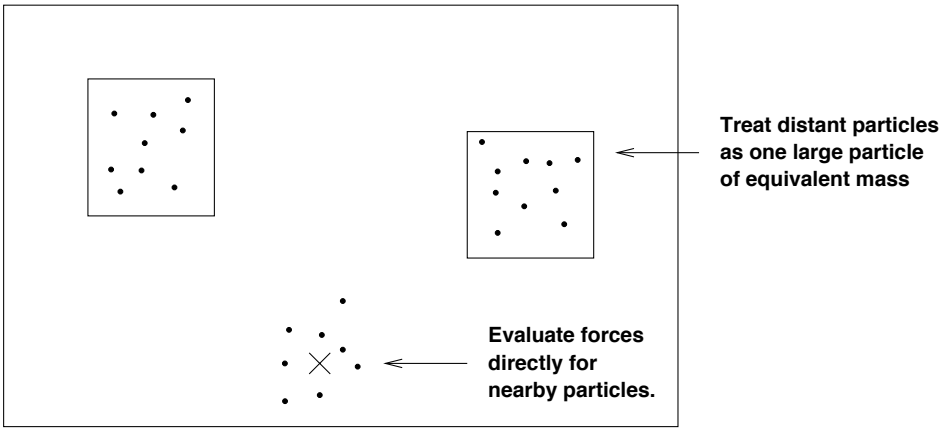
Class 21. *N*-body Techniques, Part 4

Tree Codes

Efficiency can be increased by grouping particles together:

Nearest particles exert greatest forces → direct summation.

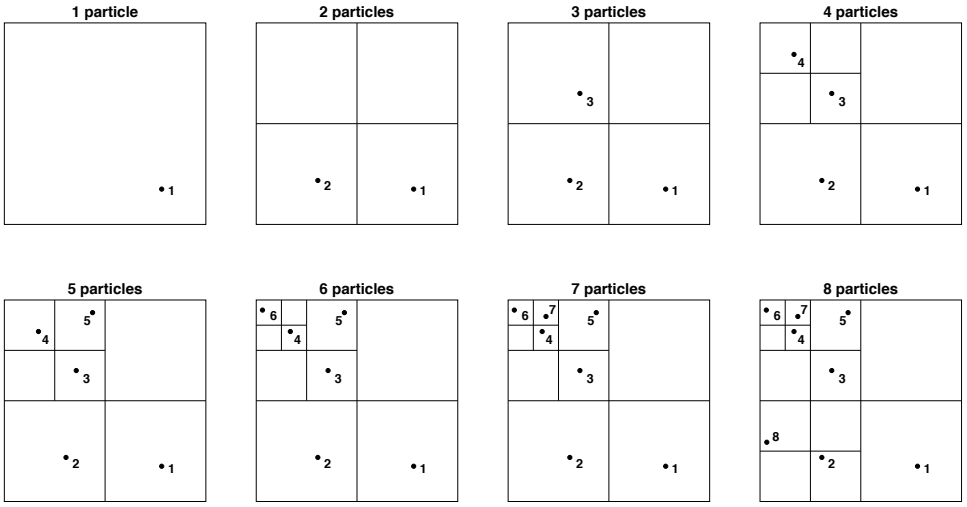
Distant particles exert smallest forces → treat in groups.



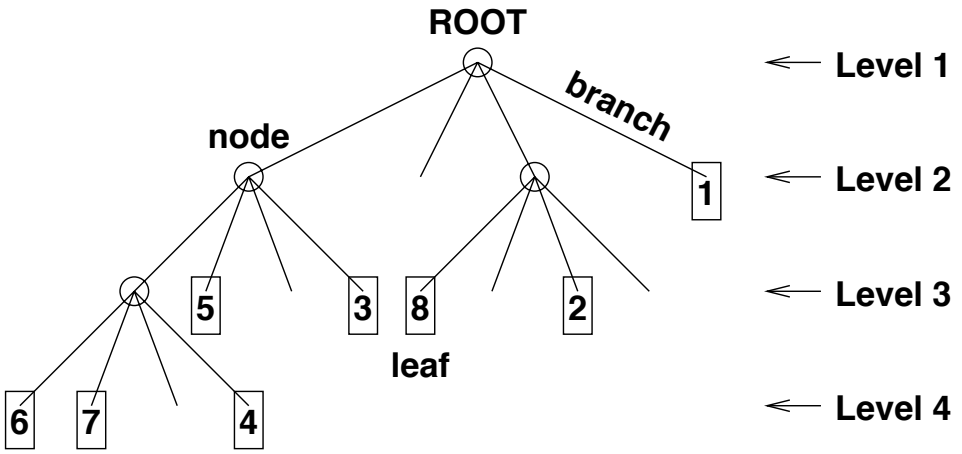
But how do we organize particles into groups? Will sketch one method (Barnes & Hut 1986, *Nature* **324**, 426; also see Hernquist 1987, *ApJS* **64**, 715), then go into more detail.

Barnes & Hut method: Overview

- The BH method is a hierarchical force-calculation algorithm:
  - Place particles on mesh one at a time.
  - Divide mesh into equal volume subdomains at each placement so that each particle occupies a single subdomain. E.g., in 2-D:



– Now, organize particles based on nesting of subdomains:



- How does this speed up force evaluation? Consider evaluation of force on particle 1:

  – If any subdomain subtends an angle $\theta = l/d < \theta_{\rm crit}$ as seen from particle 1 (l is size of subdomain, d is distance from particle 1), then treat all particles in that subdomain as one. E.g.,

  Particle 2, 8: treat directly.

  Top-left subdomain: treat as group.

  $\Longrightarrow$ just 3 summations, instead of 7.

Barnes & Hut method: Details

- Cost of tree build depends on required depth (number of levels). For homogeneous particle distribution (i.e., no cells empty), tree depth $\simeq 1 + \log_{2^k} N$. For $k = 3$, depth $\sim 1 + \log N$. $\therefore$ time required to construct tree $\sim \mathcal{O}(N \log N)$.

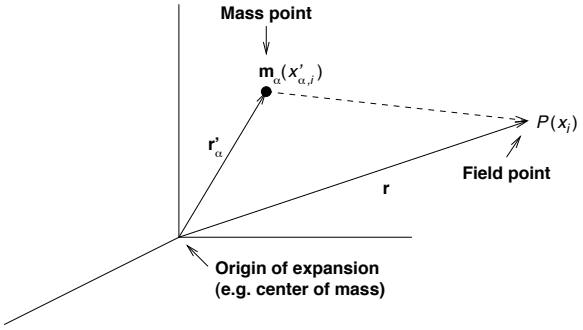
- Must also compute total mass and center-of-mass position $\Longrightarrow$ one more $\mathcal{O}(N \log N)$ pass through tree.

- Finally, force evaluation ("pruning") $\Longrightarrow \sim 2^k - 1$ sums per particle at each level $\Longrightarrow$ $\mathcal{O}(\log N)$ sums per particle (depends on $\theta_{\rm crit}$) $\Longrightarrow \mathcal{O}(N \log N)$ scaling $\ll N^2$ for $N \gg 1$.

How bad an approximation is it?

- Consider expanding potential of cell α (e.g., Marion & Heald 1980, pp. 38–40; this comes from Taylor series expansion of potential near origin):

$$\Phi_\alpha = -\frac{Gm_\alpha}{r} + Gm_\alpha \sum_i x'_{\alpha,i} \frac{\partial}{\partial x_i}\left(\frac{1}{r}\right) - \frac{Gm_\alpha}{2} \sum_{i,j} x'_{\alpha,i} x'_{\alpha,j} \frac{\partial^2}{\partial x_i \partial x_j}\left(\frac{1}{r}\right) + \cdots$$



so

$$\Phi = \sum_\alpha \Phi_\alpha = \Phi^{(1)} + \Phi^{(2)} + \Phi^{(4)} + \cdots + \Phi^{(2^l)} + \cdots$$

where

$$\Phi^{(1)} \equiv -\sum_\alpha \frac{Gm_\alpha}{r} = -\frac{GM}{r} \text{ is the "monopole",}$$

$$\Phi^{(2)} \equiv \sum_\alpha Gm_\alpha \sum_i x'_{\alpha,i} \frac{\partial}{\partial x_i}\left(\frac{1}{r}\right) \text{ is the "dipole",}$$

$$\Phi^{(4)} \equiv -\frac{1}{2}\sum_\alpha Gm_\alpha \sum_{i,j} x'_{\alpha,i}x'_{\alpha,j} \frac{\partial^2}{\partial x_i \partial x_j}\left(\frac{1}{r}\right) \text{ is the "quadrupole",}$$

$$\Phi^{(2^l)} \equiv \frac{(-1)^{(l+1)}}{l!}\sum_\alpha Gm_\alpha \sum_{i,j,\ldots,l} x'_{\alpha,i}x'_{\alpha,j}\cdots x'_{\alpha,l} \frac{\partial^l}{\partial x_i \partial x_j \cdots \partial x_l}\left(\frac{1}{r}\right) \text{ is the "}2^l\text{-pole".}$$

- If we choose expansion center to be center of mass of group, then $\sum_\alpha m_\alpha \mathbf{r}'_\alpha = 0$. But then notice that $\Phi^{(2)} = \sum_\alpha Gm_\alpha \mathbf{r}'_\alpha \cdot \boldsymbol{\nabla}(1/r) = 0$, so dipole vanishes. $\therefore$ error term dominated by quadrupole.

- (Can also write

$$\Phi = -\frac{GM}{r} - \frac{1}{2}\frac{G}{r^5}(\mathbf{rQr}),$$

where

$$Q_{ij} = \sum_k m_k(3x_{k,i}x_{k,j} - r_k^2\delta_{ij})$$

is the traceless quadrupole tensor, k is over the mass components, and $\mathbf{r}_k$ is relative to the cell center of mass. With this notation, and invoking the parallel axis theorem, the quadrupole of a parent cell can be constructed via the quadrupoles of its daughter cells: $\mathbf{Q} = \sum_i \mathbf{Q}_i + \sum_i m_i(3\mathbf{r}_i\mathbf{r}_i - r_i^2\mathbf{1})$, where i is over the daughter cells and $\mathbf{r}_i$ is relative to the parent center of mass.)

- Often, quadrupole not needed (monopole is "good enough").

- With quadrupole, for $\theta_{\rm crit} = 1$, forces typically accurate to $\sim 1\%$ (in practice, keep $\theta_{\rm crit} < 1/\sqrt{2} = 0.7$ for 2-D tree, $< 1/\sqrt{3} = 0.6$ for 3-D tree). This is *average* error; certain pathological configurations can give much larger errors. Also, trees in general break $\mathbf{F}_{ij} = -\mathbf{F}_{ji}$...

- For high precision, might consider octopole.
  - Turns out the octopole does *not* help convergence much—need to go to next higher order, the hexadecapole!
  - Obviously this means many more computations to compute force (still scales as $\mathcal{O}(N \log N)$), but can use larger $\theta_{\rm crit}$.
- On balance, probably *never* need better than hexadecapole.

Barnes & Hut method: Pseudocode

Define a node `struct`: contains size, center, mass, position, **Q**, etc. of cell, plus info on subcells (may be nodes). Following example stores only monopole (i.e., total mass).

Tree build — start with special cell ("root")

```
start
   root = new node [includes initialization]
   loop over particles i
      put_in_tree(i,root)
   calc_moments(root)

function put_in_tree(particle,node)
   to which (sub)cell does particle belong?
   is cell...
      ...empty?  : make particle a leaf in cell
                   break
      ...a leaf? : make cell a node
                   cell = new node
                   put_in_tree(leaf,cell)
      ...a node? : put_in_tree(particle,cell)

function calc_moments(node)
  [loop over non-empty (sub)cells
   is cell...
      ...a leaf? : node->mass += cell.leaf->mass
                   node->pos += (cell.leaf->mass)*(cell.leaf->pos)
                   break
      ...a node? : calc_moments(cell.node)
                   node->mass += cell.node->mass
                   node->pos += (cell.node->mass)*(cell.node->pos)
  ]
   node->pos /= node->mass
```

Tree walk — start at root

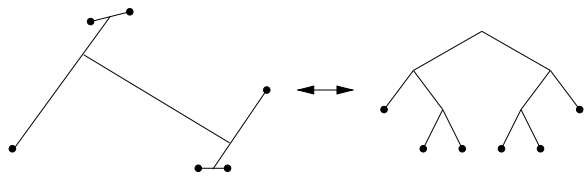
```
function add_to_force(pos,node,force)
   theta = (node->size)/(distance to node)
   theta < theta_crit? : multipole_expansion(pos,node,force) ["prune"]
                 else  : [loop over non-empty (sub)cells
                          is cell...
                             ...a leaf? : direct_force(pos,cell.leaf,force)
                                          break
                             ...a node? : add_to_force(pos,cell.node,force)
                         ]
```

Other Types of Trees

- Differ primarily in organization of particle information.

Mutually nearest neighbour

- E.g., Appel 1981, Jernigan 1985, Porter 1985.
- Given N particles, two nearest joined together $\rightarrow$ node, leaving $N-1$ entities ($N-2$ particles plus 1 node) in list.
- Node contains total mass and center-of-mass position of cluster.
- Repeat until only 1 cluster remains.
- $\mathcal{O}(\log_2 N)$ levels (binary tree), $\mathcal{O}(N \log N)$ update time.

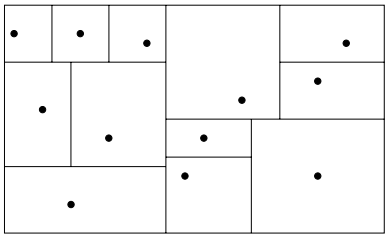


- Advantage: Preserves physical proximity of particles (binaries). Can also let particles "drift" a while before update.
- Disadvantage: Arbitrary node shapes, hard to estimate error when expanding potentials.

k-D tree (recursively bisect longest dimension)

- E.g., Olson & Packer 1996.
- First determine dimension (x, y, or z) that spans largest spatial range of particle distribution.

- Sort data on this dimension and divide into halves containing equal numbers of particles.
- Repeat with sublists until each contains only 1 particle.
- Often used for "domain decomposition" to balance work between multiple processors.



- Advantage: No empty cells, more efficient shape.
- Disadvantage: Extreme oblong shapes → larger error.

Fast Multipole Method

- Improved tree walking/pruning.
- In principle can achieve $\mathcal{O}(N)$ scaling, and momentum conservation (!), but complex implementation.
- Idea is that local information is passed *up* the tree so it can be swapped with distant nodes: mutual multipole expansion (postal service analogy).
- Cutting edge of tree code development, much of it done here at U Maryland (Computer Science)!

Summary

- PP method (direct summation) — most accurate, but $\mathcal{O}(N^2)$.
- PM method — $\mathcal{O}(N_g \log N_g)$, but resolution limited.
- Tree codes — $\mathcal{O}(N \log N)$, but sometimes difficult to implement.
- Also: PP-PM = $\mathrm{P^3M}$ — direct summation over nearby particles, use grid for distant interactions.