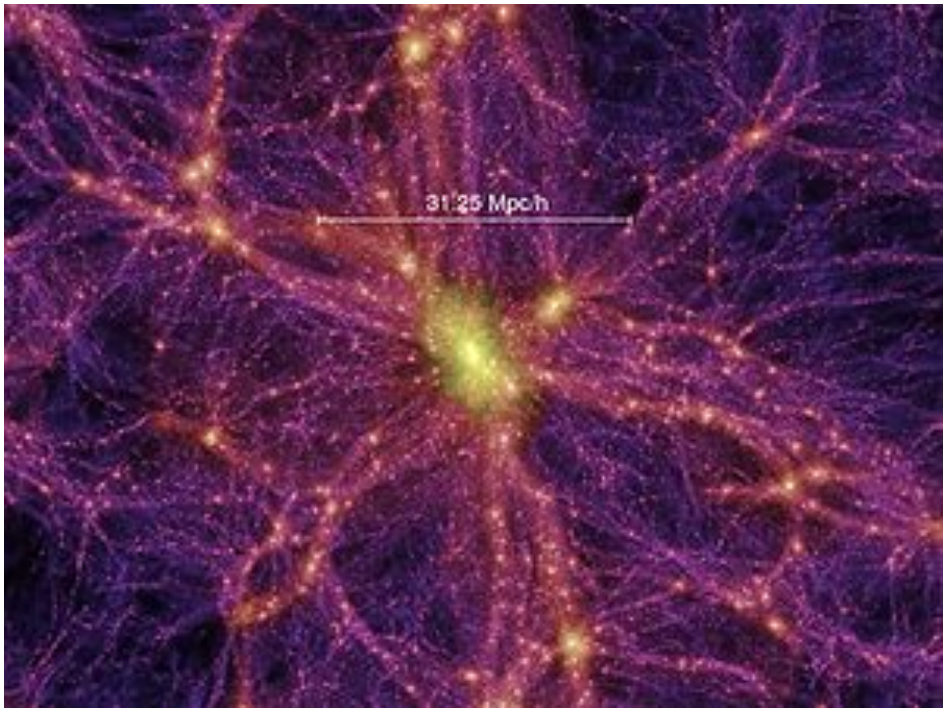
Partitioning and Divide-and-Conquer Strategies – Part II

CPSC 424/524
Lecture #10
October 31, 2018

Gravitational N-Body Problem

Model positions and movements of bodies in space subject to gravitational forces from other bodies, using Newtonian physics.

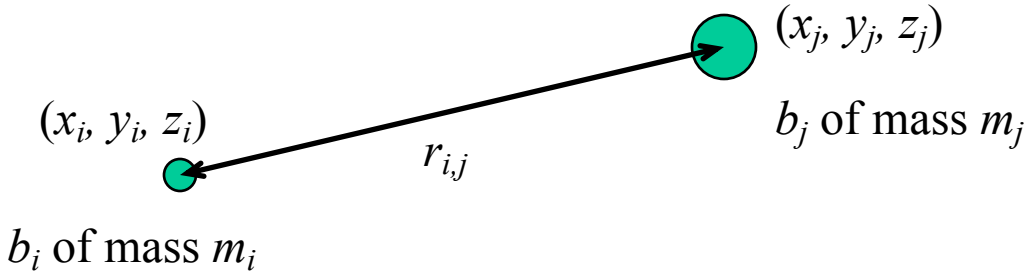
Example



Millennium Simulation traced 2160^3, or just over 10 billion, "particles" (each representing ~1 billion solar masses of dark matter) in a cube of side ~2 billion light years populated with ~20 million “galaxies.”

Required over 1 month of time on an IBM supercomputer, and generated 25 Terabytes of output

Gravitational N-Body Problem Equations



Magnitude of the gravitational force between bodies b_i and b_j is:

$$F_{i,j} = \frac{Gm_i m_j}{r_{i,j}^2}$$

G is the gravitational constant and the distance $r_{i,j}$ is

$$r_{i,j} = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2 + (z_j - z_i)^2}$$

Details

Gravitational force on b_i due to b_j can be resolved as:

$$F_{i,j\,x} = \frac{G m_i m_j}{r_{i,j}^2}\left(\frac{x_j - x_i}{r_{i,j}}\right)$$

$$F_{i,j\,y} = \frac{G m_i m_j}{r_{i,j}^2}\left(\frac{y_j - y_i}{r_{i,j}}\right)$$

$$F_{i,j\,z} = \frac{G m_i m_j}{r_{i,j}^2}\left(\frac{z_j - z_i}{r_{i,j}}\right)$$

Details

Summing, the components of total gravitational force on b_i are:

$$F_{ix} = Gm_i \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{m_j}{r_{i,j}^2} \left(\frac{x_j - x_i}{r_{i,j}} \right)$$

$$F_{iy} = Gm_i \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{m_j}{r_{i,j}^2} \left(\frac{y_j - y_i}{r_{i,j}} \right)$$

$$F_{iz} = Gm_i \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{m_j}{r_{i,j}^2} \left(\frac{z_j - z_i}{r_{i,j}} \right)$$

Details

From Newton's Second Law, we can compute the acceleration of b_i:

$$a_{i\,x} = \frac{1}{m_i} F_{i_x} \qquad a_{i\,y} = \frac{1}{m_i} F_{i_y} \qquad a_{i\,z} = \frac{1}{m_i} F_{i_z}$$

Computationally, we use these equations to compute the locations and velocities of the bodies at times t_0, t_1, t_2, …, where initial conditions at time are t_0 known, and, for a constant Δt, $t_{i+1} = t_i + \Delta t$

Details

Assuming constant acceleration over a time step interval, we can compute the average velocity over the interval $[t_k, t_{k+1}]$:

$$\bar{v}_{i_x} = v_{i_x}^k + \left(\frac{\Delta t}{2}\right) a_{i_x} \qquad \bar{v}_{i_y} = v_{i_y}^k + \left(\frac{\Delta t}{2}\right) a_{i_y} \qquad \bar{v}_{i_z} = v_{i_z}^k + \left(\frac{\Delta t}{2}\right) a_{i_z}$$

where the superscript k denotes a value at time t_k

From this, we can compute the new position at time t_{k+1}:

$$x_i^{k+1} = x_i^k + \bar{v}_{i_x} \Delta t$$

$$y_i^{k+1} = y_i^k + \bar{v}_{i_y} \Delta t$$

$$z_i^{k+1} = z_i^k + \bar{v}_{i_z} \Delta t$$

1-D Sequential Code

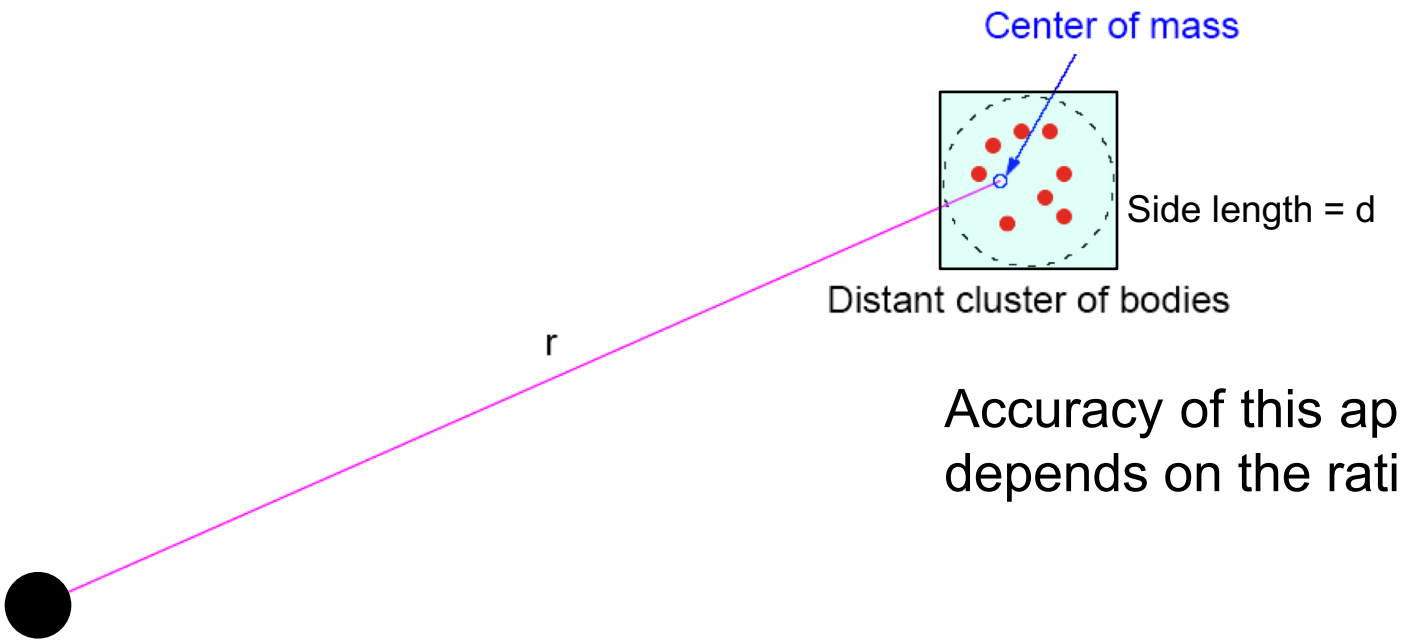
Overall gravitational *N*-body computation can be described by:

```
for (t = 0; t < tmax; t++) {          // for each time step
   F = Force_routine();               //   compute forces on all bodies
   for (i = 0; i < N; i++) {          //   for each body
      a = (F/m);                      //      compute: acceleration
      vavg = v[i] + a * dt / 2.;      //               average velocity
      x[i] = x[i] + vavg * dt;        //               new position
      v[i] = v[i] + a * dt;           //               new velocity
   }
}
```

Cost Analysis

Since there are N bodies, and each one affects all of the others, the cost of this algorithm is $O(N^2)$, which becomes prohibitive for large N

Time complexity can be reduced substantially (to $O(N\ logN)$) by approximating a cluster of distant bodies as a single distant body with mass sited at the center of mass of the cluster:



Accuracy of this approach depends on the ratio d/r

Barnes-Hut Algorithm

Structured implementation of the clustering idea based on the use of grid-based tree data structure for the bodies:

For each time step:

- Start with a cube containing all the bodies.
- Divide the cube into 8 subcubes. For each subcube *s*:
  - If *s* contains no particles, discard it (for this time step)
  - If *s* contains one body, it is a “body” cube
  - If *s* contains more than one body, recursively subdivide it

Barnes-Hut Algorithm (cont.)

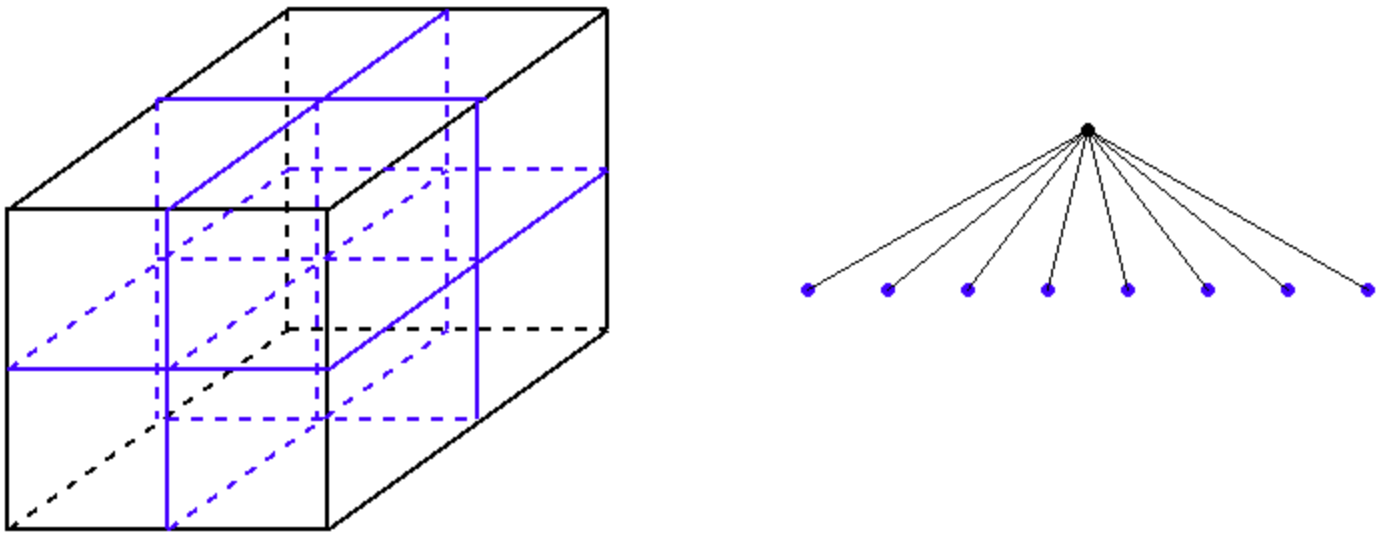
This process can be represented using a tree (an *octtree* in 3-D; a quadtree in 2-D).

- The root represents the cube containing all the bodies.
- The leaves represent subcubes containing one body.
- Intermediate nodes represent subtrees corresponding to multi-body subcubes.

As the tree is being constructed, calculate total mass and center of mass of the bodies in each subcube and store that data at the corresponding subtree root.

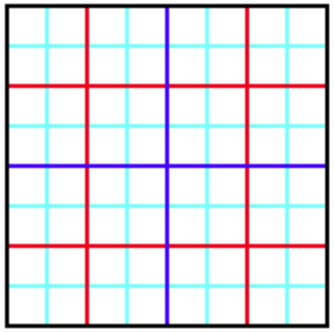
Octree

2 Levels of an Octree

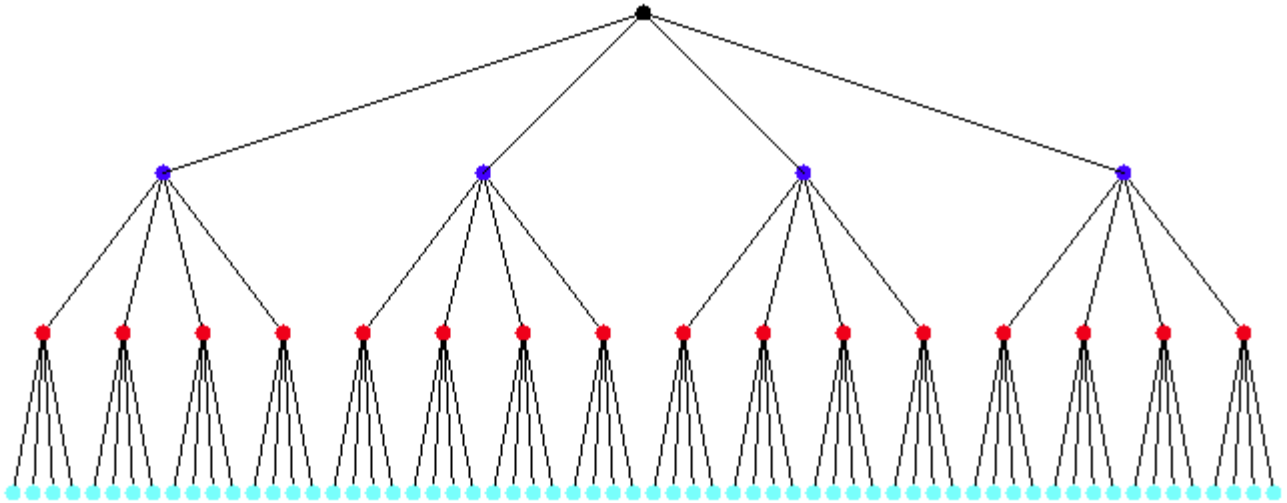


Material from http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Quadtree

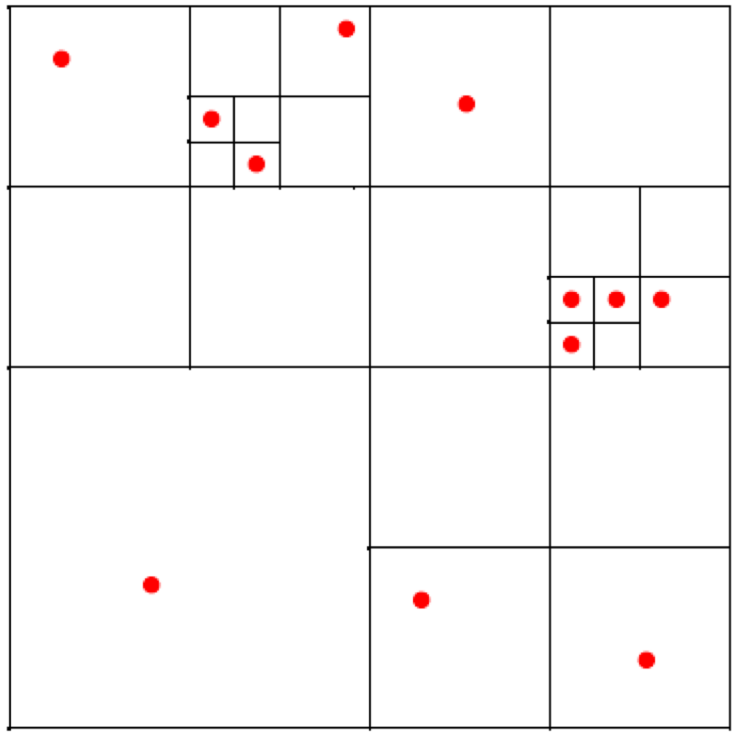


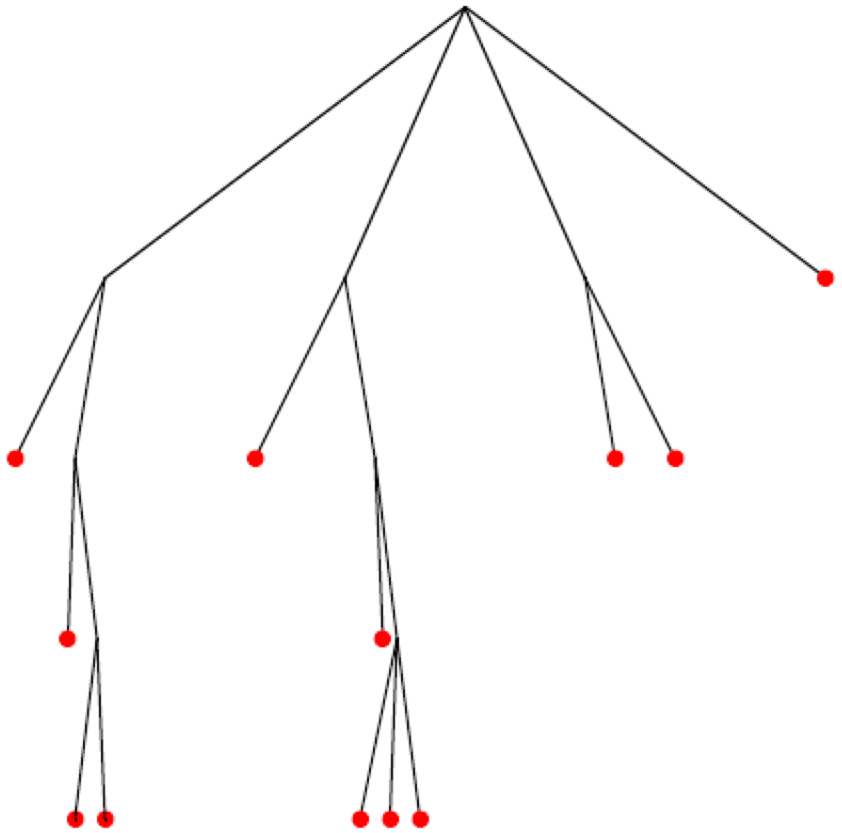
A Complete Quadtree with 4 Levels



Material from http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

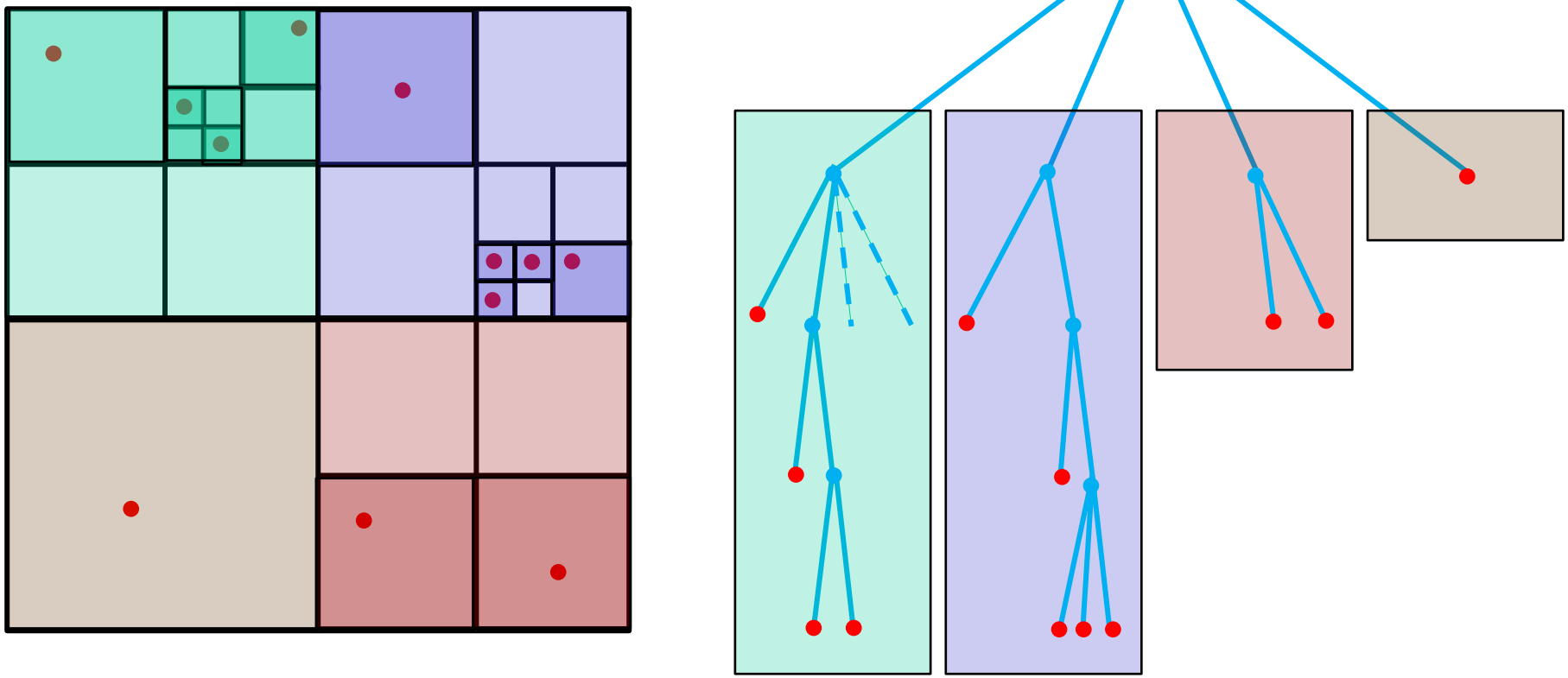
Building the Quadtree



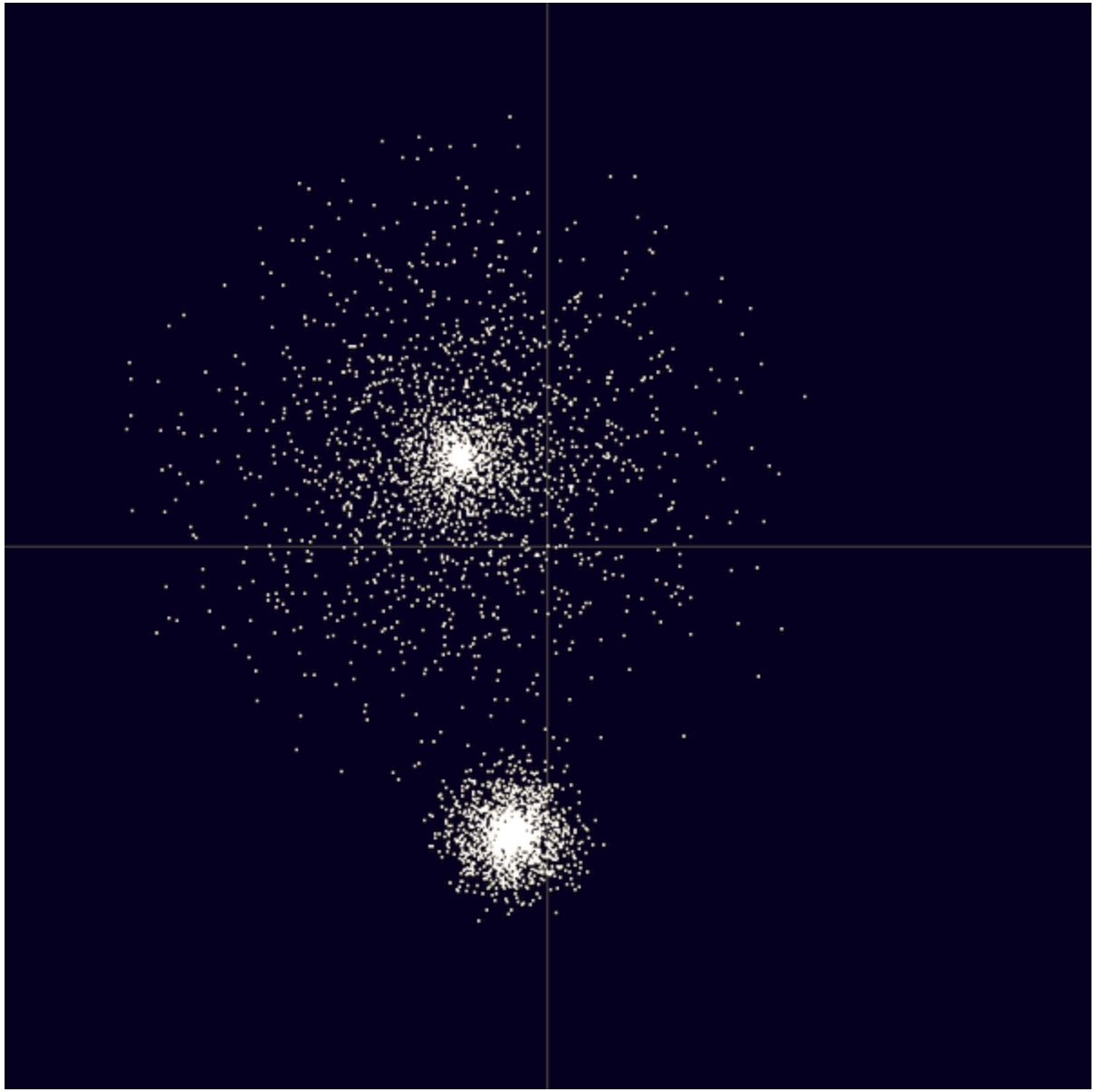


Partial quadtree

Subdivision of 2-D space



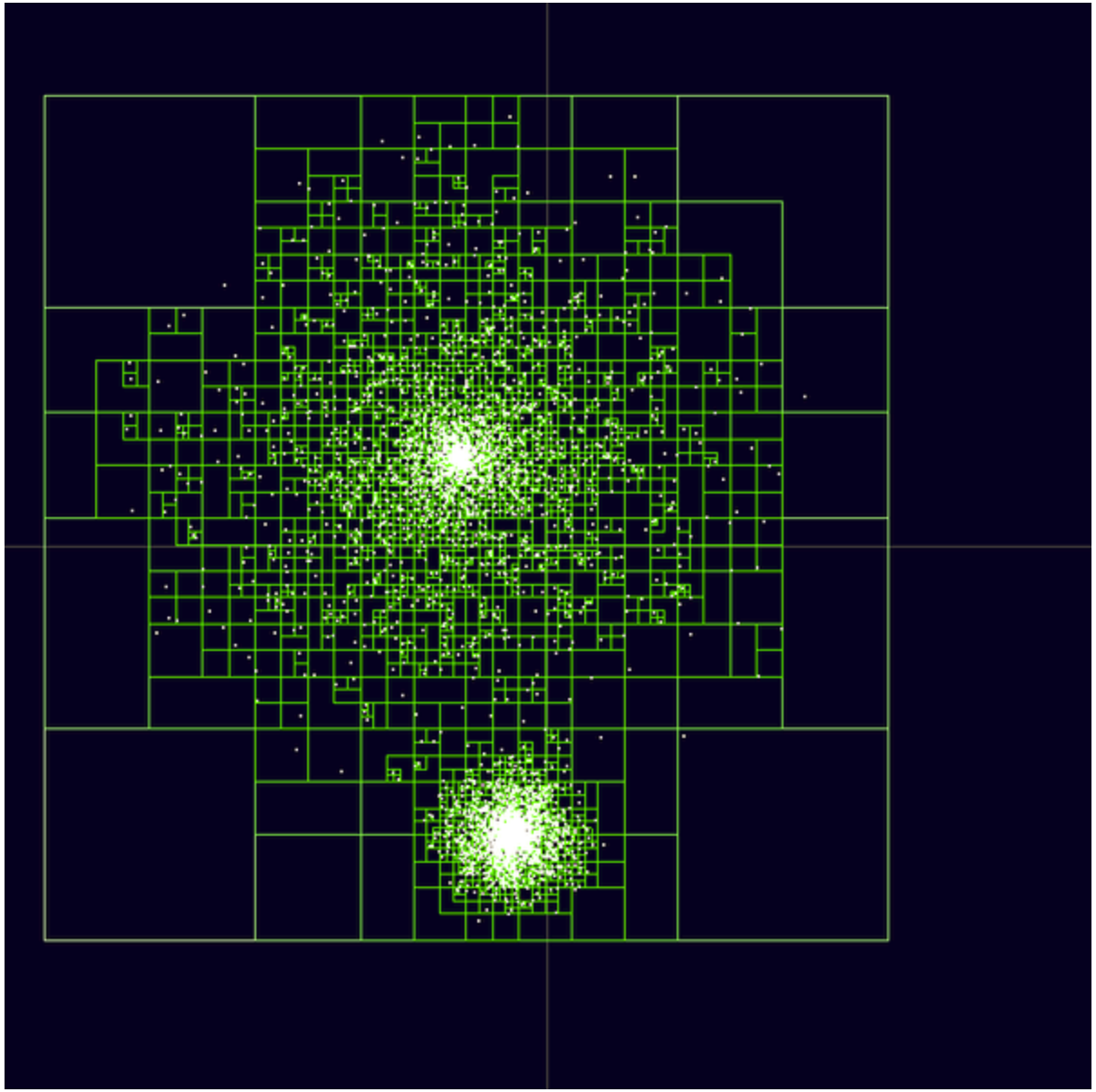
Barnes-Hut Example



Initial distribution of 5,000 bodies in 2 simulated galaxies

Source for this and other images and for video: Ingo Berg from Wikipedia. (http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation)

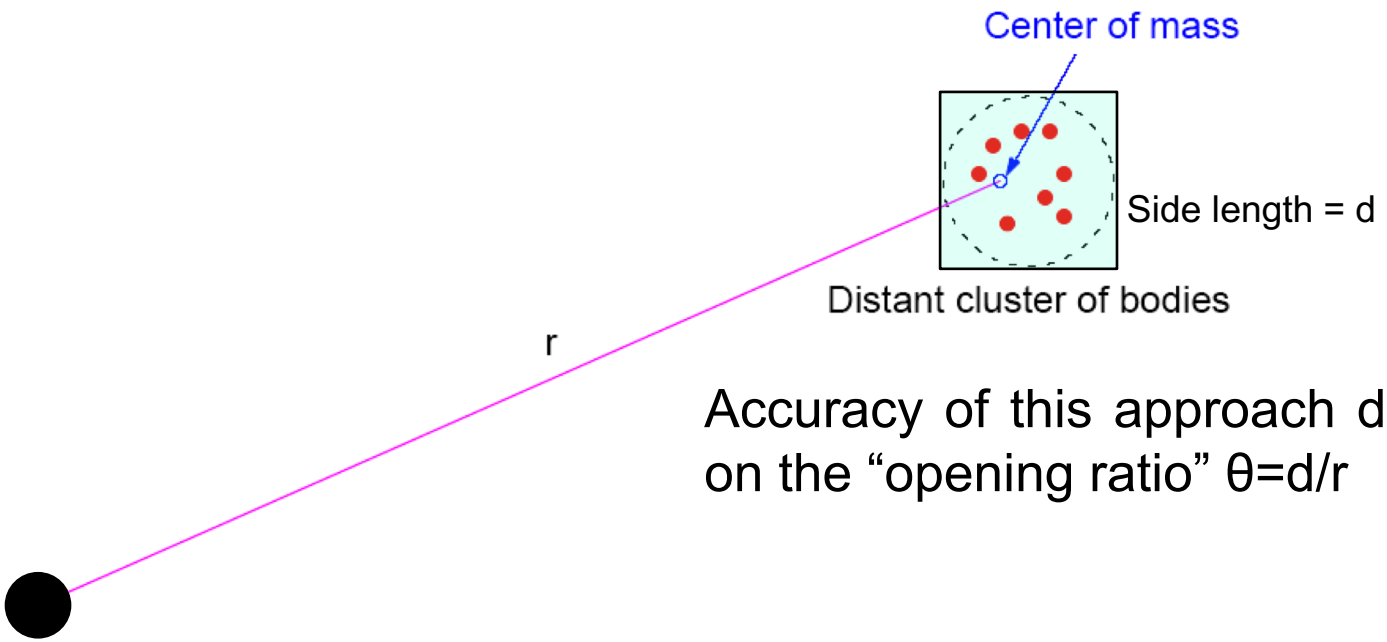
Barnes-Hut Full Partition



Shows full partition for 5,000 bodies, each in its own cell.
(Empty cells omitted.)

Clustering Approximation

Approximate the effect of a cluster of distant bodies by treating them as a single distant body with mass located at the center of mass of the cluster:



Accuracy of this approach depends on the "opening ratio" θ=d/r

Barnes-Hut Force Computation

Force on a given body *b* obtained by walking the tree, starting at the root.

At each visited node, check the "opening ratio" to see whether the clustering approximation can be used for the subtree rooted at the node:

$$\frac{d}{r} \leq \theta$$

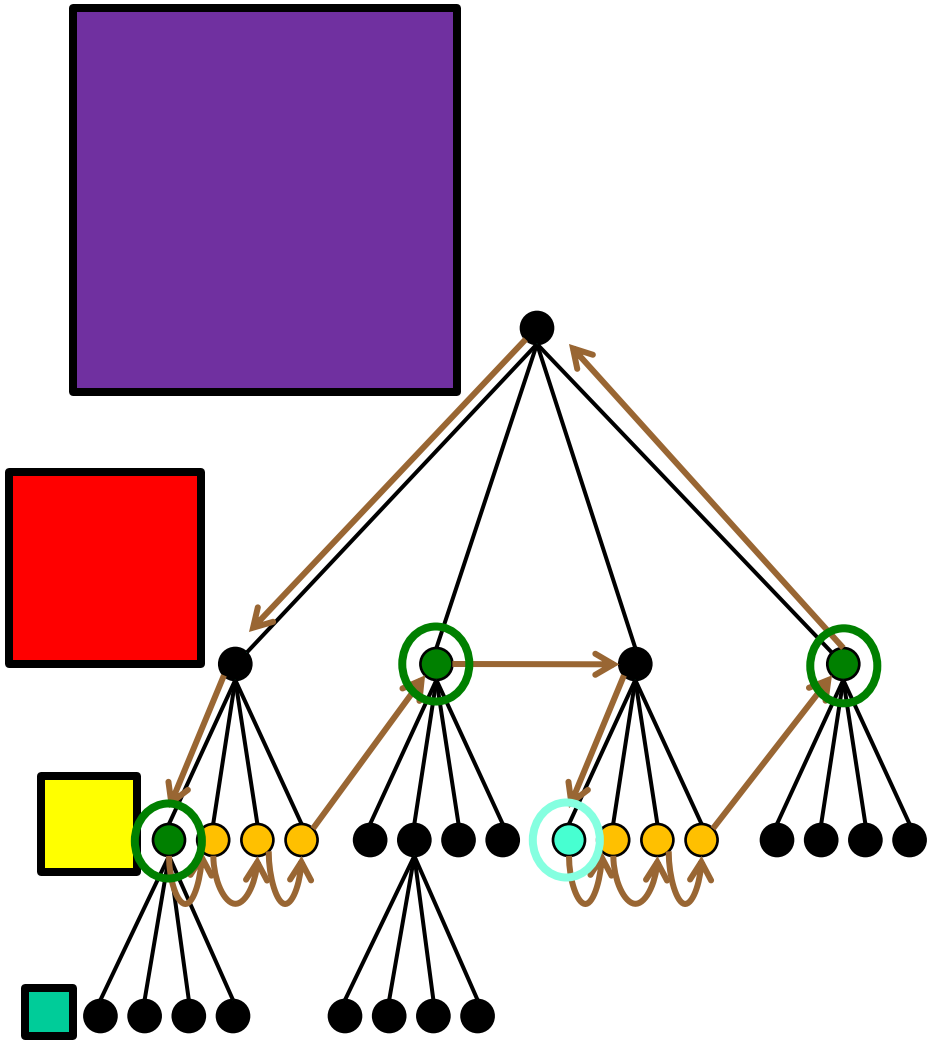
where

θ is a constant (typically $\theta \leq 1.0$),

d is the side length of the subcube that bounds the bodies in the subtree rooted at the node, and

r is the distance from *b* to the center of mass of those bodies.

Walking the Barnes-Hut tree



The force computation for body *b* starts at the root of the tree, and descends in the tree to nodes representing smaller and smaller cells until the "opening test" ($d/r < \theta$) is passed. Then the aggregated mass and c.o.m. at that node is used instead of descending further. From there the walk moves to the right (a sibling of the "unopened" cell, if there is one) or up (a sibling of the cell's parent, if there is one).

Barnes-Hut Force Computation

```
For (i = 1; i<N i++)
    f(i) = F(i,root);

float F(i,n){
  // Compute gravitational force on body i
  // due to all bodies in the subtree rooted at node n
  f = 0.;
  if (n contains one body)
    f = compute the force from this body;
  else {
    r = distance from i to the c.o.m. of bodies in subtree;
    d = side of bounding box for subtree;
    if (d/r < theta) compute f using info about subtree;
    else
        for all children c of n
            {f = f + F(i,c)};
  }
}
```

Material based on http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Barnes-Hut Force Computation

Force on a given body *b* obtained by walking the tree, starting at the root.

At each visited node, check the "opening ratio" to see whether the clustering approximation can be used for the subtree rooted at the node:

$$\frac{d}{r} \leq \theta$$

where

θ is a constant (typically $\theta \leq 1.0$),

d is the side length of the subcube that bounds the bodies in the subtree rooted at the node, and

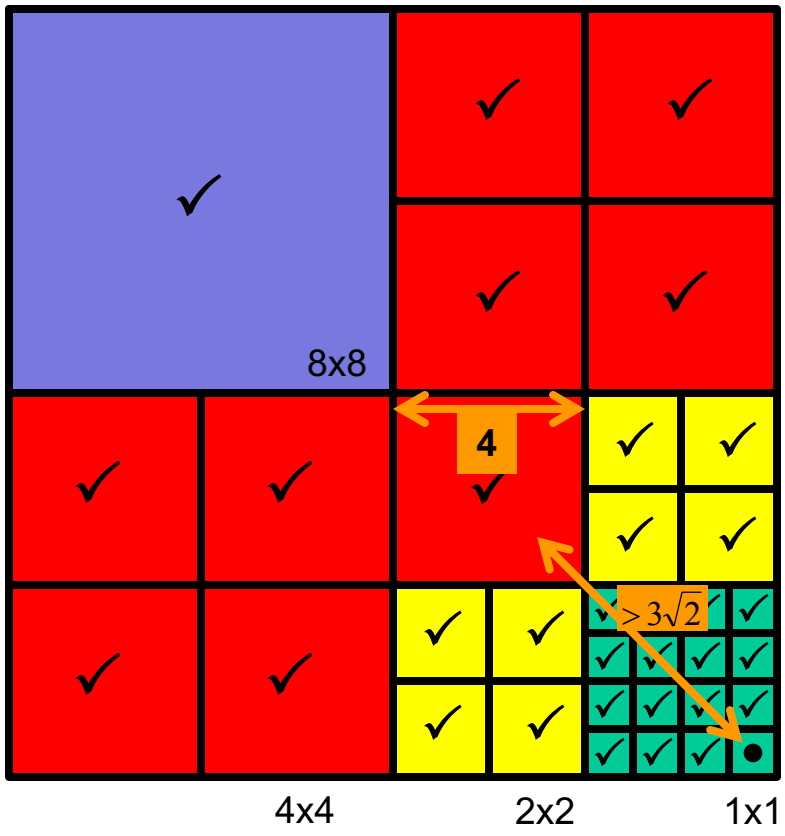
r is the distance from *b* to the center of mass of those bodies.

Times: Constructing tree: O(*n* log *n*) time.
Walking tree and computing forces for b: O(log n)
Overall time complexity of method is O(*n* log *n*) per time step.

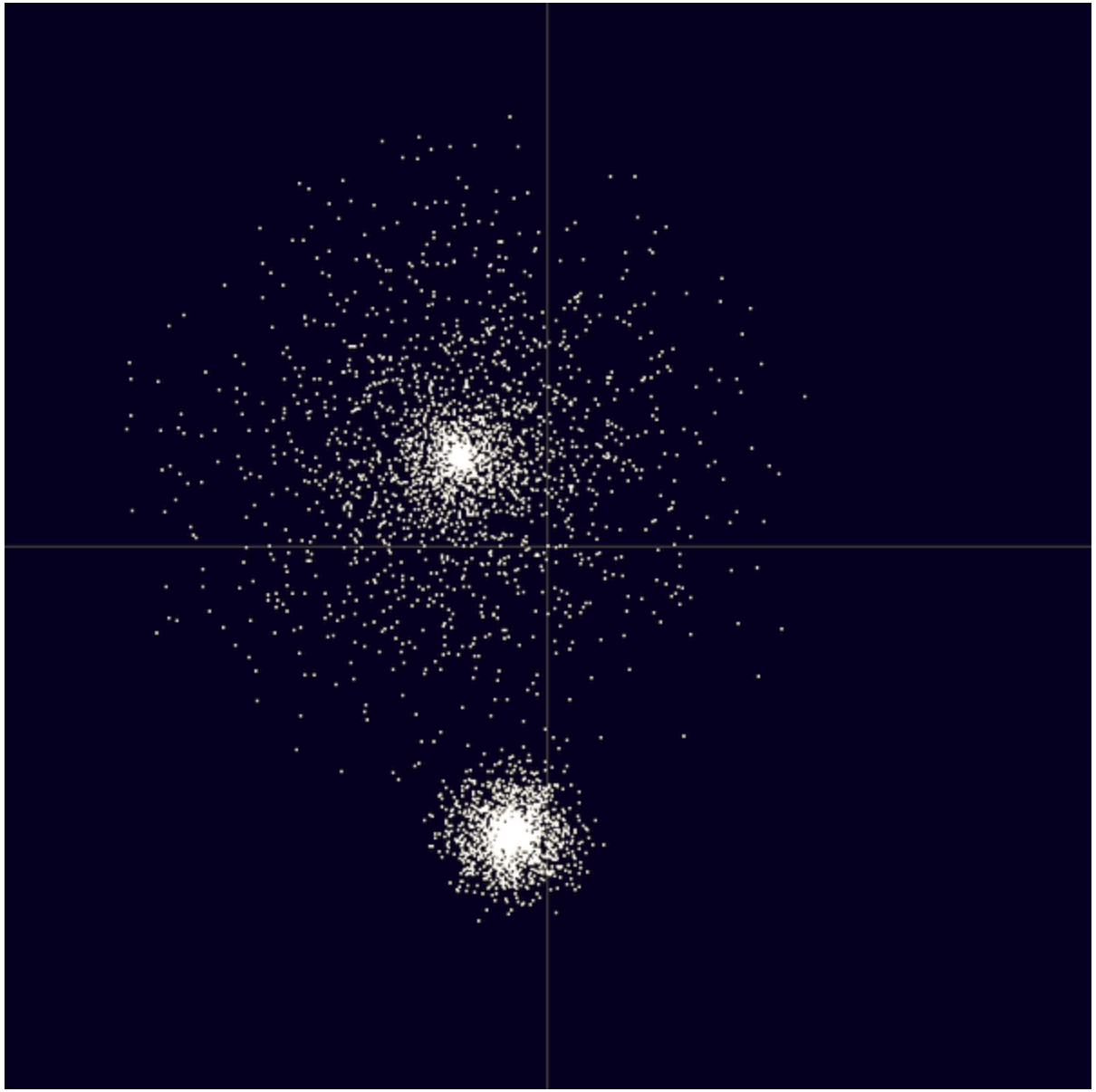
Computational Cost

For each body, the amount of work is proportional to its depth in the tree, since only a bounded number of cells at each level of the tree need to be used. Since the maximum depth is $O(\log N)$, this bounds the overall cost of force calculation as $O(N \log N)$.

Example: Suppose that θ=1. For the body shown at the lower right of the figure, the work involves a force calculation at no more than the set of check-marked squares. (Note that the c.o.m. of a cell may be very close to the edge of the cell.) This involves no more than 9 cells of each color except for the cells very near the body.



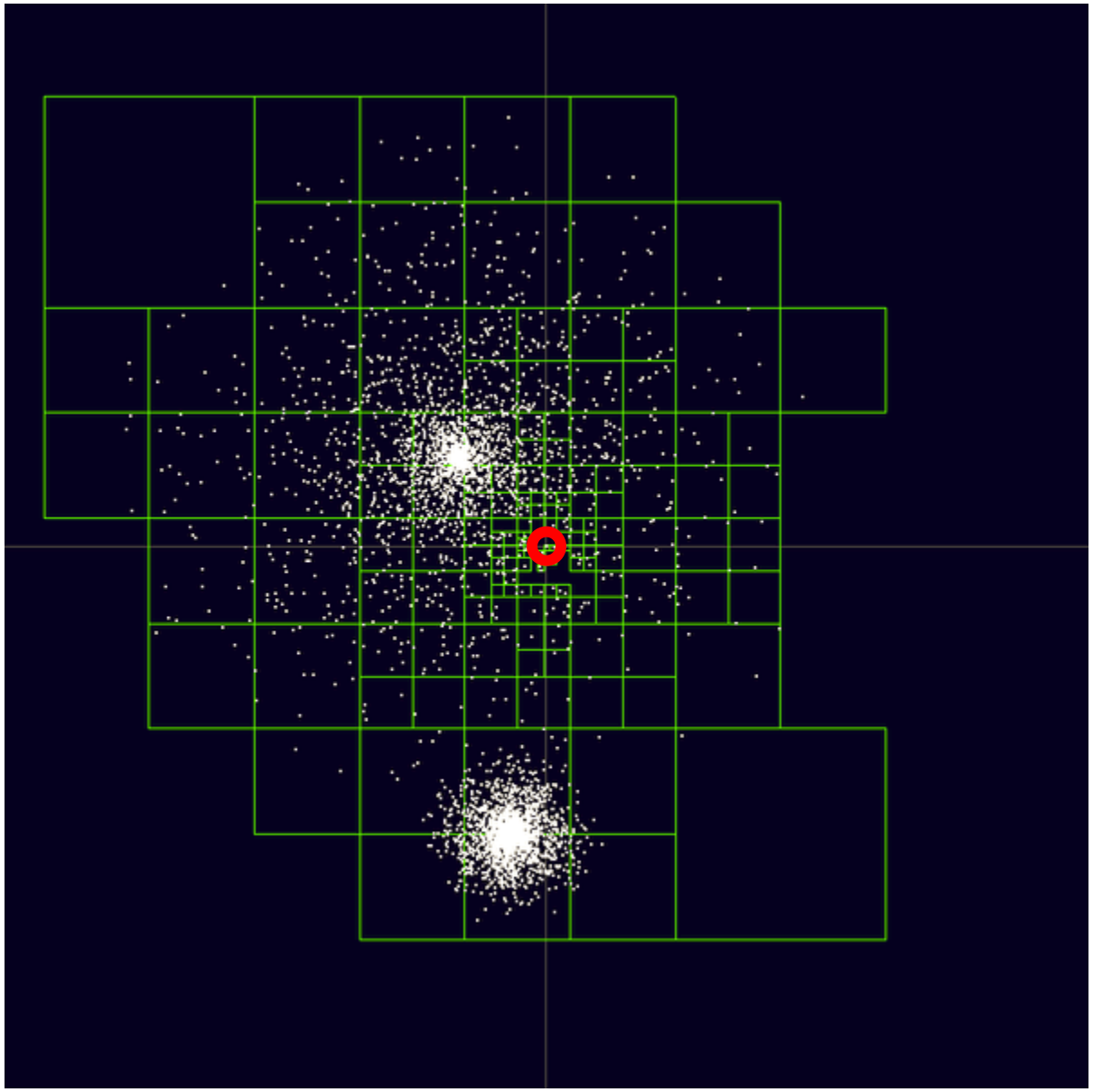
Barnes-Hut Example



Initial distribution of 5,000 bodies in 2 simulated galaxies

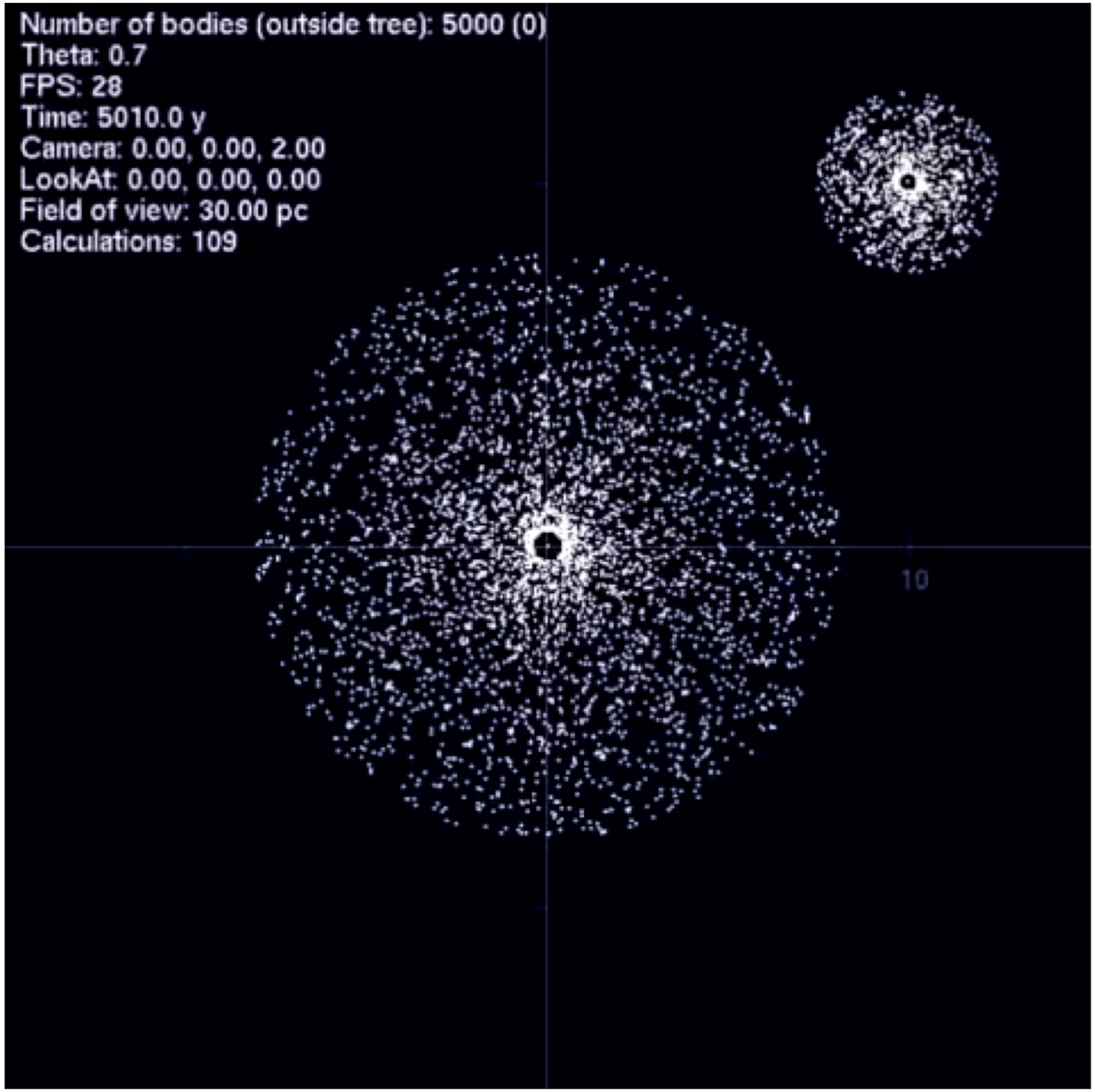
Source for this and other images and for video: Ingo Berg from Wikipedia. (http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation)

Barnes-Hut Computational Nodes



Only 146 tree nodes were actually examined in the computation for a node near the origin because of the cut-off criterion

Colliding Galaxies Video



Source:
http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation

Parallelizing the N-Body Problem

The "obvious" brute-force direct (non-B-H) algorithm costs $O(N^2)$ (for one iteration) as each of the N bodies is influenced by each of the other N - 1 bodies.

Serial B-H reduces the cost to O(N log N), but that's still too costly.

Even with parallelism it's not feasible to use the direct algorithm for most interesting *N*-body problems where *N* is very large.

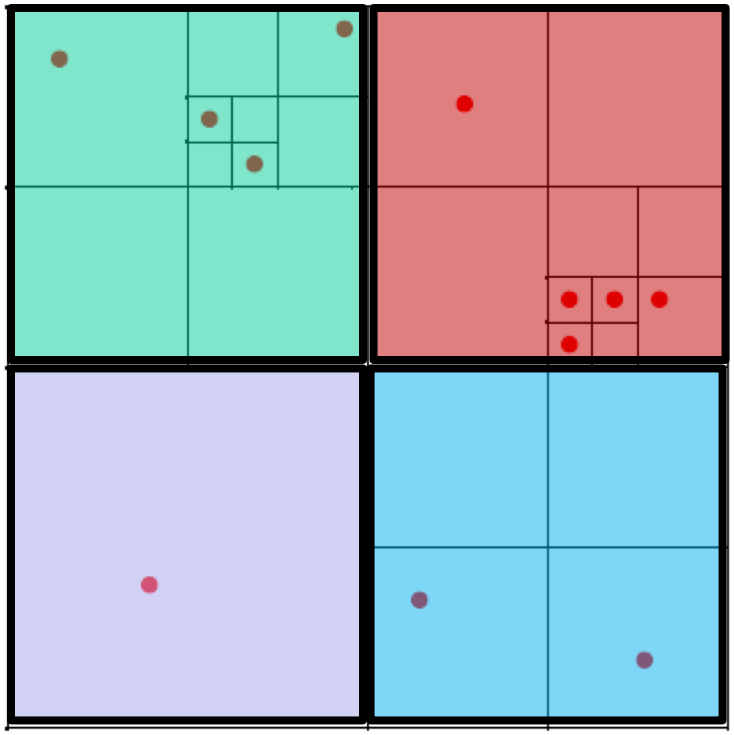
[However, variations of it make interesting parallel programming exercises.]

But parallel B-H is an interesting potential algorithm.

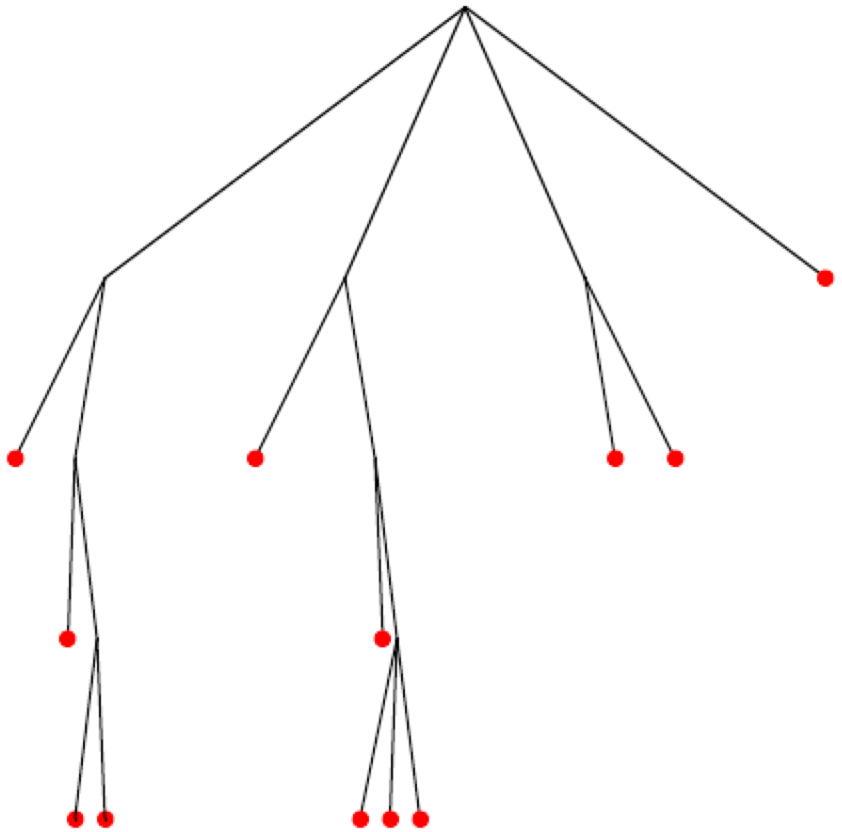
Parallelizing Barnes-Hut

- Must parallelize both major components
  - Tree construction
  - Force evaluation
- Must reduce memory requirements
  - Keep in mind that N may be very large
- Must limit communication
  - Overall computation and communication both O(N log N), though with different constants. (Because of constants, the computation will take much more time in practice.)
  - Approaches:
    - Spatial partitioning
    - Tree partitioning (not described here)

Decomposing the Domain



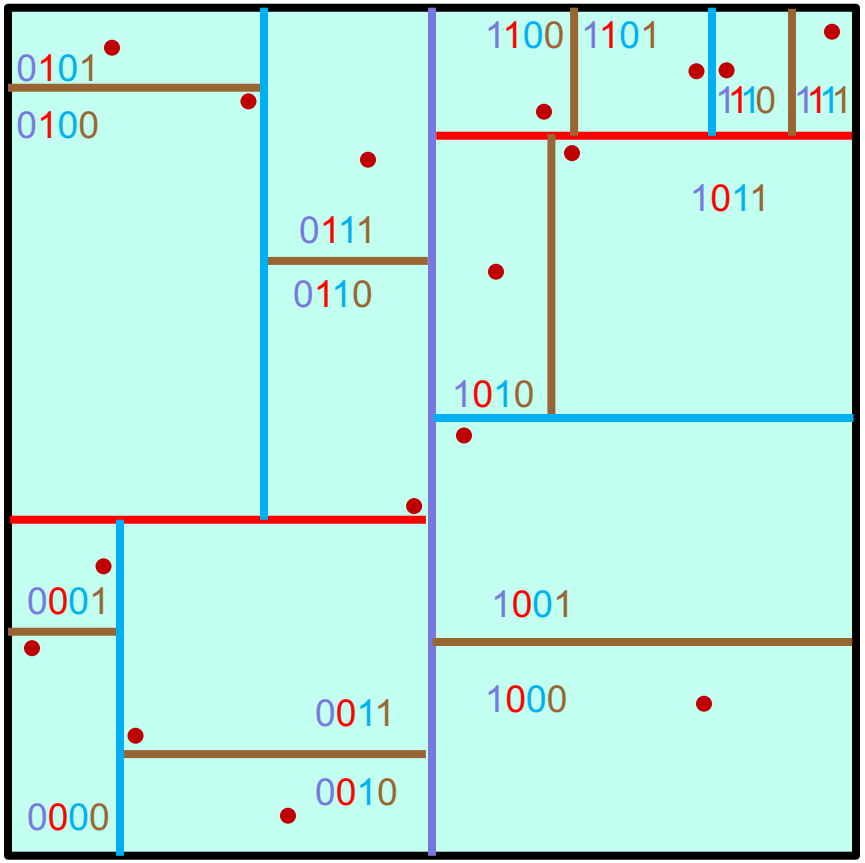
Bodies



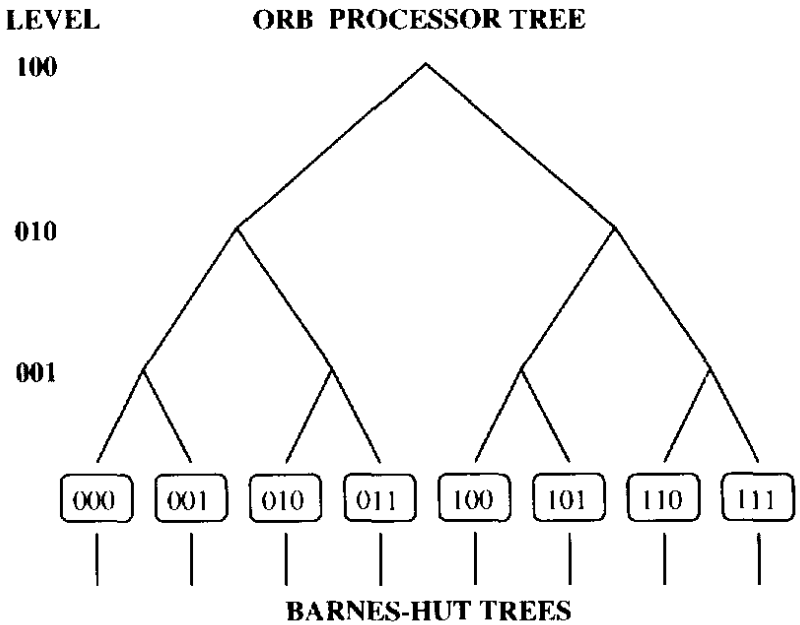
Partial quadtree

Spatial Partitioning: Orthogonal Recursive Bisection

For 2-dimensions, with 2^{k-1} procs: First, find line dividing bounding box into 2 "spherical" subboxes with ~equal numbers of bodies, labeling right box "1" and left box "0". Repeat recursively until k levels have been subdivided, extending box labels by 1 bit each time. The box labels now represent the process numbers (as binary numbers). (Note: "spherical" boxes help with accuracy.)



In general, process stops when there are p leaf subtrees. Each of p processors will then use ordinary Barnes-Hut for its subtree. For example:



From J. Dubinsky, "A Parallel Tree Code," *New Astronomy* 1 (1996) pp. 133-147.

Parallelizing Force Computation

- Very easy, in principle
  - Force at each body costs ~k log N
    - Requires using a bounded number of cells per level in the tree
  - Simple strategy: Let each process handle the N/p bodies it has after recursive orthogonal bisection
- Practical issues
  - How much of the tree is required in each process?
    - "Locally Essential Tree": Just that parts the must be opened ($d/r > \theta$)
    - Lots of communication, but can be organized into O(p log p) messages
    - Tree may take a lot of memory
  - Load balance
    - Not every body requires the same amount of computation

Locally Essential Trees

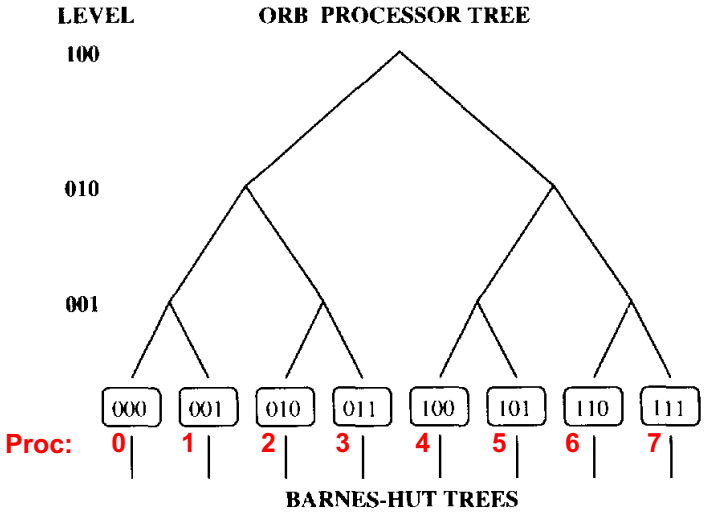
- Each process *p* needs:
  - Portion of tree containing bodies it owns
  - "Nearby" ("Essential") portions of tree from other processes
- Simplified criteria for Locally Essential Tree for process *p*
  - Let *n* be a node of the tree not containing any of *p*'s bodies; let *d(n)* be the side length for *n* (the "diameter"); and let *r(n)* be the shortest distance from any point owned by *p* to the rectangle for *n*
  - Then node *n* is in the LET for *p* if either
    - $d(n)/r(n) \geq \theta$
      - This means we need node *n* and all of its ancestors
    - $d(n)/r(n) < \theta$, but $d(pn)/r(pn) \geq \theta$ for the parent *pn* of *n*
      - This means we need node *n*, but not its children
      - Also means that node *n* is the "highest" node that we don't need to open
  - For a dividing line at any level, *p* has to determine and deliver the part of its tree needed by processors on other side of the line.

Building the LET

```
// On each processor p (2^(k-1) total procs)

For (i = 1; i<=k; i++) {

   OTHERPROC = p xor 2^(k-i); //Flip (k-i)-th bit of my proc number
                  // Node in similar position in other subtree

   Compute the subset of my LET that any processor on the
      other side of the level-i bisecting line might need;

   Send the subset to OTHERPROC;

   Receive corresponding subset from OTHERPROC and add it to
      my LET;
}

Prune my LET of unnecessary parts;
```

Material based on http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Building the LET



Algorithm starts from local B-H tree in each process and determines which parts are relevant to other processes by examining each ORB partition edge in turn. Involves significant interprocessor cooperation (log p steps involving p/2 pairwise exchanges)

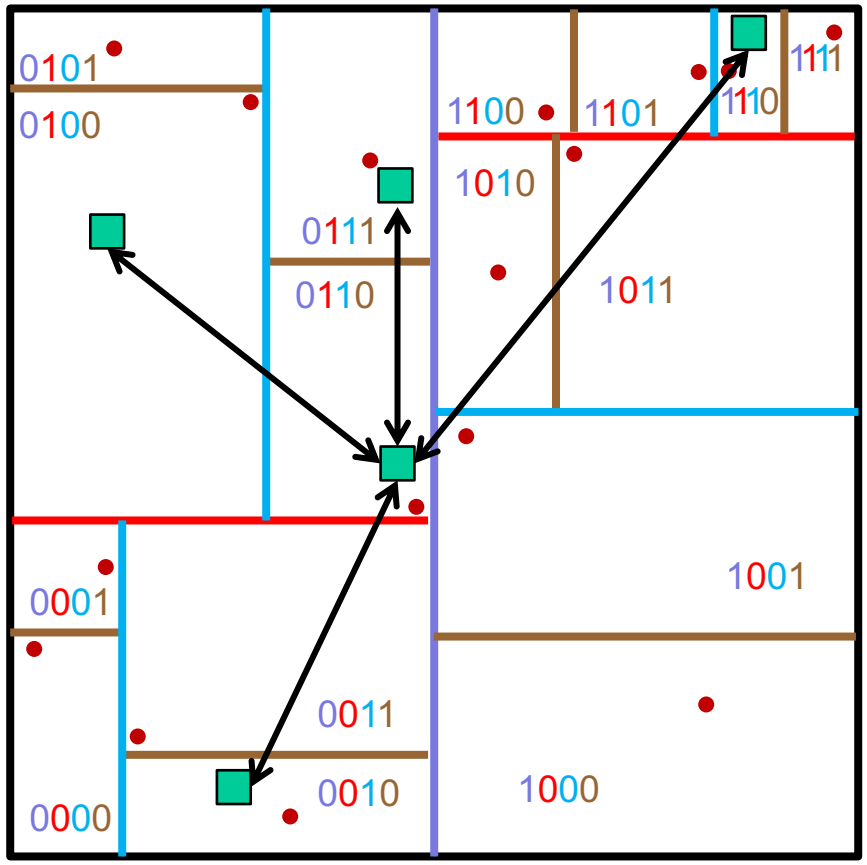
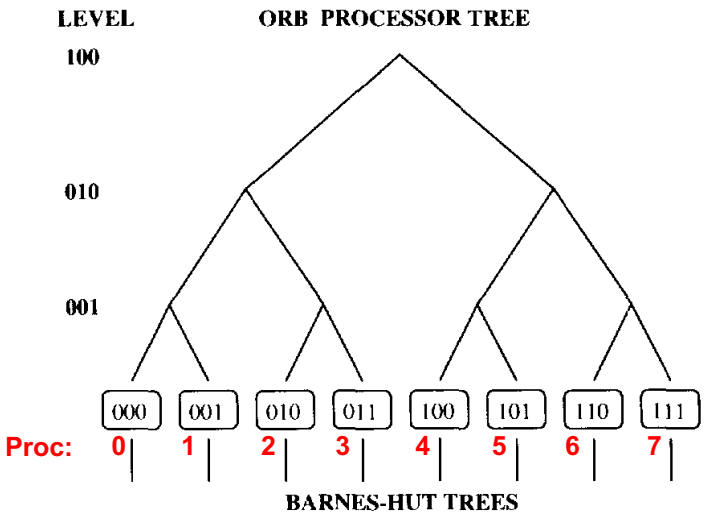


Figure from http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Building the LET



Algorithm starts from local B-H tree in each process and determines which parts are relevant to other processes by examining each ORB partition edge in turn. Involves significant interprocessor cooperation (log p steps involving p/2 pairwise exchanges)

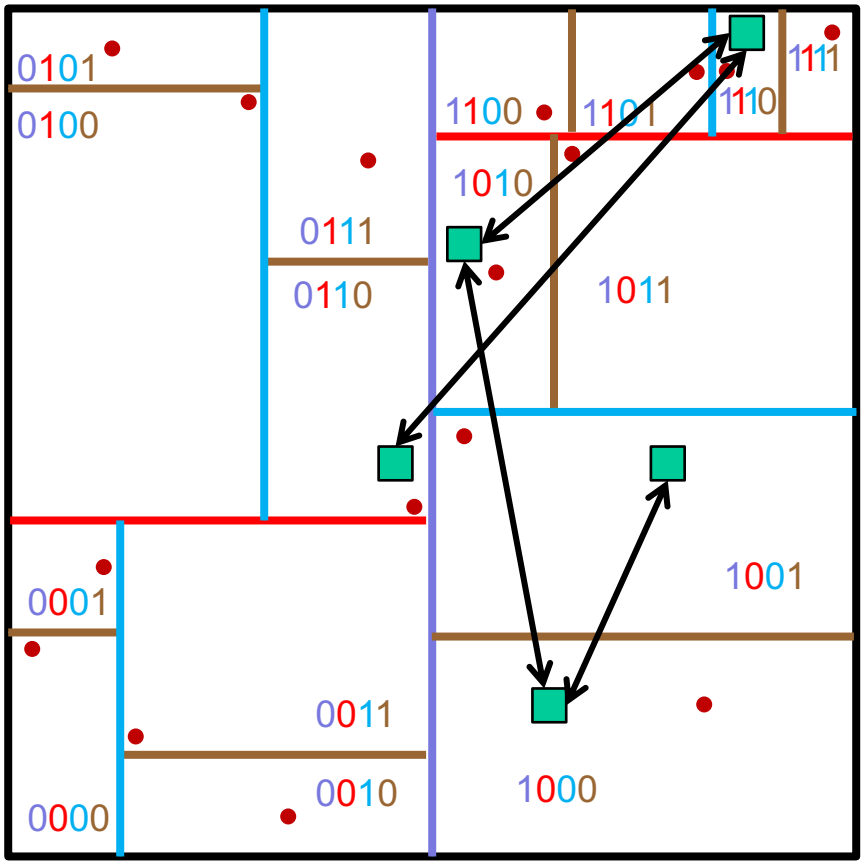
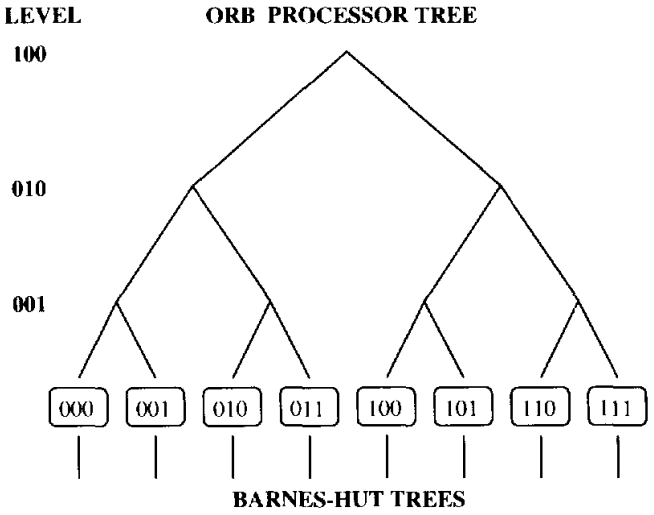


Figure from http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html

Building the LET



Algorithm starts from local B-H tree in each process and determines which parts are relevant to other processes by examining each ORB partition edge in turn. Involves significant inter-processor cooperation.

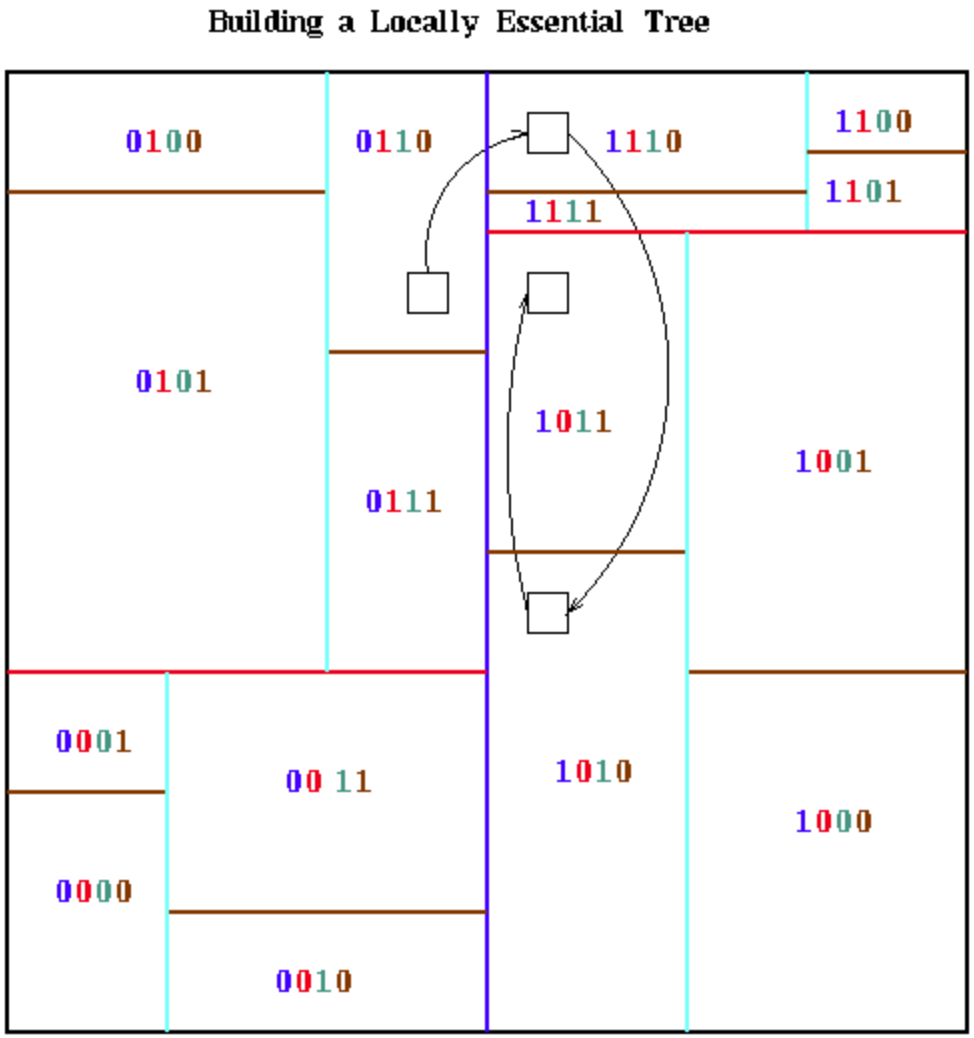


Figure from http://www.eecs.berkeley.edu/~demmel/cs267/lecture26/lecture26.html