
CS 267: Applications of Parallel Computers

Tree-Structured Codes for N-Body Simulations

Kathy Yelick

<http://www.cs.berkeley.edu/~yelick/cs267>

Motivation

- Particle methods are used for a variety of applications
- Astrophysics
 - The particles are stars or galaxies
 - The force is gravity
- Particle physics
 - The particles are ions, electrons, etc.
 - The force is due to Coulomb's Law
- Molecular dynamics
 - The particles are atoms or
 - The forces is electrostatic
- Vortex methods in fluid dynamics
 - Particles are blobs of fluid

Outline

- Motivation
 - Obvious algorithm 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
- Alternative approach: lots of hardware

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
```

- Let $f(i)$ be the force on particle i
- $f(i) = \text{external_force} + \text{nearest_neighbor_force} + \text{NBody_force}$

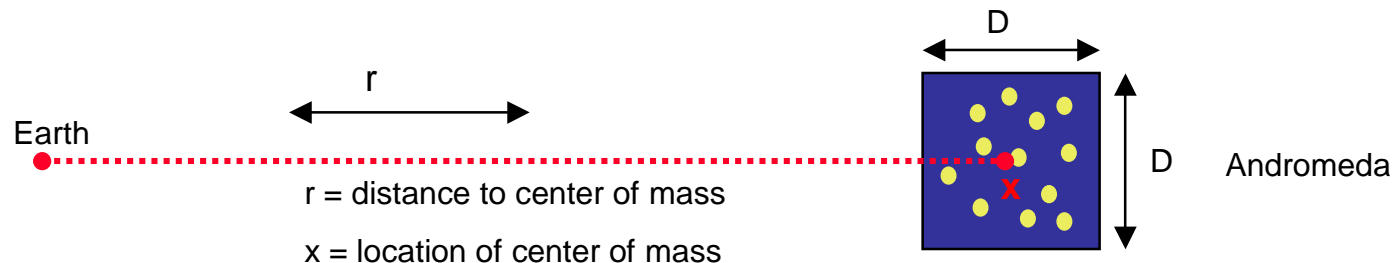
Particle Simulation

$f(i) = \text{external_force} + \text{nearest_neighbor_force} + \text{NBody_force}$

- External_force (e.g., current) is usually embarrassingly parallel, $O(N)$
- Nearest_neighbor_force is with a few neighbors, so still $O(N)$
- N-Body_force (gravity or electrostatics) requires all-to-all interactions
 - $f(i) = \sum_{k \neq i} f(i,k)$... $f(i,k) = \text{force on } i \text{ from } k$
 - $f(i,k) = c \cdot v / \|v\|^3$ in 3 dimensions or
 - $f(i,k) = c \cdot v / \|v\|^2$ in 2 dimensions
 - c = product of masses or charges and appropriate constants
 - v = vector from particle i to particle k , $\|v\|$ = length of v
 - Obvious algorithm costs $O(N^2)$, but we can do better...

Reducing the Number of Particles in the Sum

- Previous 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 \geq number of visible stars
 - One “star” is really the Andromeda galaxy, which is billions of stars
 - A lot of work if we compute this per star ...
- 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”

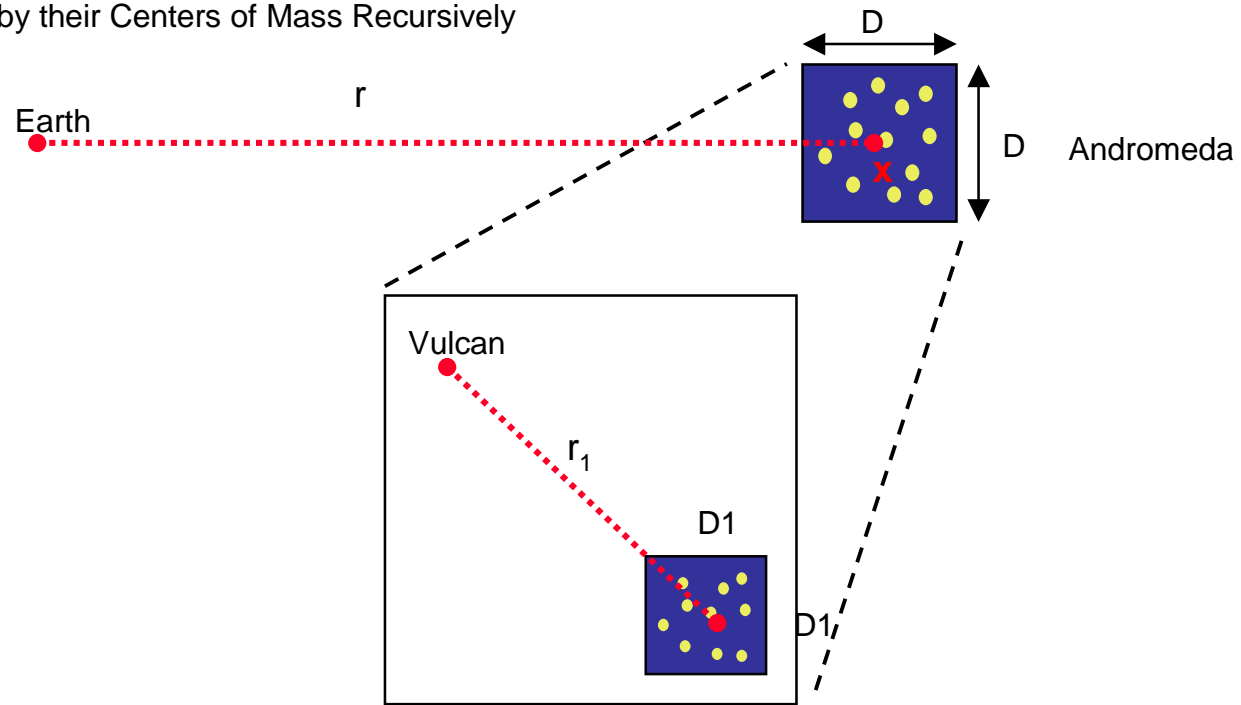


- Idea not new: Newton approximated earth and falling apple by CMs

What is new: Using points at CM *Recursively*

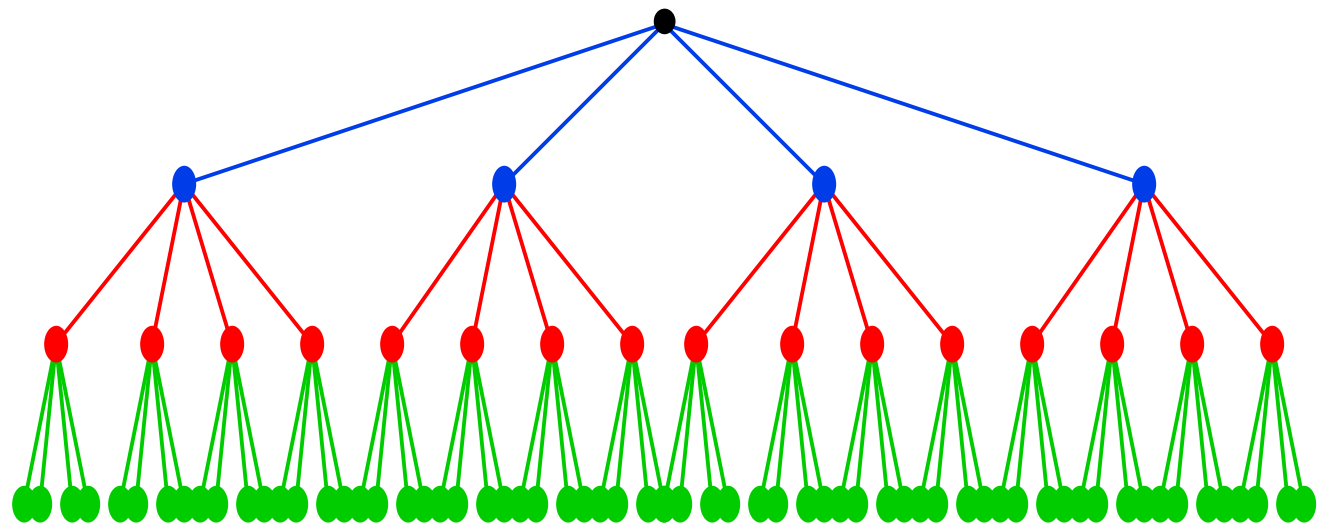
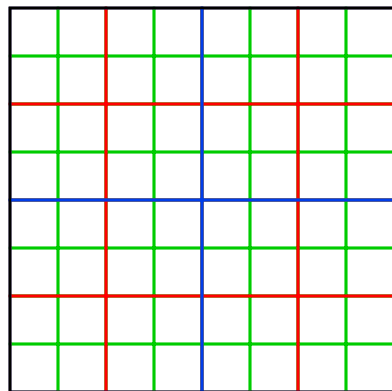
- From Andromeda's point of view, Milky Way is also a point mass
- Within Andromeda, picture repeats itself
 - As long as D_1/r_1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
 - Boxes nest in boxes recursively

Replacing clusters by their Centers of Mass Recursively



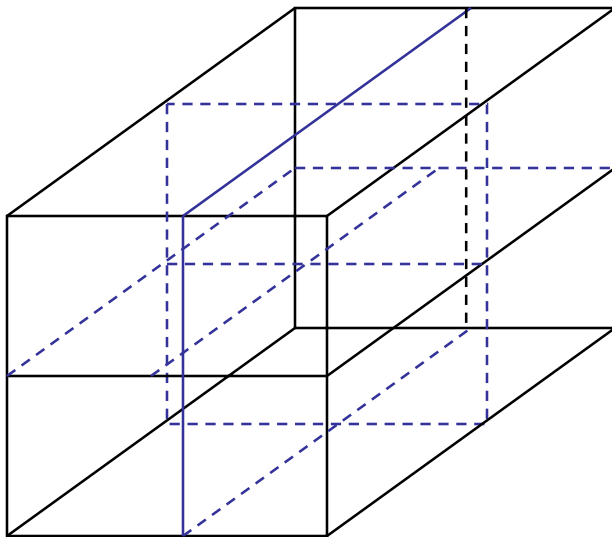
Quad Trees

- 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

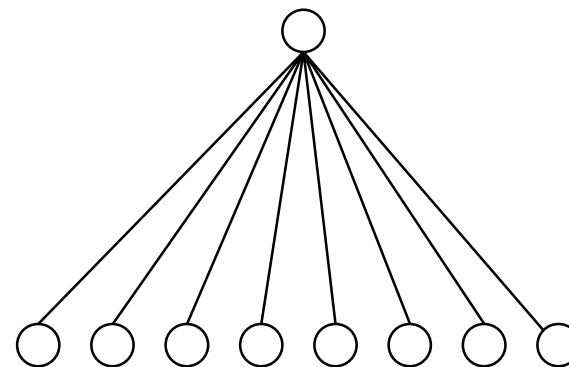


Oct Trees

- Similar Data Structure to subdivide 3D space
- Analogous to 2D Quad tree--each cube is divided into 8 sub-cubes



Two Levels of an OctTree

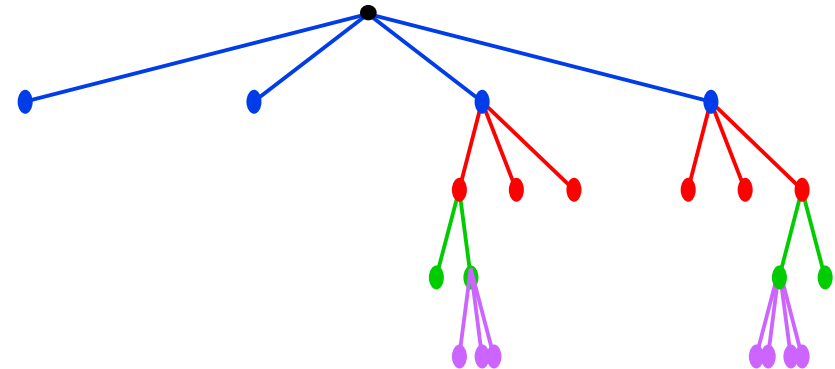
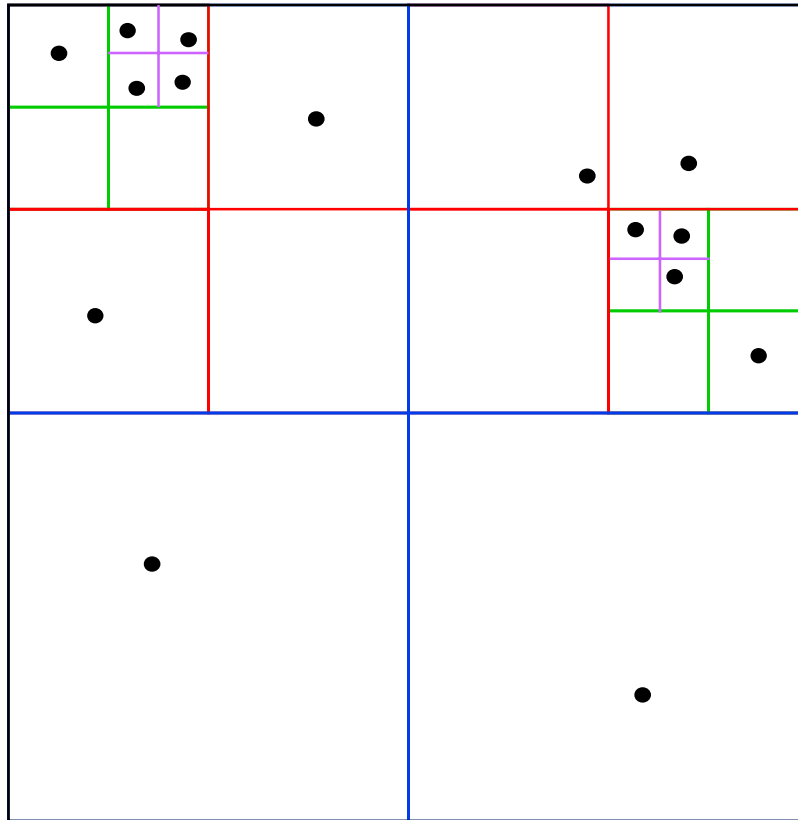


Using Quad Trees and Oct Trees

- All our algorithms begin by constructing a tree to hold all the particles
- Interesting cases have non-uniform particle distribution
 - In a complete tree (full at lowest level), most nodes would be empty, a waste of space and time
- **Adaptive** Quad (Oct) Tree only subdivides space where particles are located
 - More compact and efficient computationally, but harder to program

Example of an Adaptive Quad Tree

Adaptive quad tree where no space contains more than 1 particle



Child nodes enumerated counterclockwise from SW corner
Empty ones excluded

Adaptive Quad Tree Algorithm

class QuadTree

build (particles)

 QuadTree t = new QuadTree();

 for each j in particles t.insert(j)

 t.insert(j)

 end build

... loop over all N particles

... insert particle j in QuadTree

insert(j)

... Try to insert particle j at node n in QuadTree

 if this node is empty

... empty

 add j as the (only) particle in this node

 if this is an internal node (has 4 children)

... internal

 determine which child c contains particle j

 c.insert(j)

 else (this quadtree contains 1 particle)

... leaf

 add n's 4 children to the QuadTree

 let c be the child of the particle k already here

 c.insert(k)

 let c be the child of n containing j

 c.insert(j)

 end insert

Cost of Adaptive QuadTree Construction

Cost $\leq N * \text{maximum cost of QuadTreeInsert}$
= $O(N * \text{maximum depth of QuadTree})$

1. Uniform distribution \Rightarrow

depth of QuadTree = $O(\log N)$, so
Cost = $O(N \log N)$

2. Arbitrary distribution \Rightarrow

depth of Quad Tree = $O(b)$,
where $b = \text{\#bits in particle coordinates}$, 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), plus many later papers.

- Good for low accuracy calculations:

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

(other measures better if some true $f(k) \sim 0$)

- High Level Algorithm (in 2D, for simplicity)

- 1) Build the QuadTree using QuadTree.build
... 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 $O(bN)$

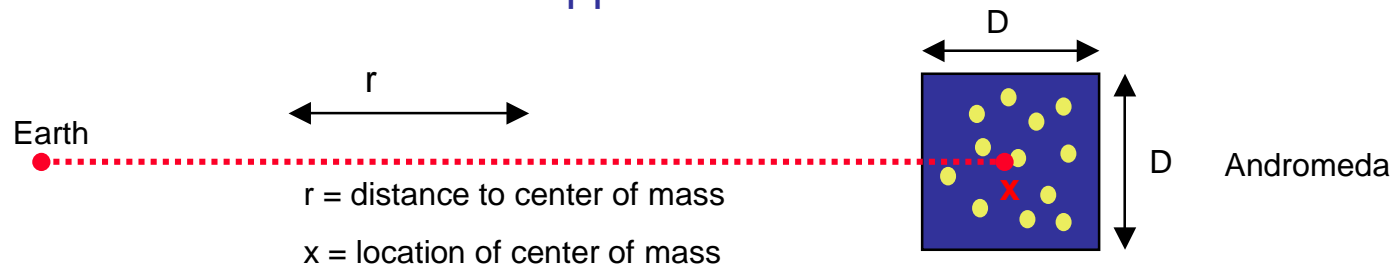
Step 2: Compute CM and TM of Each Node

```
... Compute the CM = Center of Mass and TM = Total Mass
... of all the particles in each node of the QuadTree
computeMass( ) ... compute the CM and TM of node n
  if this node contains 1 particle, p
    ... the TM and CM are identical to the particle's mass and location
    TM = p.mass; CM = p.position
  else ... "post order traversal": process parent after all children
    TM = 0; CM = 0
    for all non-empty children c ... between 1 and 4 children
      c.computeMass()
      TM += c.TM ... TM is the sum of the children's
      CM += c.TM*c.CM
    endfor
    CM = CM/ TM ... CM is the mass-weighted sum of children's CMs
end computeMass
```

$$\begin{aligned}\text{Cost} &= O(\# \text{ nodes in QuadTree}) \\ &= O(N)\end{aligned}$$

Step 3: Compute Force on Each Particle

- For each node, 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 enough away” 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 $D/r < \theta$



Computing Force on a Particle Due to a Node

- Use example of Gravity ($1/r^2$)
- Given node n and particle k , satisfying $D/r < \theta$
 - 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 $k \sim$

$$G * m * TM * \frac{x_{CM} - x_k}{r^3} \quad \frac{y_{CM} - y_k}{r^3} \quad \frac{z_{CM} - z_k}{r^3}$$

Details of Step 3 of BH

... for each particle, traverse the QuadTree to compute the force on it
for each k in particles

$f[k] = t.\text{treeForce}(k)$

... compute force on k due to all particles in tree t

f treeForce(k)

... compute force on particle k due to all particles inside node n

$f = 0$

if this node contains one particle ... evaluate directly

$f =$ force computed using formula on last slide

else

$r =$ distance from particle k to CM of this node

$D =$ size of this node

if $D/r < \theta$... 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

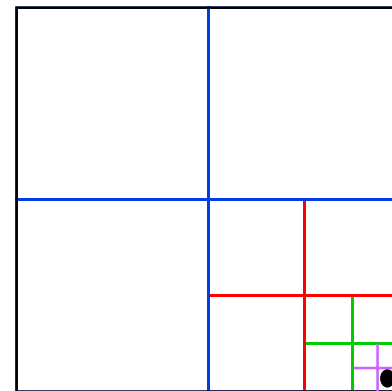
$f = f + c.\text{TreeForce}(k)$

end treeForce

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 $t.\text{treeForce}(k) = O(\text{depth in } t \text{ of leaf containing } k)$
 - Proof by Example (for $\theta > 1$):
 - For each undivided node, (except one containing k), $D/r < 1 < \theta$
 - There are 3 nodes at each level of the QuadTree
 - There is $O(1)$ work per node
 - Cost = $O(\text{level of } k)$
 - Total cost = $O(\sum_k \text{level of } k) = O(N \log N)$
 - Strongly depends on θ

Sample BH calculation,
assuming $\theta > 1$



Fast Multiple Method (FMM)

- “A fast algorithm for particle simulation”, L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, plus 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.

Fast Multiple Method (FMM)

- 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
$$\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2} + \frac{d^2\phi}{dz^2} = 0$$
- Force = $-\text{grad } \phi(x,y,z) = -(\frac{d\phi}{dx}, \frac{d\phi}{dy}, \frac{d\phi}{dz})$
- FMM will compute a compact expression 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 $z = x + iy$ (complex number)
 - potential = $\log |z|$
 - potential satisfies 2D Poisson equation
$$\frac{d^2\phi}{dx^2} + \frac{d^2\phi}{dy^2} = 0$$
 - equivalent to gravity between “infinite parallel wires” instead of point masses

2D Multipole (Taylor Expansion in 1/z)

$$\begin{aligned}
 \phi(z) &= \text{potential due to } z_k, k=1, \dots, n \\
 &= \sum_k m_k * \log |z - z_k| \quad \dots \text{ sum potential over all particles} \\
 &= \text{Real}(\sum_k m_k * \log (z - z_k)) \\
 &\quad \dots \text{ since } \log z = \log |z|e^{i\theta} = \log |z| + i\theta \\
 &\quad \dots \text{ drop Real() from now on} \\
 &= \sum_k m_k * [\log(z) + \log (1 - z_k/z)] \\
 &\quad \dots \text{ how logarithms work} \\
 &= M * \log(z) + \sum_k m_k * \log (1 - z_k/z) \\
 &\quad \dots \text{ where } M = \sum_k m_k \\
 &= M * \log(z) + \sum_k m_k * \sum_{e \geq 1} (z_k/z)^e \\
 &\quad \dots \text{ Taylor expansion converges if } |z_k/z| < 1 \\
 &= M * \log(z) + \sum_{e \geq 1} z^{-e} \sum_k m_k z_k^e \\
 &\quad \dots \text{ swap order of summation} \\
 &= M * \log(z) + \sum_{e \geq 1} z^{-e} \alpha_e \\
 &\quad \dots \text{ where } \alpha_e = \sum_k m_k z_k^e \\
 &= M * \log(z) + \sum_{r \geq e \geq 1} z^{-e} \alpha_e + \text{error}(r) \\
 &\quad \dots \text{ where error}(r) = \sum_{r < e} z^{-e} \alpha_e
 \end{aligned}$$

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

Parallelizing Hierarchical N-Body Codes

- Barnes-Hut, FMM and related algorithm 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

Parallelizing Hierarchical N-Body Codes

- One parallelization scheme will work for them all
 - Assign regions of space to each processor
 - Regions may have different shapes, to get 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
 - Top of tree stored by all processors, lower 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
 - (Description based on SC97 paper by D. Blackston and T. Suel)

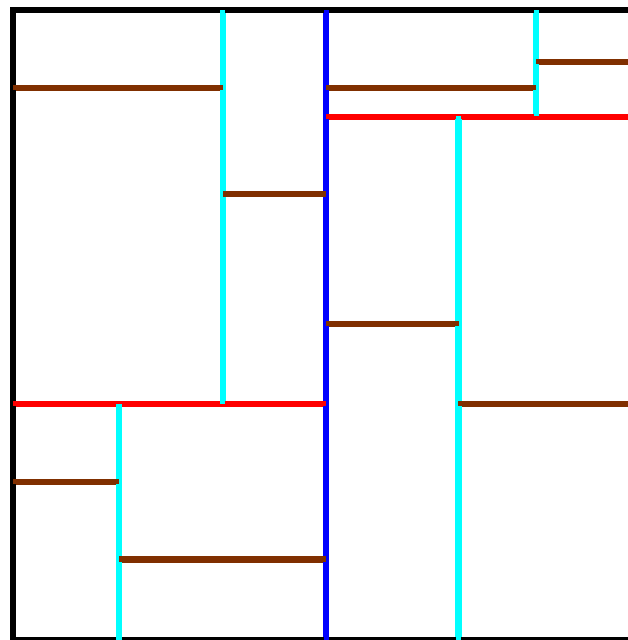
Programming Model in PBody

- BSP Model = Bulk Synchronous Programming Model
 - All processors compute; barrier; all processors communicate; barrier; repeat
 - Common style in MPI and other SPMD models
- Advantages and Disadvantage
 - + easy to program
 - + easy to port (MPI, shared memory, TCP network)
 - Rigidly synchronous style might mean inefficiency?
- Summary of performance results in PBody
 - FMM 80% efficient on 32 processor Cray T3E
 - FMM 90% efficient on 4 PCs on slow network
 - FMM 85% efficient on 16 processor SGI SMP (Power Challenge)
 - Better efficiencies for Barnes-Hut, other algorithms

Load Balancing 1: ORB

- Orthogonal Recursive Bisection (ORB)
 - Warren and Salmon, Supercomputing 92
 - Recursively split region along axes into regions containing equal numbers of particles
 - Works well for 2D, not 3D (available in Pbody)
Orthogonal Recursive Bisection

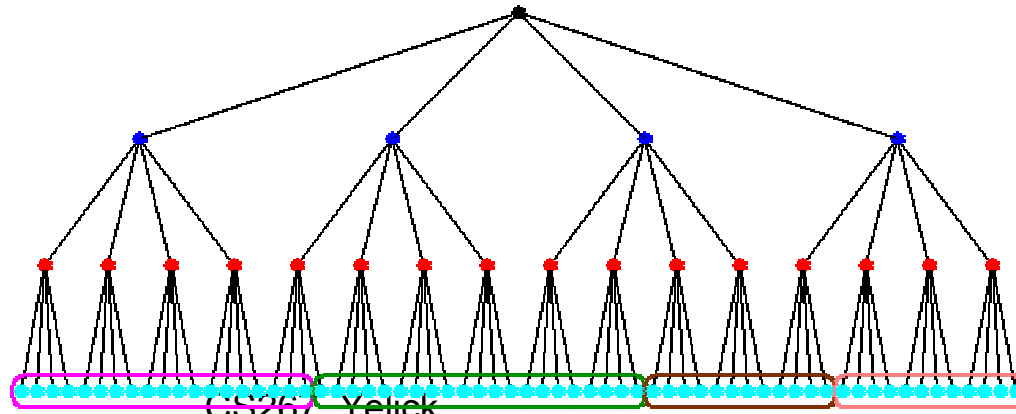
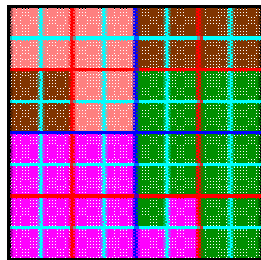
**Partitioning
for 16 procs:**



Load Balancing 2: Costzones

- Called Costzones for Shared Memory
 - PhD thesis, J.P. Singh, Stanford, 1993
- Called “Hashed Oct Tree” for Distributed Memory
 - Warren and Salmon, Supercomputing 93
- We will use the name Costzones for both; also in Pbody
- 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 blocks of nodes with work W/p to processors
 - Works well in 3D

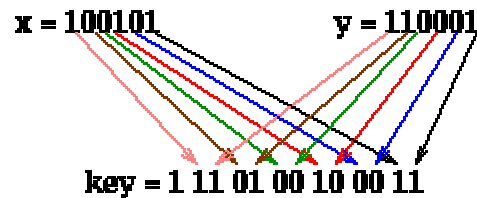
Using costzones to layout a quadtree on 4 processors
Leaves are color coded by processor color



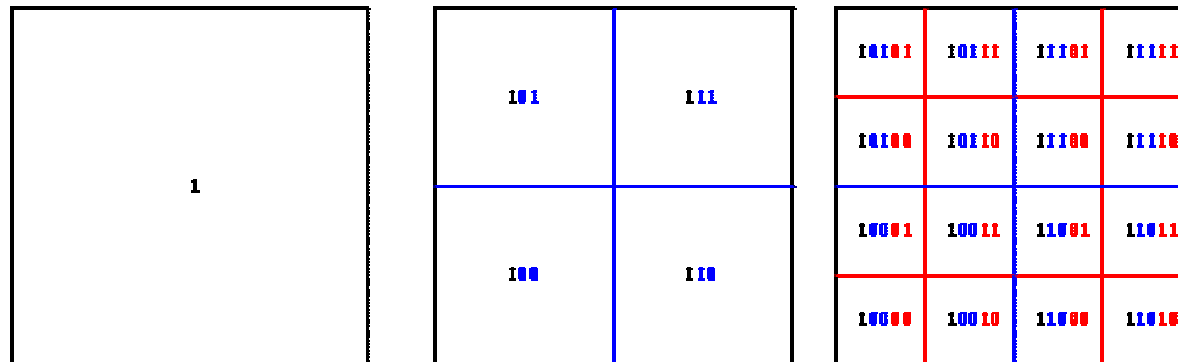
Linearly Ordering Nodes for Costzones

- Hashed QuadTrees (Warren and Salmon)
- Assign unique key to each node in QuadTree, then compute hash(key) to get integers that can be linearly ordered
- If (x,y) are coordinates of center of node, interleave bits to get key
 - Put 1 at left as “sentinel”
 - Nodes at root of tree have shorter keys

Building a key for a hashed Quadtree



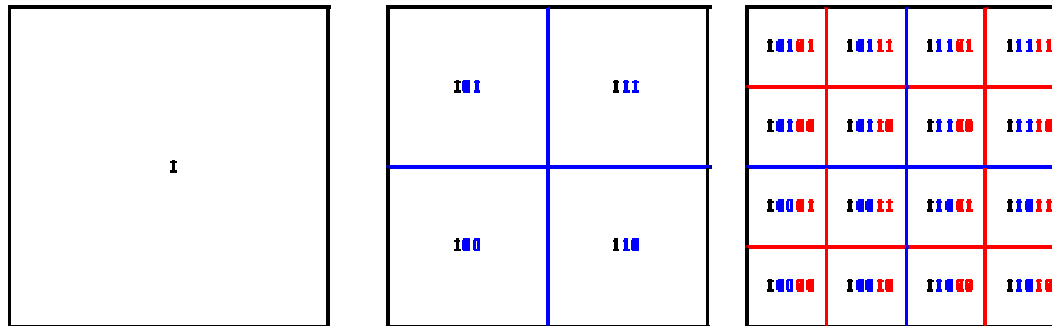
Assigning Keys to Quadtree Nodes



Linearly Ordering Nodes for Costzones

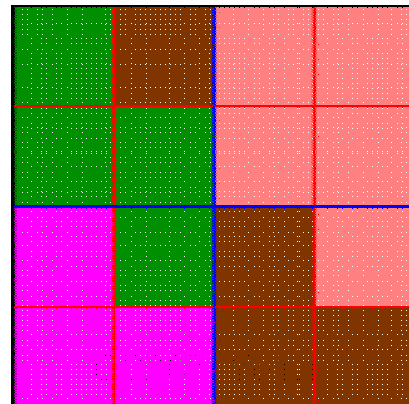
- Assign unique key to each node in QuadTree, then compute hash(key) to get a linear order
- key = interleaved bits of x,y coordinates of node, prefixed by 1

Assigning Keys to Quadtree Nodes



- Hash(key) = bottom h bits of key (eg h=4)
- Assign contiguous blocks of hash(key) to same processors

Assigning Hash Table Entries to 4 Processors



Determining Costzones in Parallel

- Not practical to compute QuadTree, in order to compute Costzones, to then determine how to best build QuadTree
- Random Sampling:
 - All processors send small random sample of their particles to Proc 1
 - Proc 1 builds small Quadtree serially, determines its Costzones, and broadcasts them to all processors
 - Other processors build part of Quadtree they are assigned by these Costzones
- All processors know all Costzones; we need this later to compute LETs

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

Locally Essential Trees in Distributed Memory

- How does each processor decide which other processors need parts of its local subset of the Quadtree?
- Barnes-Hut:
 - Let j and k be processors, n a node on processor j
 - Let $D(n)$ be the side length of n
 - Let $r(n)$ be the shortest distance from n to any point owned by k
 - If either
 - (1) $D(n)/r(n) < \theta$ and $D(\text{parent}(n))/r(\text{parent}(n)) \geq \theta$, or
 - (2) $D(n)/r(n) \geq \theta$then node n is part of k 's LET, and so proc j should send n to k
 - Condition (1) means (TM,CM) of n can be used on proc k , but this is not true of any ancestor
 - Condition (2) means that we need the ancestors of type (1) nodes too
- FMM
 - Simpler rules based just on relative positions in QuadTree

Applications of Fast N-Body Algorithms

- 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)

Performance Results - 1

- 512 Proc Intel Delta
 - Warren and Salmon, Supercomputing 92
 - 8.8 M particles, uniformly distributed
 - .1% to 1% RMS error
 - 114 seconds = 5.8 Gflops
 - Decomposing domain 7 secs
 - Building the OctTree 7 secs
 - Tree Traversal 33 secs
 - Communication during traversal 6 secs
 - Force evaluation 54 secs
 - Load imbalance 7 secs
 - Rises to 160 secs as distribution becomes nonuniform

Performance Results - 2

- Cray T3E
 - Blackston, 1999
 - 10^{-4} RMS error
 - General 80% efficient on up to 32 processors
 - Example: 50K particles, both uniform and non-uniform

	Uniform		Nonuniform	
	<u>1 proc</u>	<u>4 procs</u>	<u>1 proc</u>	<u>4 procs</u>
Tree size	2745	2745	5729	5729
MaxDepth	4	4	10	10
Time(secs)	172.4	38.9	14.7	2.4
Speedup		4.4		6.1
Speedup vs $O(n^2)$		>50		>500

Alternate Approach: Hardware

Grape-6 System

- The 6th generation of GRAPE (Gravity Pipe) Project
- Gravity calculation for many particles with 31 Gflops/chip
- 32 chips / board 0.99 Tflops/board
- 64 boards of full system is installed in University of Tokyo 63 Tflops
- On each board, all particles data are set onto SRAM memory, and each target particle data is injected into the pipeline, then acceleration data is calculated
- Gordon Bell Prize at SC2000, SC2001, and lifetime in SC2002 (Prof. Makino, U. Tokyo)

