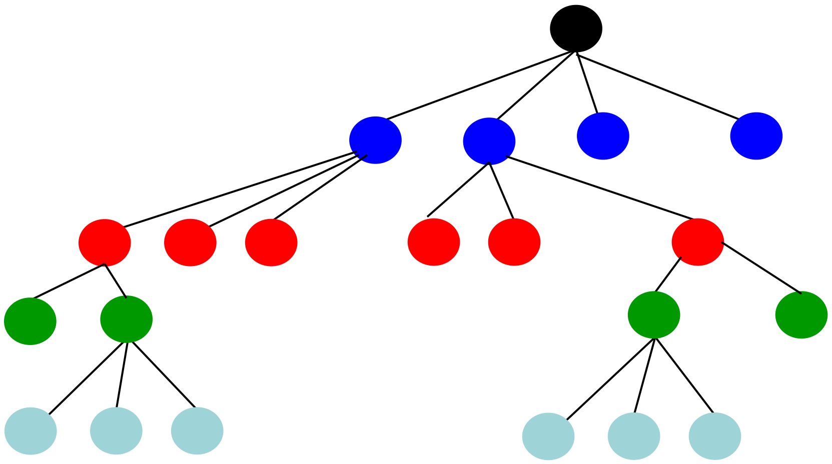
Child nodes enumerated counterclockwise from SW corner, empty ones excluded

- In practice, have q>1 particles/square; tuning parameter

Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Adaptive Quad Tree





- Child nodes numbered as per *Z-order numbering*

Adaptive Quad Tree Construction

```
Procedure Quad_Tree_Build
  Quad_Tree = {emtpy}
  for j = 1 to N                         … loop over all N particles
     Quad_Tree_Insert(j, root)           … insert particle j in QuadTree
  endfor
  …   At this point, each leaf of Quad_Tree will have 0 or 1 particles
  …   There will be 0 particles when some sibling has 1
  Traverse the Quad_Tree eliminating empty leaves  … via, say Breadth First Search

Procedure Quad_Tree_Insert(j, n)  … Try to insert particle j at node n in Quad_Tree
  if n an internal node               … n has 4 children
     - determine which child c of node n contains particle j
     - Quad_Tree_Insert(j, c)
  else if n contains 1 particle   …  n is a leaf
     - add n's 4 children to the Quad_Tree
     - move the particle already in n into the child containing it
     - let c be the child of n containing j
     - Quad_Tree_Insert(j, c)
  else                                …  n empty
     - store particle j in node n
  end
```


Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Adaptive Quad Tree Construction – Cost?

```
Procedure Quad_Tree_Build
   Quad_Tree = {emtpy}                          ≤ N *max cost of Quad_Tree_Insert
   for j = 1 to N                        … loop over all N particles
      Quad_Tree_Insert(j, root)           … insert particle j in QuadTree
   endfor
   …   At this point, each leaf of Quad_Tree will have 0 or 1 particles
   …   There will be 0 particles when some sibling has 1
   Traverse the Quad_Tree eliminating empty leaves  … via, say Breadth First Search
```

```
Procedure Quad_Tree_Insert(j, n) … Try to insert particle j at node n in Quad_Tree
   if n an internal node           … n has 4 children
      - determine which child c of node n contains particle j
      - Quad_Tree_Insert(j, c)
  else if n contains 1 particle    …  n is a leaf
      - add n's 4 children to the Quad_Tree
      - move the particle already in n into the child containing it
      - let c be the child of n containing j
      - Quad_Tree_Insert(j, c)                  ≤ max depth of Quad Tree
   else                            …  n empty
      - store particle j in node n
   end
```


Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Adaptive Quad Tree Construction – Cost?

- Cost of calling Max Depth of Tree:
  - For uniformly distributed points?
  - For arbitrarily distributed points?

- Total Cost = ?

Adaptive Quad Tree Construction – Cost?

- Cost of calling Max Depth of Tree:
  - For uniformly distributed points? = O(log N)
  - For arbitrarily distributed points? = O(b)
    - b is number bits used to represent the coordinates

- Total Cost = $O(\ b\ N)$ or $O(N * \log N)$

Barnes-Hut

- Simplest hierarchical method for N-Body simulation
  - "A Hierarchical O(n log n) force calculation algorithm" by J. Barnes and P. Hut, Nature, v. 324, December 1986
- Widely used in astrophysics
- Accuracy $\geq$ 1% (good when low accuracy is desired/acceptable. Often the case in astrophysics simulations.)

Barnes-Hut: Algorithm

(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
   Center of Mass (CM) and total mass (TM) of all the particles it contains.

3) For each particle, traverse the QuadTree to compute the force on it,

Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Barnes-Hut: Algorithm (step 2)

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

```
(TM, CM)  = Compute_Mass( n )    //compute the CM and TM of node n
   if n contains 1 particle
        //TM and CM are identical to the particle's mass and location
        store (TM, CM) at n
        return (TM, CM)
  else
      for each child 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
```


Slide based on : CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Barnes-Hut: Algorithm (step 2 cost)

(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
   Center of Mass (CM) and total mass (TM) of all the particles it contains.
   … cost = O(number of nodes in the tree) = O(N log N) or O(b N)

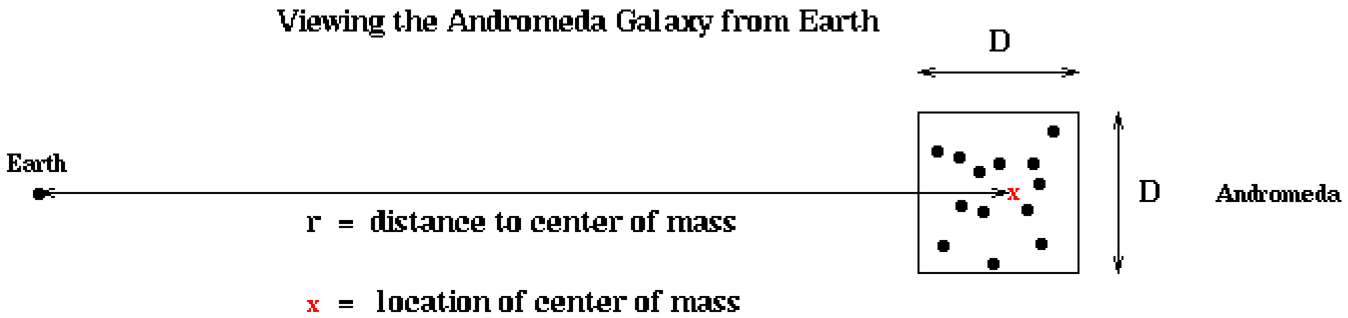
3) For each particle, traverse the QuadTree to compute the force on it,

Slide courtesy: CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Barnes-Hut: Algorithm (step 3)

Goal: Compute the force on each particle by traversing the tree. For each particle, use as few nodes as possible to compute force, subject to accuracy constraint.

- 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 is "far away enough" from the particle
- 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**



Slide based on : CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Barnes-Hut: Algorithm (step 3)

```
//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 (except k)
endfor
function f = TreeForce( k, n )
   //compute force on particle k due to all particles inside node n (except k)
   f = 0
   if n contains one particle (not k)  //evaluate directly
       return f = force computed using direct formula
   else
       r = distance from particle k to CM of particles in n
       D = side length of n
       if  D/r  < q    //ok to approximate by CM and TM
            return f = computed approximately using CM and TM
       else                //need to look inside node
            for each child c(j) of n //j=1,2,3,4
                  f = f + TreeForce ( k, c(j) )
            end for
            return f
       end if
   end if
```

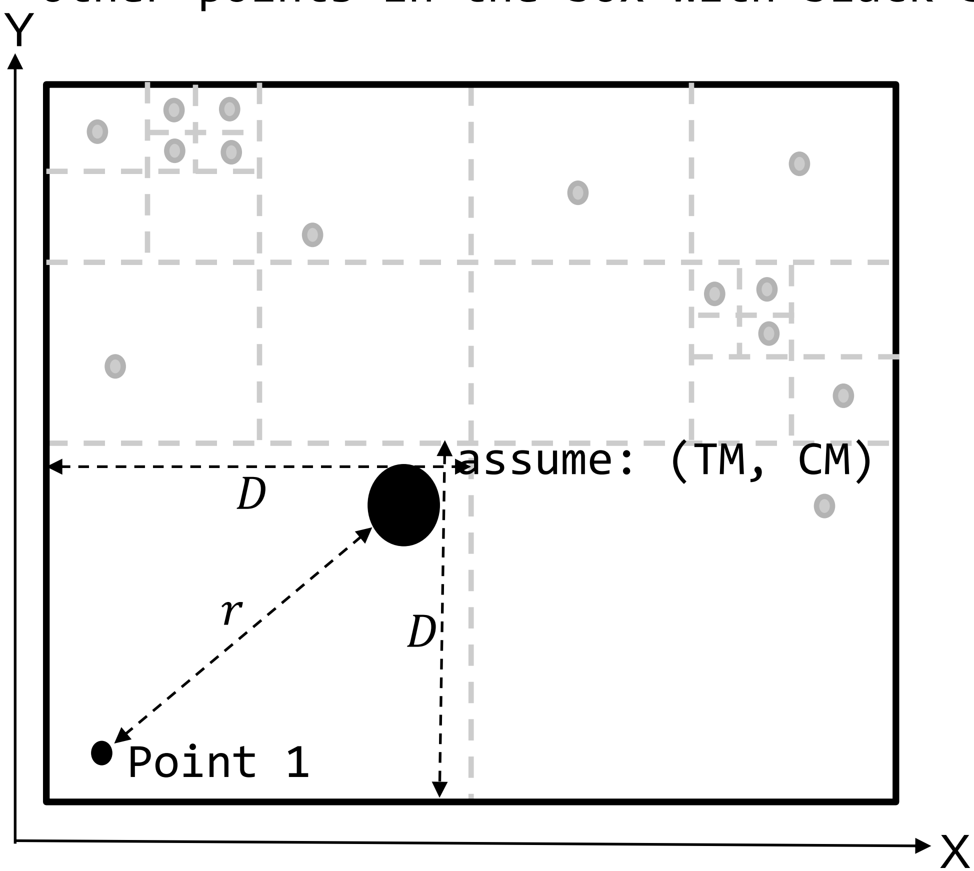

Slide based on : CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

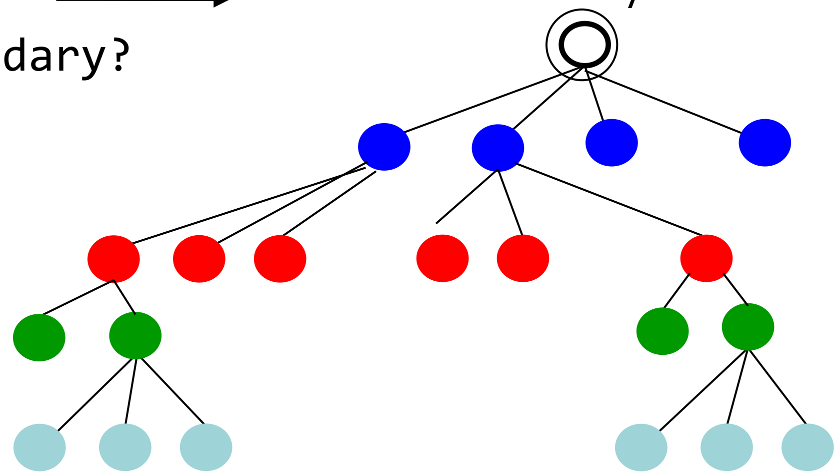
Barnes-Hut: step 3 example

- Example: Assume $\theta \geq 1$. In practice $\theta < 1$.

What is the force on Point 1 due to all other points in the box with black-boundary?

Point 1: is $D/r < \theta$?



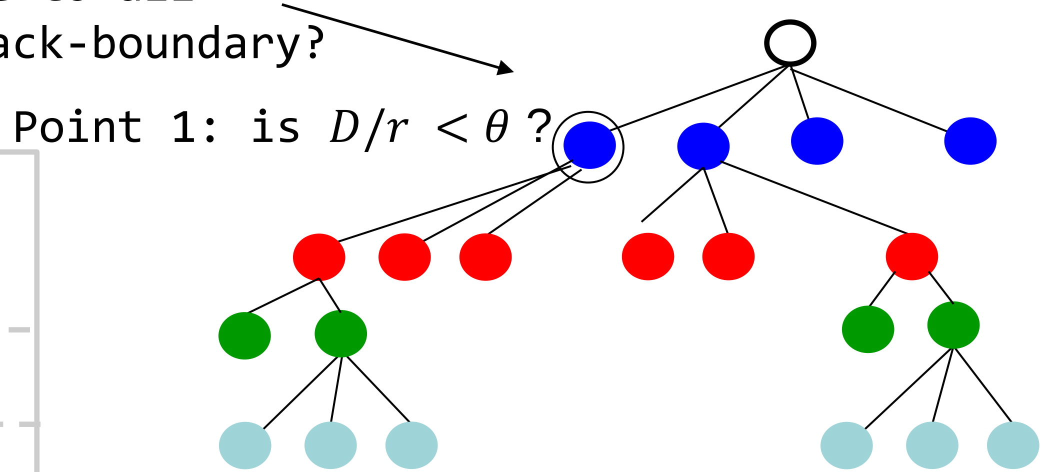


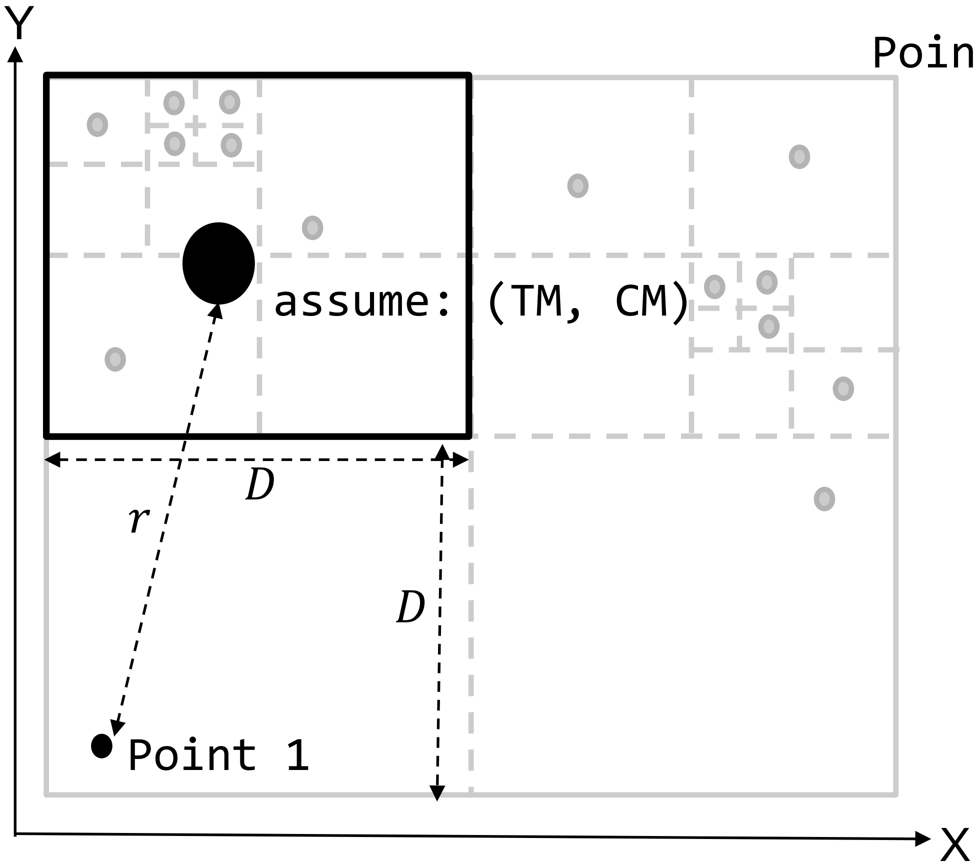
No. Compute force due to each child of the root node (i.e. particles in each quadrant of the square). Start with child 1: c(1).

Barnes-Hut: step 3 example

- Example: Assume $\theta \geq 1$. In practice $\theta < 1$.

What is the force on Point 1 due to all other points in the box with black-boundary?



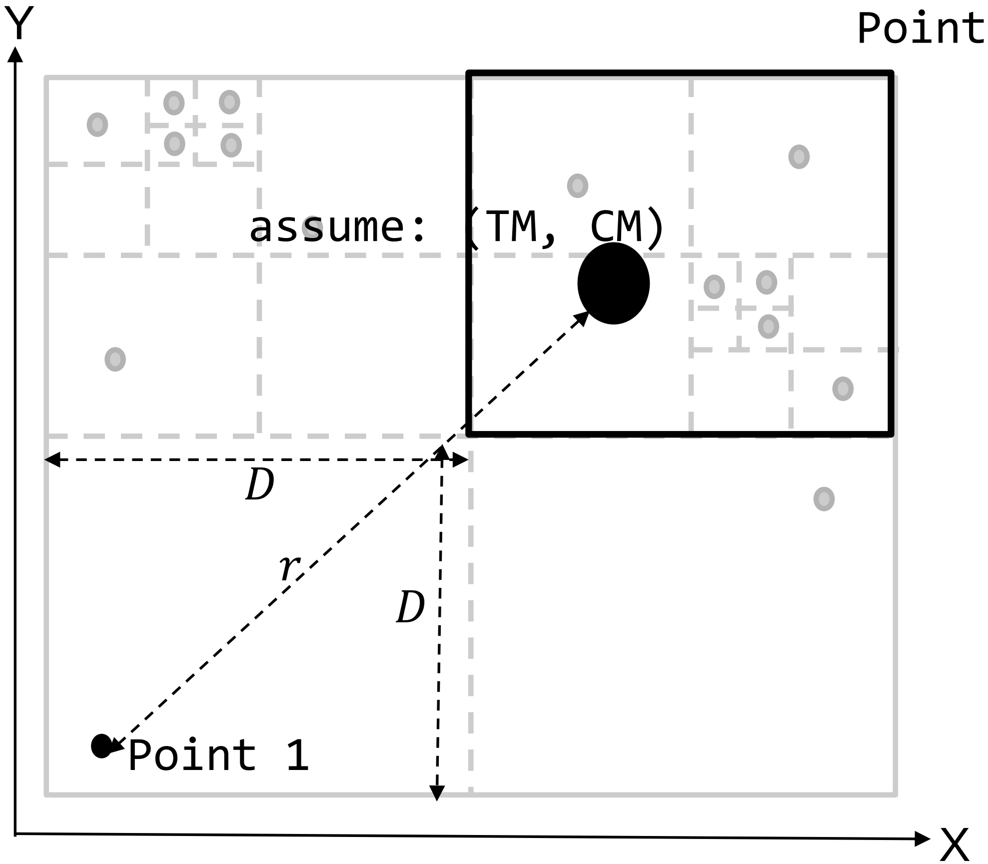


Yes. Approximate force due to each particle contained in the black-boundary box by the TM and CM of the box.

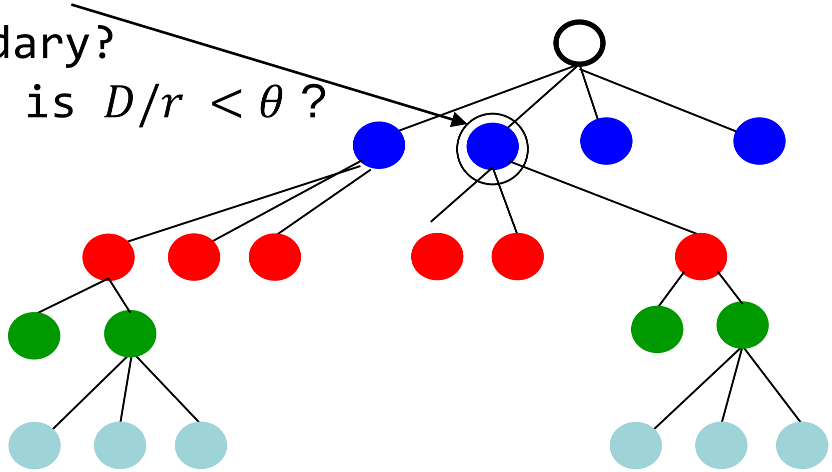
Barnes-Hut: step 3 example

- Example: Assume $\theta \geq 1$. In practice $\theta < 1$.

What is the force on Point 1 due to all other points in the box with black-boundary?



Point 1: is $D/r < \theta$?



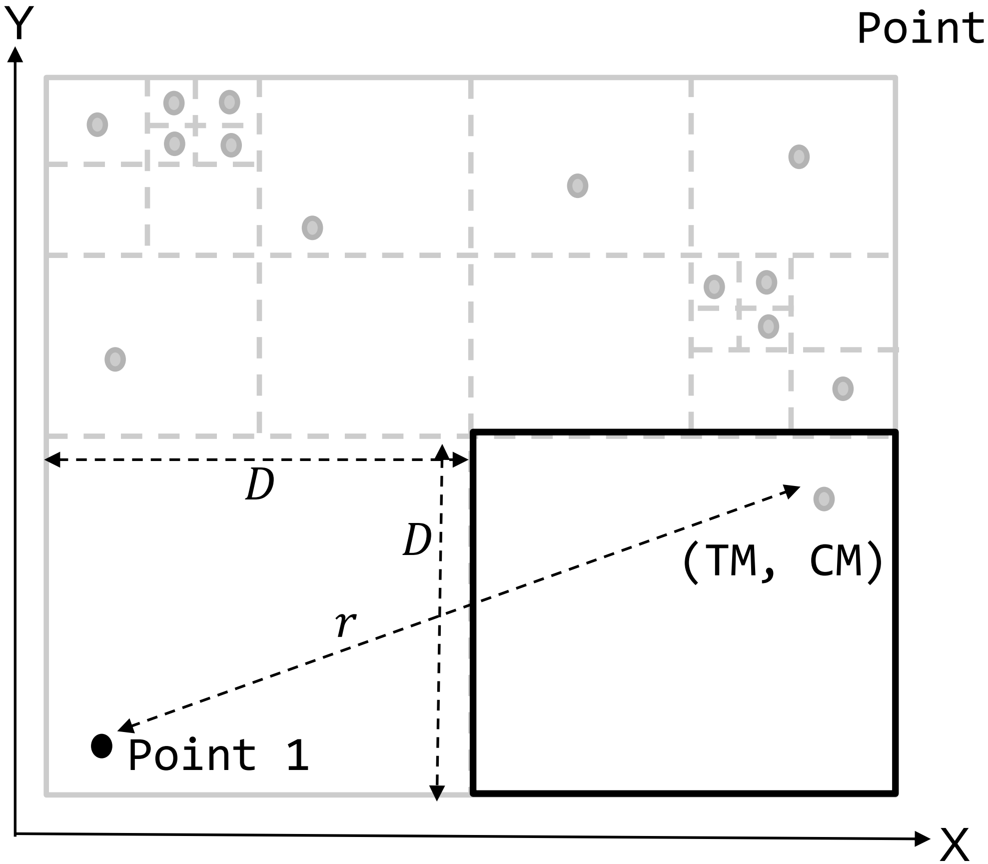
Yes. Approximate force due to each particle contained in the black-boundary box by the TM and CM of the box.

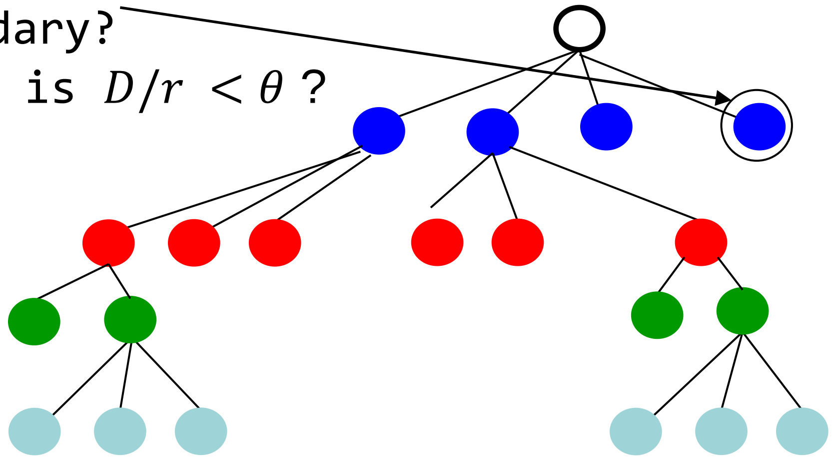
Barnes-Hut: step 3 example

- Example: Assume $\theta \geq 1$. In practice $\theta < 1$.

What is the force on Point 1 due to all other points in the box with black-boundary?

Point 1: is $D/r < \theta$?





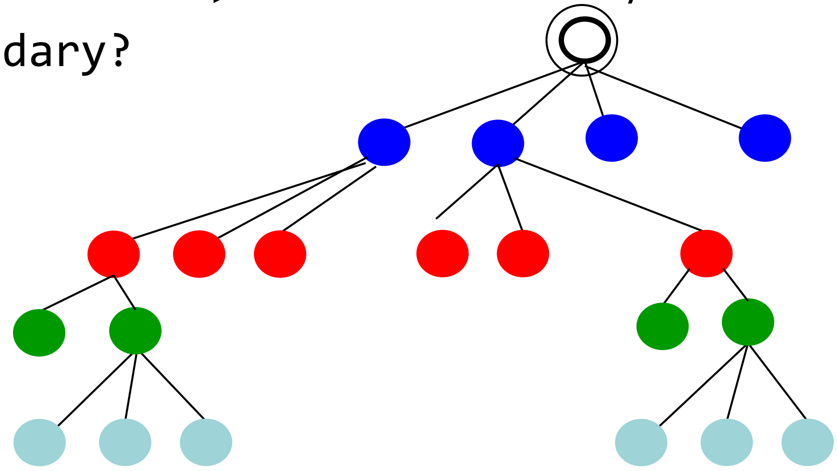
Contains 1 particle / leaf node. Compute force using direct formula.

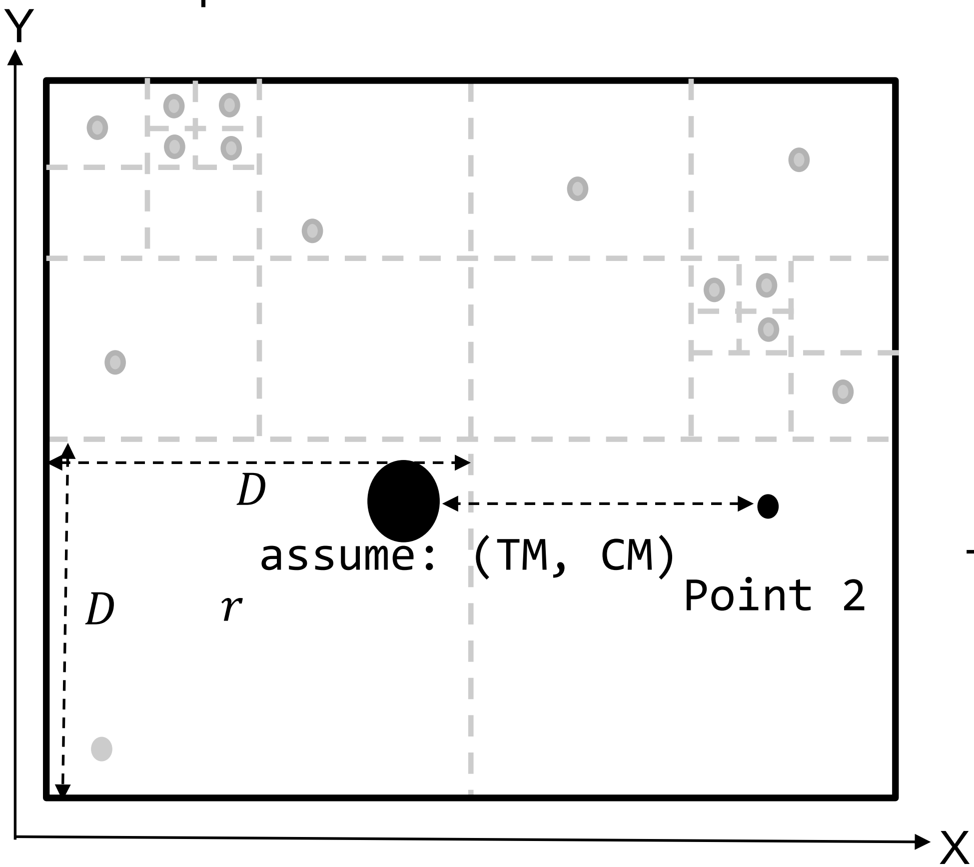
Barnes-Hut: step 3 example

- Example: Assume $\theta \geq 1$. In practice $\theta < 1$.

What is the force on **Point 2** due to all other points in the box with black-boundary?

Point 2: is $D/r < \theta$?





Traverse the tree for particle 2.

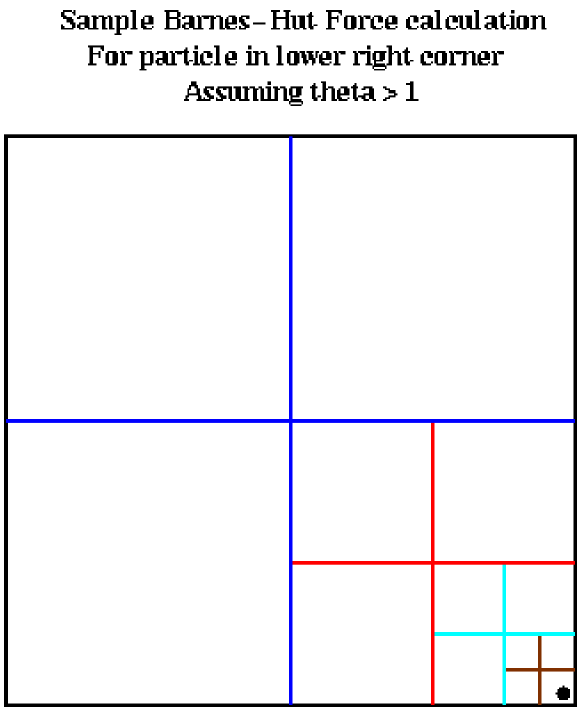
Barnes-Hut: Algorithm (step 3 cost)

- **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 of leaf containing k in the QuadTree)**
  - Proof by Example (for $\theta>1$):

- For each undivided node = square, (except one containing k), D/r < 1 < θ
- There are at most 3 undivided 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



Slide based on : CS267 Lecture 24, https://sites.google.com/lbl.gov/cs267-spr2019/

Barnes-Hut: Algorithm (step 3 cost)

(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 Center of Mass (CM) and total mass (TM) of all the particles it contains.
   … cost = O(number of nodes in the tree) = O(N log N) or O(b N)

3) For each particle, traverse the QuadTree to compute the force on it,
   … cost depends on accuracy desired (θ) but still O(N log N) or O(bN)

N-Body Simulation: Big Picture

- Recall:

```
t=0
while(t<t^final) {
//initialize forces

//Accumulate forces
      BH(steps 1 to 3)

//Integrate equations of motion

//Update time counter
      t = t + Δt
}
```