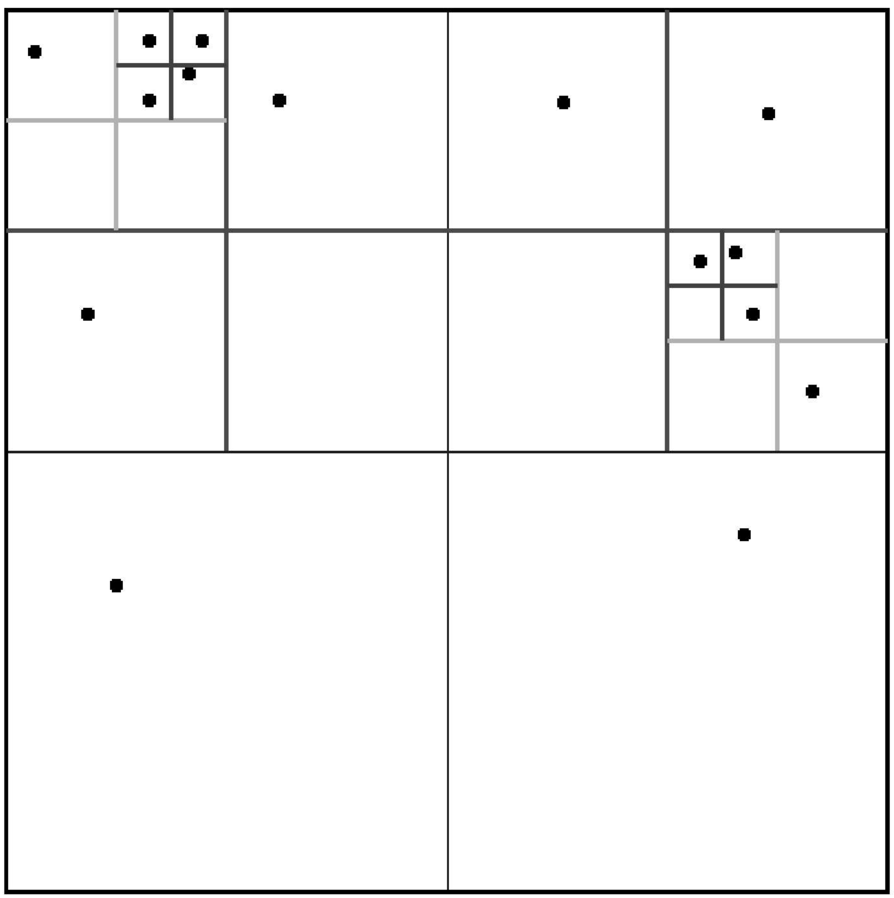
Particle Simulation, bottlenecks

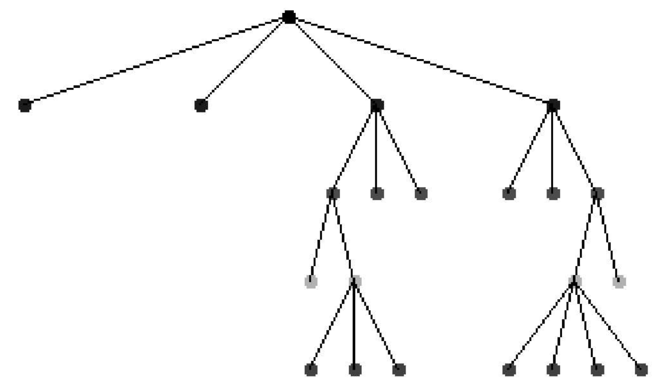
- **Computational bottlenecks?**
  - **force calculation; contains a nested loop** — **Yes**
  - **move operation; fixed amount of work per particle** — **No**
  - **calculation of properties** — **Possible**
- **For high performance, focus on bottleneck force calculation**

- **Types of force to distinguish (in practice)** — **(in parallel code)**
  - **External force** — **Trivial**
    - **Is independent on the other particles**
  - **Short-range force** — **Easy**
    - **Depends on particles within fixed range**
  - **Long-range force** — **Difficult**
    - **Depends on all other particles**

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 are excluded

Adaptive Quad Tree Building Cost

° **Cost ≤ N * maximum cost of a Quad_Tree_Insert**
= 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 coordinates),
so Cost = O(b N)

– **Note: depth of QuadTree b ≥ 1 + ^{4}log N**

Barnes-Hut Algorithm

° **High Level description:**

1) **Build the QuadTree using QuadTreeBuild**
   … already described, cost = O(N log N) or O(b N)

2) **For each node, a sub-square 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)

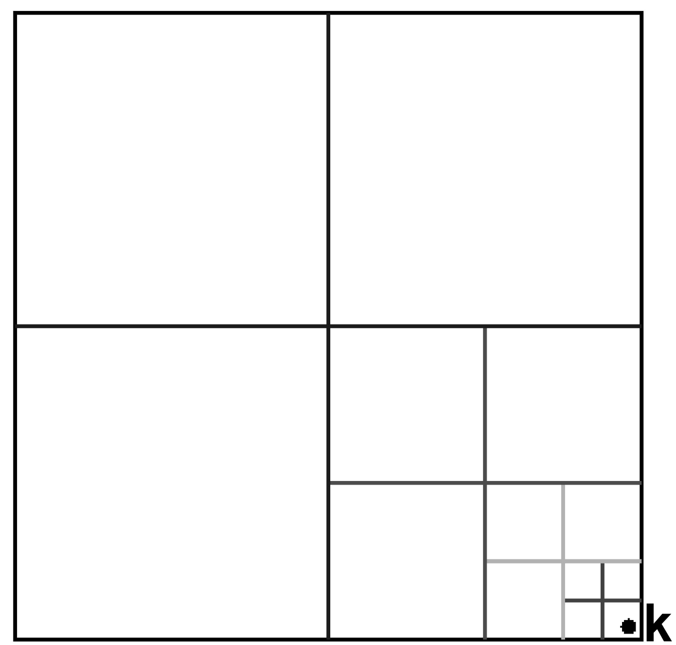
3) **For each particle, traverse the QuadTree to compute the force on it,**
   using the CM and TM of "distant" sub-squares
   … core of algorithm
   … cost depends on accuracy desired but still O(N log N) or O(bN)

Total cost: still O(N log N) or O(bN)

→ Achieved order reduction per time step: $O(N^2)$ to $O(N \log N)$

Step 3 of BH: Analysis

- **Correctness: 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"; Example: Assume $\theta = 1$**
    - **For each undivided node, see fig., (except one containing k), $D/r < 1 < \theta$**
    - **There are only 3 nodes to consider at each level of the QuadTree, see fig.**
    - **There is O(1) work per node**
    - **Cost = O(level of k)**
  - **Total cost = $O(\Sigma_k$ level of k) = O(N log N)**
    - **Strongly depends on θ**



Fast Multiple Method (FMM)

° "A fast algorithm for particle simulation", L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, many later papers

° 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.

Parallelizing Hierarchical N-Body codes

- **Barnes-Hut, FMM and related algorithms have similar computational structure:**
  1) Build the QuadTree
  2) Traverse QuadTree from leaves to root and build outer expansions (just (TM,CM) for Barnes-Hut)
  3) Traverse QuadTree from root to leaves and build any inner expansions
  4) Traverse QuadTree to accumulate forces for each particle

- **QuadTree changes dynamically when the particles move, so the tree has to be rebuilt (or adjusted) every time step.**
- **But: No doubly nested loop over all particles anywhere in the algorithm**

- **All 4 phases have to be parallelized efficiently.**

Parallelizing Hierachical N-Body codes

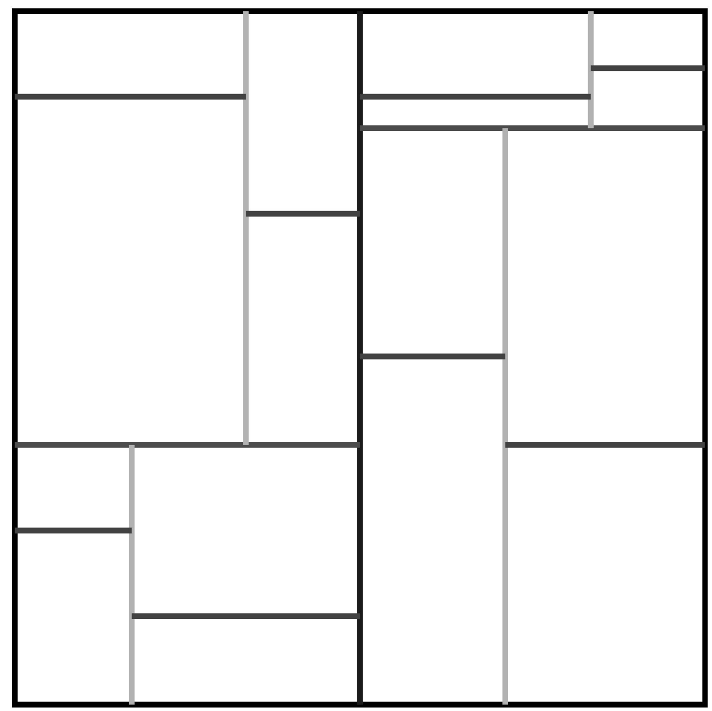
- **General idea: Domain decomposition**
  - **Assign regions of space to each processor**
  - **Regions may have different size or shape, to get a good load balance**
    - **Each region will have about N/p particles**
  - **Each processor will store part of QuadTree containing all particles (=leaves) in its region, and their ancestors in QuadTree**
    - **Root of tree and some generations stored by all processors, nodes may also be shared**
  - **Each processor will also store adjoining parts of QuadTree needed to compute forces for particles it owns**
    - **Subset of QuadTree needed by a processor called the Locally Essential Tree (LET)**
  - **Given the LET, all force accumulations (step 4)) are done in parallel, without communication**
- **Coarse grained parallelism**
  - **Each domain solves its own N-body problem, but somehow has to take into account the effect of all the other particles as well; like the ghost-points in the Poisson problem. Here those particles generate some "background" potential (in FMM)**

Load Balancing Scheme 1

- **Orthogonal Recursive Bisection (ORB) of space**
  - **Warren and Salmon, Supercomputing 92**
- **Recursively split region along axes into regions containing equal numbers of particles**
  - **Particles are grouped in rectangular regions; may be very elongated**
  - **No relation with tree**

Partitioning for 16 procs:

Orthogonal Recursive Bisection



Load Balancing Scheme 2

° **Idea: Partition QuadTree instead of space**

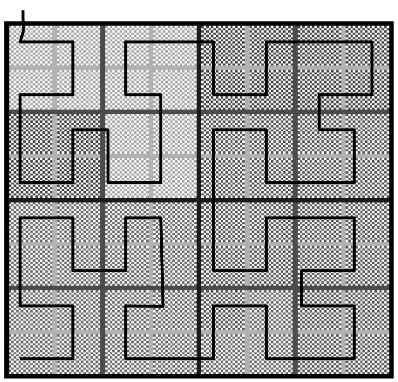
- **Estimate work for each node, call total work W**
- **Arrange nodes of QuadTree in some linear order (lots of choices)**
- **Assign contiguous groups of nodes with work W/p to processors**

° **Method called: Costzones or Hashed Tree**

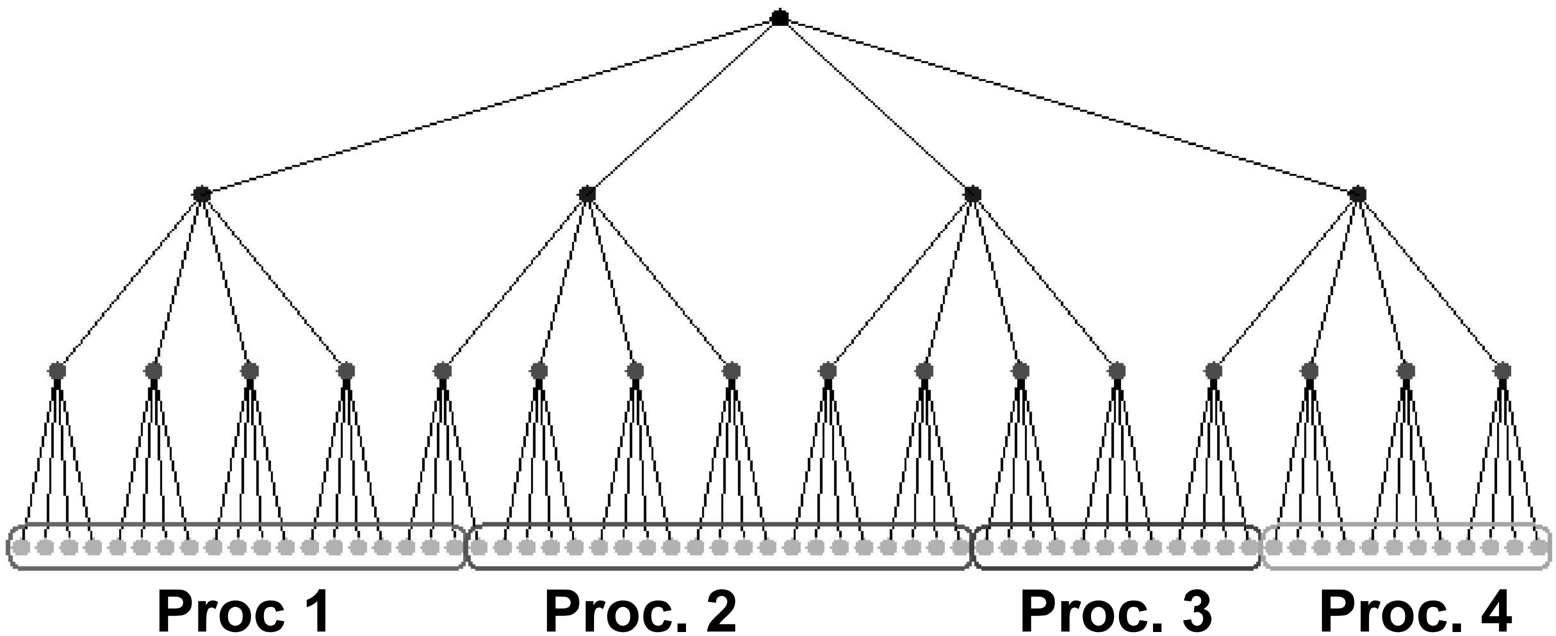
- **J.P. Singh, PhD thesis, Stanford, 1993**
- **Warren and Salmon, Supercomputing 93**

Load Balancing Scheme 2

- Make sure that neighboring leaves in the tree are also neighboring in space
  - Which of the 4 children of a node is “first” depends on the position of the node
  - Orientation changes: clockwise / counter-clockwise



Using costzones to layout a quadtree on 4 processors
Leaves are color coded by processor color



Implementing Costzones

- Random Sampling
  - All processors own some particles,
  - All processors send a small random sample of their particles to Processor 1
  - Processor 1
    - builds small Quadtree serially,
    - determines its Costzones, and
    - broadcasts them to all processors
  - Other processors build the part of Quadtree they are assigned by these Costzones

- All processors know all Costzones
- This is needed later to compute LET's

Locally Essential Trees (LETs)

- **Warren and Salmon, 1992; Liu and Bhatt, 1994**
- **Definition:**
  - **A LET of a process is that part of the Tree that is necessary to compute the force on the particles that are owned by that process.**
- **Information about nodes near the root of the tree is present in all processes**
- **Information about nodes near some leaves of the tree that are all owned by a single process is needed in one or a few processes**

Computing Locally Essential Trees (LETs)

° Warren and Salmon, 1992; Liu and Bhatt, 1994

° Every processor needs a subset of the whole QuadTree, called the LET, to compute the force on all particles it owns

° Shared Memory

- Receiver driven protocol
- Each processor reads part of QuadTree it needs from shared memory on demand, keeps it in cache
- Drawback: cache memory appears to need to grow proportionally to P to remain scalable

° Distributed Memory

- Sender driven protocol
- Each processor decides which other processors need parts of its local subset of the Quadtree, and sends these subsets

Barnes-Hut

Which nodes are needed?

- Let j and k be processes, n a node on process j
- Let D(n) be the side length of n
- Let $r_k(n)$ be the shortest distance from n to any point owned by k
- If either

  (1) $D(n)/r_k(n) < \theta$ and $D(parent(n))/r_k(parent(n)) \geq \theta$, or

  (2) $D(n)/r_k(n) \geq \theta$

  then node n is part of k's LET, and so process j should send n to k

- Condition (1) means (TM,CM) of n can be used on process k, but this is not true of any ancestor
- Condition (2) means that we need the ancestors of type (1) nodes too