

---

# **CS 267 Applications of Parallel Computers**

## **Hierarchical Methods for the N-Body problem**

**James Demmel**

**[www.cs.berkeley.edu/~demmel](http://www.cs.berkeley.edu/~demmel)**

## **Big Idea**

- Suppose the answer at each point depends on data at all the other points
  - Electrostatic, gravitational force
  - Solution of elliptic PDEs
  - Graph partitioning
- Seems to require at least  $O(n^2)$  work, communication
- If the dependence on “distant” data can be compressed
  - Because it gets smaller, smoother, simpler...
- Then by compressing data of groups of nearby points, can cut cost (work, communication) at distant points
  - Apply idea recursively: cost drops to  $O(n \log n)$  or even  $O(n)$
- Examples:
  - Barnes-Hut or Fast Multipole Method (FMM) for electrostatics/gravity/...
  - Multigrid for elliptic PDE
  - Multilevel graph partitioning (METIS, Chaco,...)

# Outline

---

- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force 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 related algorithms**

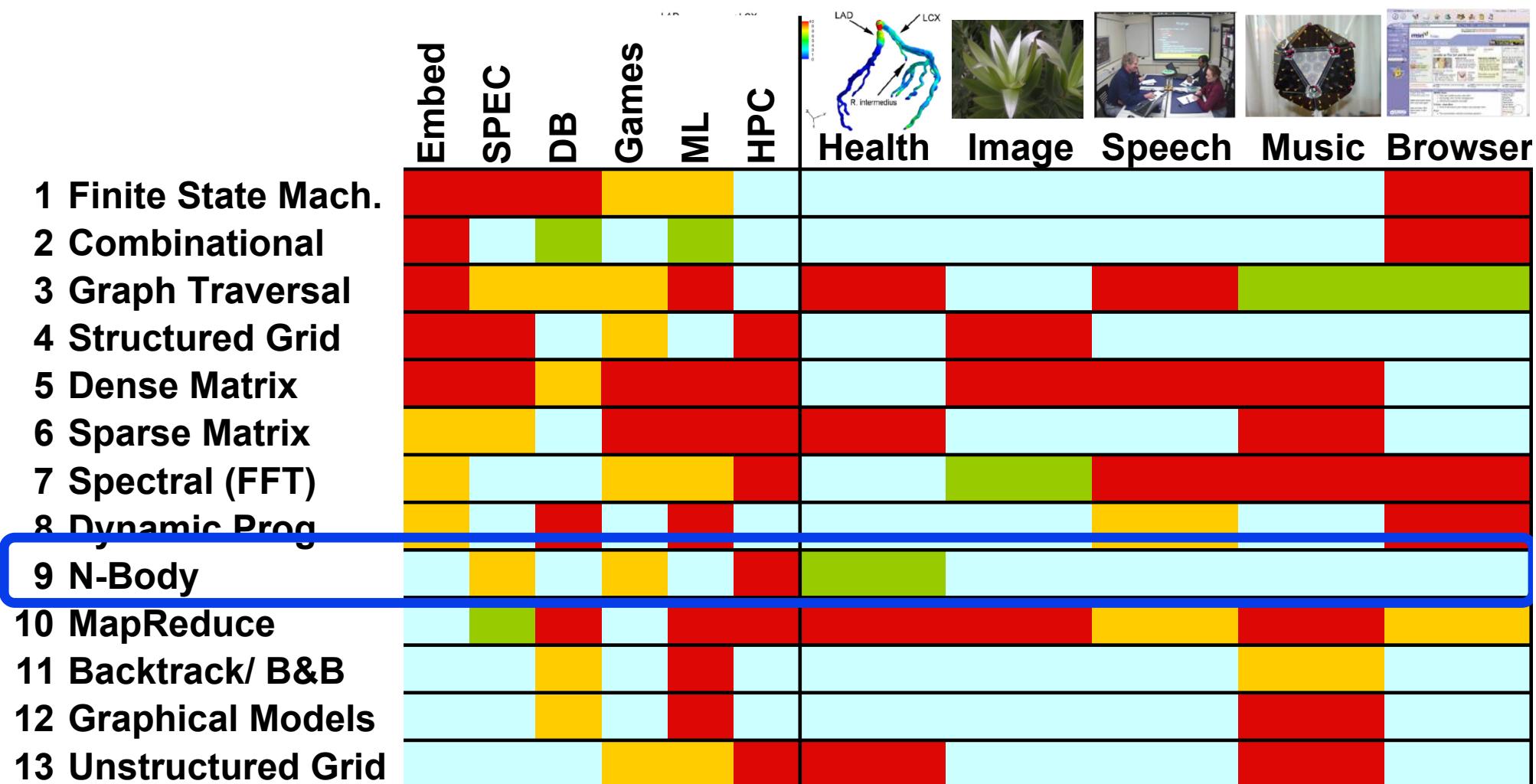
# Particle Simulation

```
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
```

- ° **f(i) = external\_force + nearest\_neighbor\_force + N-Body\_force**
  - External\_force is usually embarrassingly parallel and costs O(N) for all particles
    - external current in Sharks and Fish
  - Nearest\_neighbor\_force requires interacting with a few neighbors, so still O(N)
    - van der Waals, bouncing balls (HW2)
  - N-Body\_force (gravity or electrostatics) requires all-to-all interactions
    - $f(i) = \sum_{k \neq i} f(i,k)$  ... **f(i,k) = force on i from k**
    - $f(i,k) = c * v / \|v\|^3$  in 3 dimensions or  $f(i,k) = c * v / \|v\|^2$  in 2 dimensions
      - $v$  = vector from particle i to particle k ,  $c$  = product of masses or charges
      - $\|v\|$  = length of  $v$
    - **Obvious algorithm costs O(n<sup>2</sup>)**, but we can do better...

# What do commercial and CSE applications have in common?

## Motif/Dwarf: Common Computational Methods (Red Hot → Blue Cool)



## Applications (1/2)

---

- ° **Astrophysics and Celestial Mechanics - 1992**

- 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

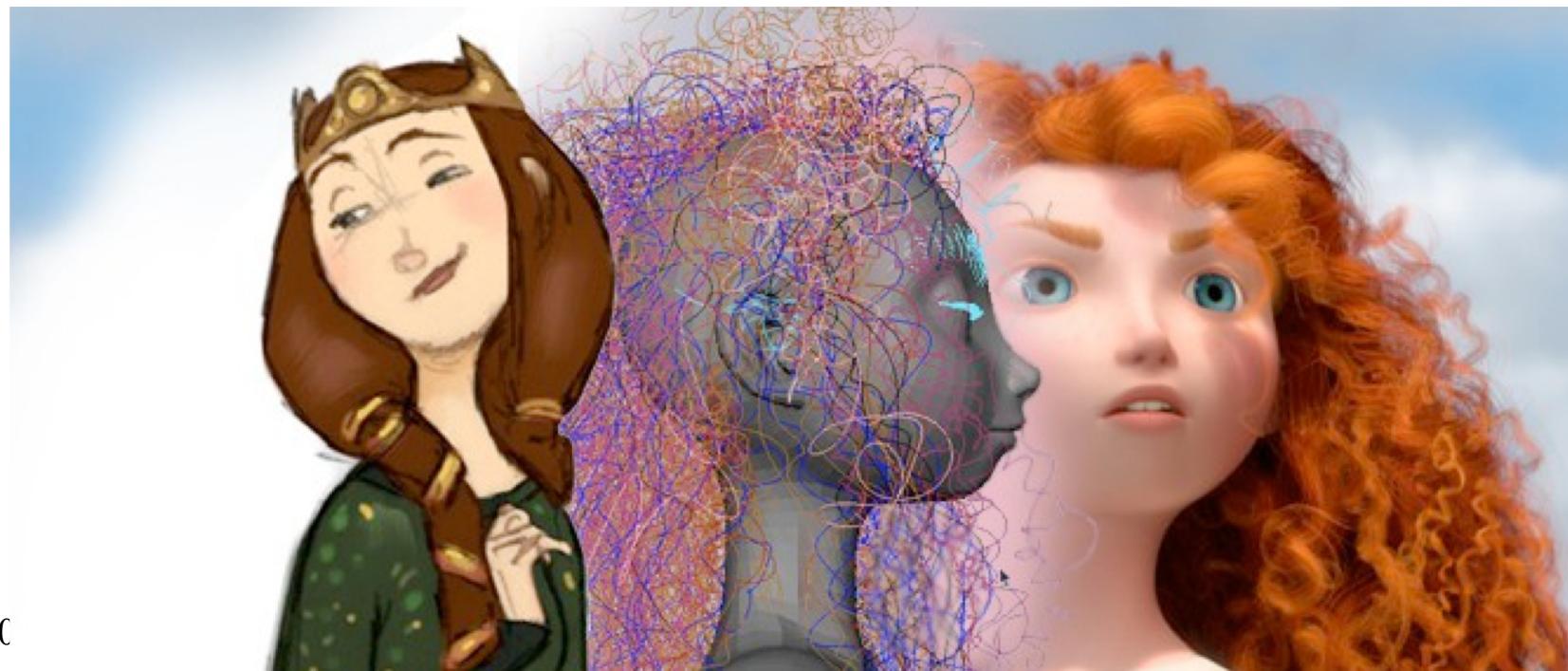
- ° **Vortex particle simulation of turbulence – 2009**

- Cluster of 256 NVIDIA GeForce 8800 GPUs
- **16.8 million particles**
  - T. Hamada, R. Yokota, K. Nitadori, T. Narumi, K. Yasuki et al
  - Gordon Bell Prize for Price/Performance at Supercomputing **2009**
- Sustained **20 Teraflops, or \$8/Gigaflop**

## Applications (2/2)

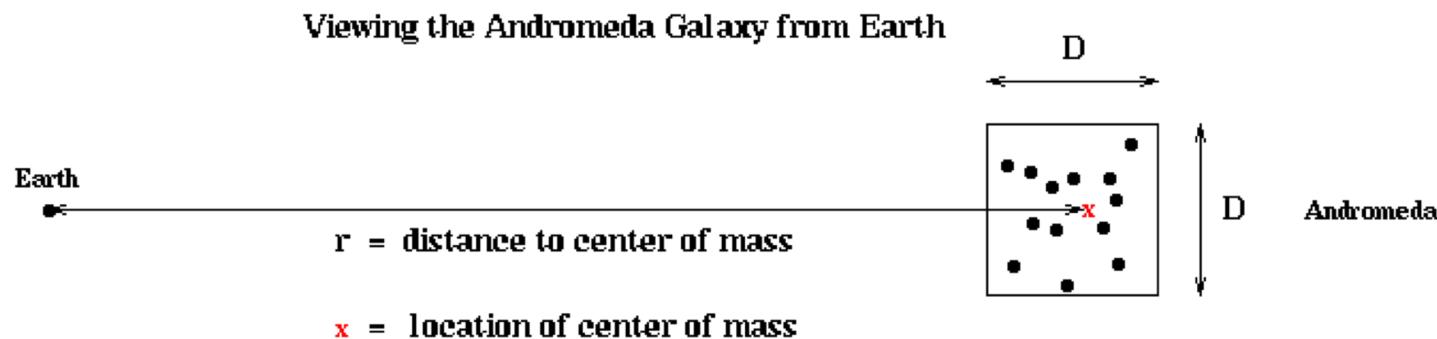
---

- **Molecular Dynamics**
- **Plasma Simulation**
- **Electron-Beam Lithography Device Simulation**
- **Hair ...**
  - [www.fxguide.com/featured/brave-new-hair/](http://www.fxguide.com/featured/brave-new-hair/)
  - [graphics.pixar.com/library/CurlyHairA/paper.pdf](http://graphics.pixar.com/library/CurlyHairA/paper.pdf)



# Reducing the number of particles in the force sum

- ° All later 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
  - Oops! One “star” is really the Andromeda galaxy, which contains billions of real stars
    - Seems like a lot more work than we thought ...
- ° Don't worry, ok to approximate all stars in Andromeda by a single point at its center of mass (CM) with same total mass (TM)
  - D = size of box containing Andromeda , r = distance of CM to Earth
  - Require that  $D/r$  be “small enough”

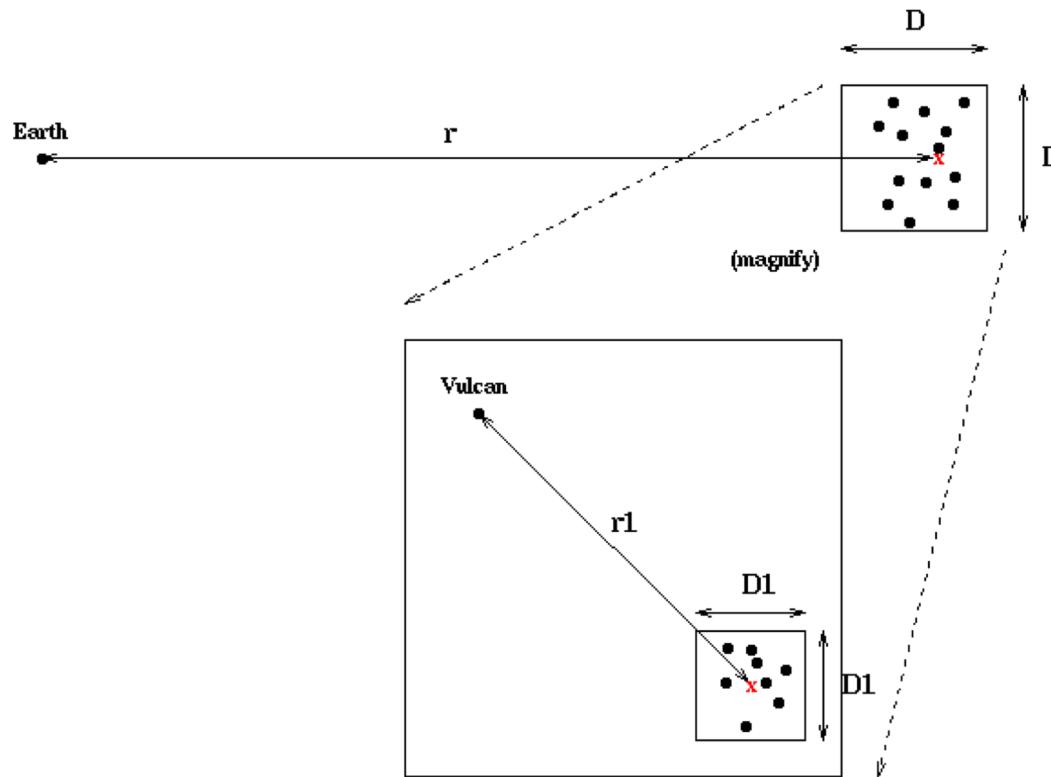


- Idea not new: Newton approximated earth and falling apple by CMs

## 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



# Outline

---

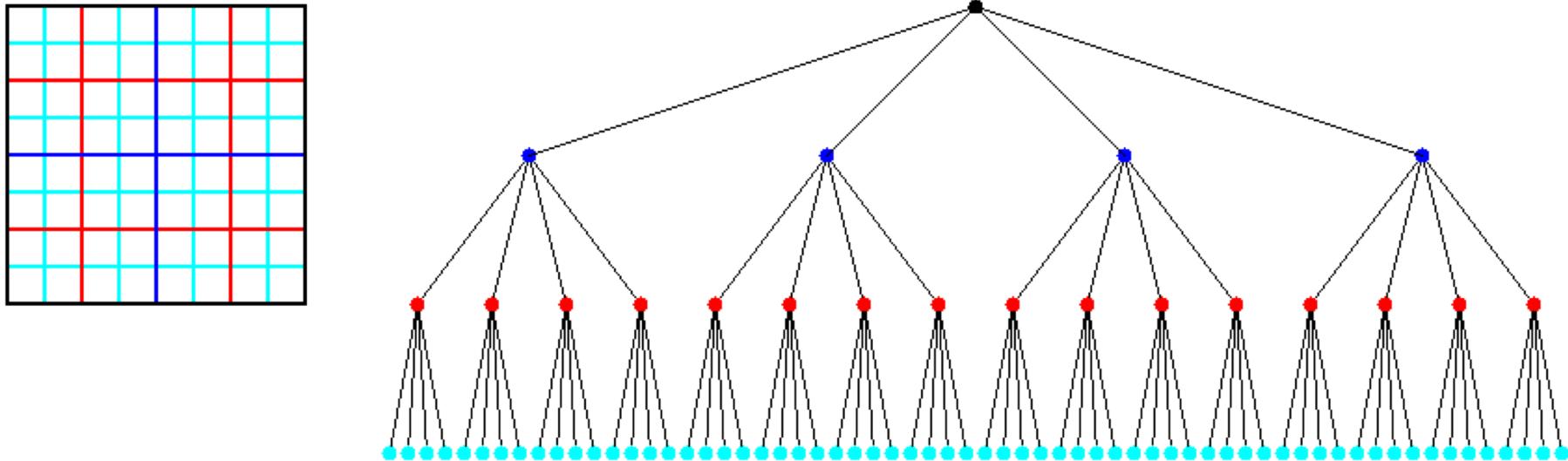
- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force on N bodies takes  $O(N^2)$  work
- **How to reduce the number of particles in the force sum**
  - We must settle for an approximate answer (say 2 decimal digits, or perhaps 16 ...)
- **Basic Data Structures: Quad 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 related algorithms**

# 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

A Complete Quadtree with 4 Levels

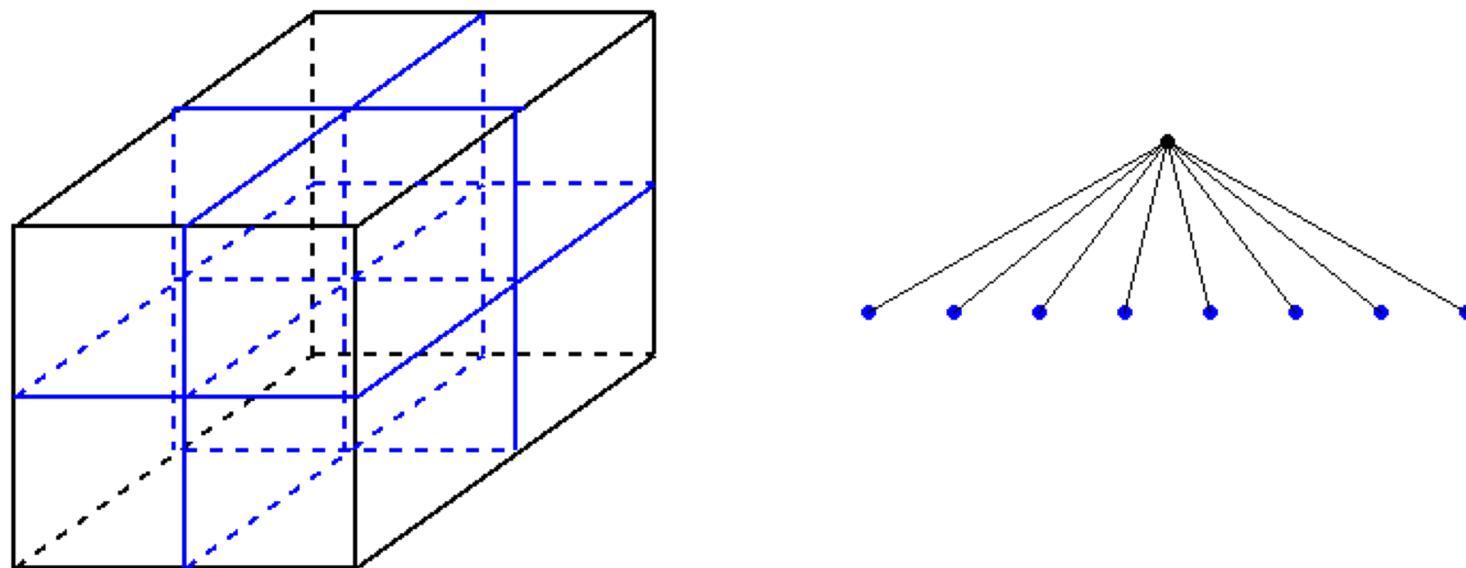


# Oct Trees

---

- ° Similar Data Structure to subdivide space

2 Levels of an Octree



## Using Quad Trees and Oct Trees

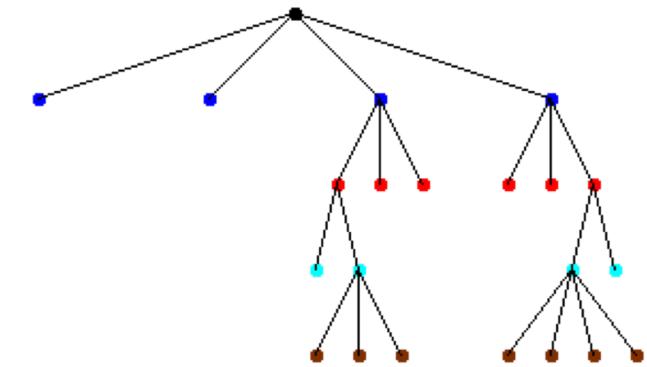
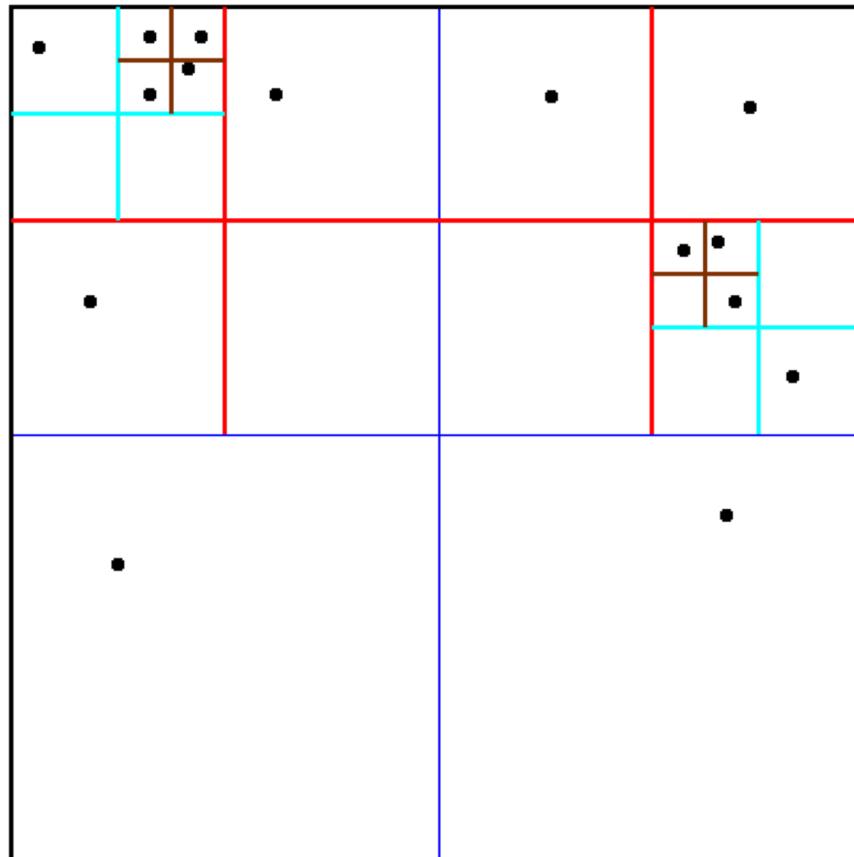
---

- ° All our algorithms 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

# 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 excluded

**In practice, have  $q > 1$  particles/square; tuning parameter  
(code to build data structure on hidden slide)**

## Cost of Adaptive Quad Tree Construction

---

- ° **Cost  $\leq N * \text{maximum cost of Quad\_Tree\_Insert}$**   
 **$= O(N * \text{maximum depth of Quad\_Tree})$**
- ° **Uniform Distribution of particles**
  - Depth of Quad\_Tree =  $O(\log N)$
  - Cost  $\leq O(N * \log N)$
- ° **Arbitrary distribution of particles**
  - Depth of Quad\_Tree =  $O(\# \text{ bits in particle coords}) = O(b)$
  - Cost  $\leq O(bN)$

# Outline

---

- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force 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 related algorithms**

# Barnes-Hut Algorithm

---

- “A Hierarchical  $O(n \log n)$  force calculation algorithm”,  
J. Barnes and P. Hut, Nature, v. 324 (1986), 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  $f(k) \sim 0$ )

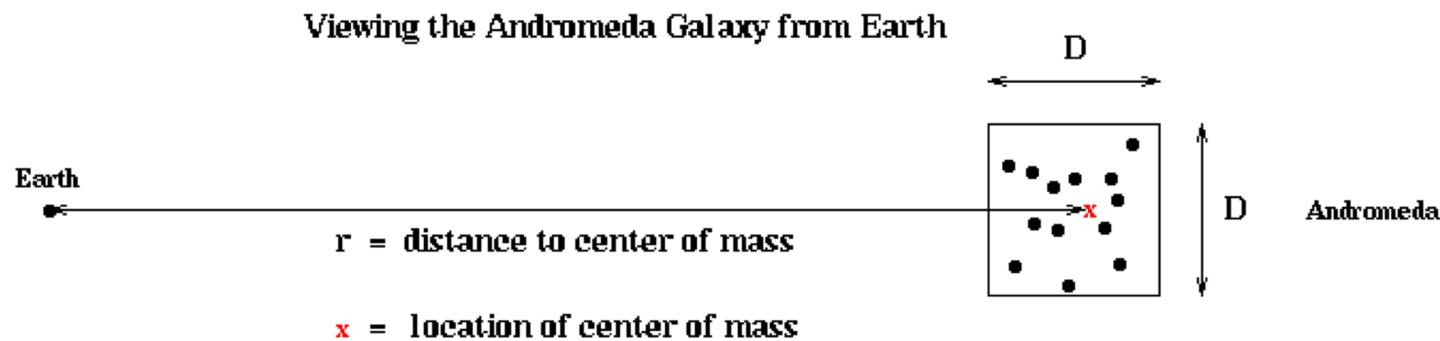
- High Level Algorithm (in 2D, for simplicity)

- 1) Build the QuadTree using QuadTreeBuild
  - ... already described, cost =  $O(N \log N)$  or  $O(bN)$
- 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  $O(bN)$
  - ... (code on hidden slide)
- 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 3 of BH: compute force on each particle

---

- 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
- 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
  - $\theta$  = user supplied error tolerance  $< 1$
  - Use CM and TM to approximate force of node on box if  $D/r < \theta$



## Details of Step 3 of BH (analysis on hidden slide)

---

```
... 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  
        f = force computed using direct formula  
    else  
        r = distance from particle k to CM of particles in n  
        D = size of n  
        if D/r < θ ... ok to approximate by CM and TM  
            compute f approximately using CM and TM  
        else ... need to look inside node  
            for all children c of n  
                f = f + TreeForce ( k, c )  
            end for  
        end if  
    end if
```

# Outline

---

- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force 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 related algorithms**

# Fast Multiple Method (FMM)

---

- “A fast algorithm for particle simulation”, L. Greengard and V. Rokhlin, J. Comp. Phys. V. 73, 1987, many later papers
  - Many awards
- 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.
- 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

---

- FMM will compute a compact expression for potential  $\phi(x,y,z)$  which can be evaluated and/or differentiated at any point
- In 3D with  $x,y,z$  coordinates
  - Potential =  $\phi(x,y,z) = -1/r = -1/(x^2 + y^2 + z^2)^{1/2}$
  - Force = -grad  $\phi(x,y,z) = -(\frac{d\phi}{dx}, \frac{d\phi}{dy}, \frac{d\phi}{dz}) = -(x,y,z)/r^3$
- In 2D with  $x,y$  coordinates
  - Potential =  $\phi(x,y) = \log r = \log(x^2 + y^2)^{1/2}$
  - Force = -grad  $\phi(x,y) = -(\frac{d\phi}{dx}, \frac{d\phi}{dy}) = -(x,y)/r^2$
- In 2D with  $z = x+iy$  coordinates,  $i = \sqrt{-1}$ 
  - Potential =  $\phi(z) = \log |z| = \text{Real}(\log z)$   
... because  $\log z = \log |z|e^{i\theta} = \log |z| + i\theta$
  - Drop  $\text{Real}()$  from calculations, for simplicity
  - Force =  $-(x,y)/r^2 = -z / |z|^2$
- Later: Kernel Independent FMM

## 2D Multipole Expansion (Taylor expansion in 1/z)

---

$\phi(z) = \text{potential due to } z_k, k=1,\dots,n$

$$= \sum_k m_k * \log |z - z_k|$$

$$= \text{Real}(\sum_k m_k * \log (z - z_k))$$

... since  $\log z = \log |z|e^{i\theta} = \log |z| + i\theta$

... drop Real() from now on

$$= \sum_k m_k * [\log(z) + \log(1 - z_k/z)]$$

... how logarithms work

$$= M * \log(z) + \sum_k m_k * \log(1 - z_k/z)$$

... where  $M = \sum_k m_k$

$$= M * \log(z) - \sum_k m_k * \sum_{e \geq 1} (z_k/z)^e/e$$

... Taylor expansion converges if  $|z_k/z| < 1$

$$= M * \log(z) - \sum_{e \geq 1} z^{-e} \sum_k m_k z_k^e/e$$

... swap order of summation

$$= M * \log(z) - \sum_{e \geq 1} z^{-e} \alpha_e$$

... where  $\alpha_e = \sum_k m_k z_k^e/e$  ... called Multipole Expansion

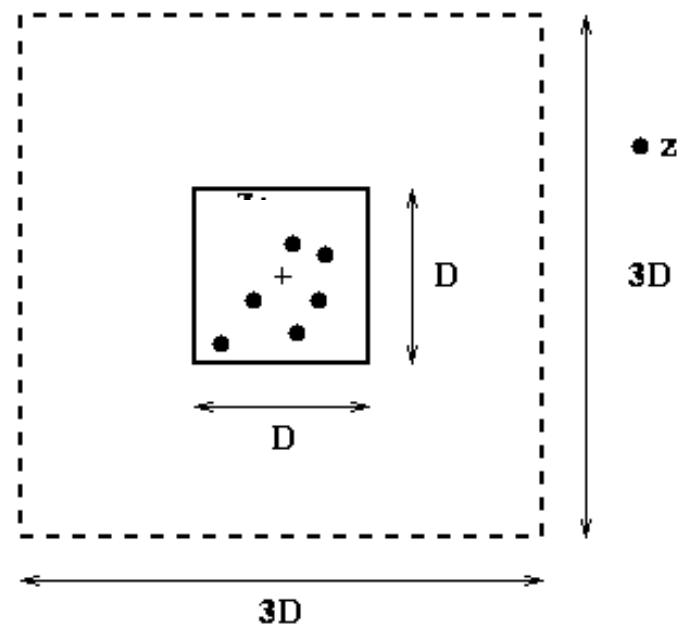
Keep leading r terms of sum, error will decrease like  $(\max |z_k|/|z|)^{r+1}$

First two terms contain same info as BH ( $M$  and  $\alpha_1 = \sum_k m_k z_k$ )

# Outer(n) and Outer Expansion

- Same idea for expansion around node  $n$  with center  $z_n$ :  
$$\phi(z) \sim M * \log(z - z_n) - \sum_{r \geq e \geq 1} (z - z_n)^{-e} \alpha_e$$
- $\text{Outer}(n) = (M, \alpha_1, \alpha_2, \dots, \alpha_r, z_n)$ 
  - Stores data for evaluating  $\phi(z)$  outside node  $n$  due to particles inside  $n$
  - Error small for  $z$  outside dotted line
  - Cost of evaluating  $\phi(z)$  is  $O(r)$ , independent of #particles inside  $n$
  - Cost grows linearly with desired number of bits of precision  $\sim r$
- Will be computed for each node in QuadTree
- Analogous to (TM, CM) in Barnes-Hut

Error outside larger box bounded by  $O(1/2^r)$

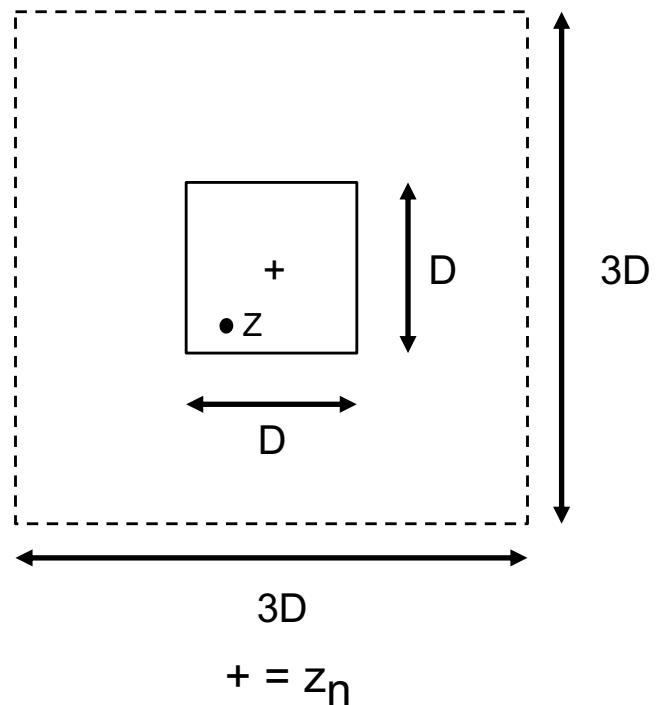


$$+ = z_n$$

## Inner(n) and Inner Expansion

- ° Outer(n) used to evaluate potential outside node n due to particles inside n
- ° Inner(n) will be used to evaluate potential inside node n due to particles outside n

- $\sum_{0 \leq e \leq r} \beta_e * (z - z_n)^e$
- $z_n$  = center of node n, a D-by-D box
- $\text{Inner}(n) = (\beta_0, \beta_1, \dots, \beta_r, z_n)$
- Particles outside n must lie outside 3D-by-3D box centered at  $z_n$



# Top Level Description of FMM

---

- (1) Build the QuadTree
- (2) *Call Build\_Outer(root), to compute outer expansions of each node n in the QuadTree*
  - ... Traverse QuadTree from bottom to top,
  - ... combining outer expansions of children
  - ... to get out outer expansion of parent
- (3) Call Build\_Inner(root), to compute inner expansions of each node n in the QuadTree
  - ... Traverse QuadTree from top to bottom,
  - ... converting outer to inner expansions
  - ... and combining them
- (4) For each leaf node n, add contributions of nearest particles directly to Inner(n)
  - ... final Inner(n) is desired output: expansion for potential at each point due to all particles

## Step 2 of FMM: compute Outer(n) for each node n in QuadTree

... Compute Outer(n) for each node of the QuadTree

outer = Build\_Outer( root )

function ( M,  $\alpha_1, \dots, \alpha_r$ ,  $z_n$ ) = Build\_Outer( n ) ... compute outer expansion of node n

if n if a leaf ... it contains 1 (or a few) particles

compute and return Outer(n) = ( M,  $\alpha_1, \dots, \alpha_r$ ,  $z_n$ ) directly from  
its definition as a sum

else ... “post order traversal”: process parent after all children

Outer(n) = 0

for all children  $c(k)$  of n ...  $k = 1, 2, 3, 4$

Outer(  $c(k)$  ) = Build\_Outer(  $c(k)$  )

Outer(n) = Outer(n) +

Outer\_shift( Outer( $c(k)$ ), center(n))

... convert expansion around  $c(k)$  to

... expansion around center(n)

... (details on hidden slides)

... then add component by component

endfor

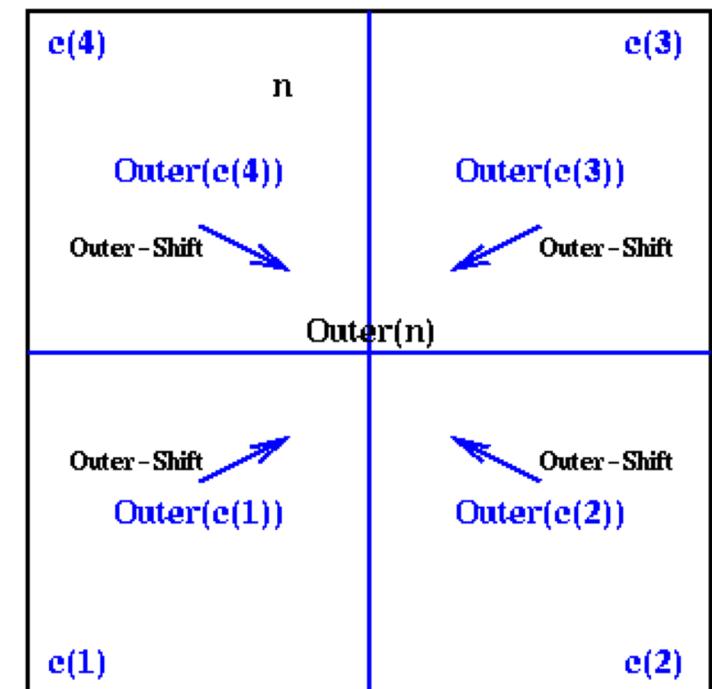
return Outer(n)

end if

Cost =  $O(\# \text{ nodes in QuadTree}) = O(N)$

same as for Barnes-Hut

Inner Loop of Build\_Outer



# Top Level Description of FMM

---

- (1) Build the QuadTree
- (2) Call `Build_Outer(root)`, to compute outer expansions of each node  $n$  in the QuadTree
  - ... Traverse QuadTree from bottom to top,
  - ... combining outer expansions of children
  - ... to get out outer expansion of parent
- (3) Call `Build_Inner(root)`, to compute inner expansions of each node  $n$  in the QuadTree
  - ... Traverse QuadTree from top to bottom,
  - ... converting outer to inner expansions
  - ... and combining them
- (4) For each leaf node  $n$ , add contributions of nearest particles directly into `Inner(n)`
  - ... final `Inner(n)` is desired output: expansion for potential at each point due to all particles

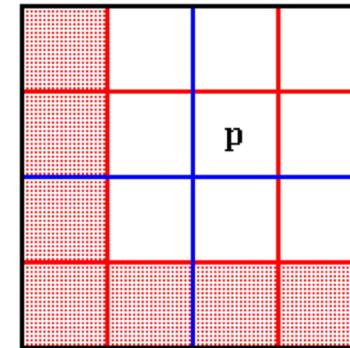
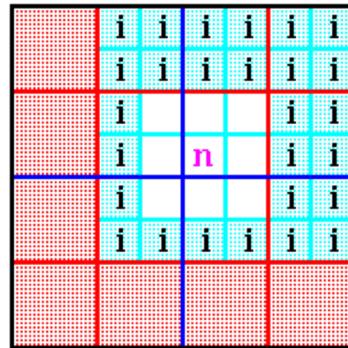
## Step 3 of FMM: Computing Inner( $n$ ) from other expansions

- ° Which other expansions?

- As few as necessary to compute the potential accurately
- Inner expansion of  $p = \text{parent}(n)$  will account for potential from particles far enough away from parent (**red nodes** below)
- Outer expansions will account for potential from particles in boxes at same level in **Interaction Set (nodes labeled i below)**

Interaction\_Set( $n$ ) for the Fast Multipole Method

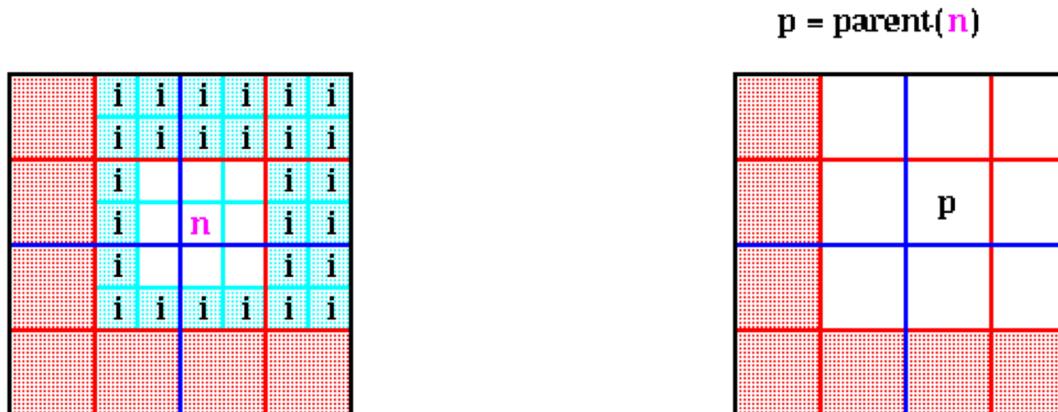
$p = \text{parent}(n)$



## Step 3 of FMM: Computing Inner(n) from other expansions

- We will use `Inner_shift` and `ConvertO2I` to build each `Inner(n)` by combining expansions from other nodes
- Which other nodes?
  - As few as necessary to compute the potential accurately
  - `Inner_shift(Inner(parent(n)), center(n))` will account for potential from particles far enough away from parent (**red nodes** below)
  - `ConvertO2I(Outer(i), center(n))` will account for potential from particles in boxes at same level in **Interaction Set (nodes labeled i below)**
  - (details on hidden slides)

Interaction\_Set(n) for the Fast Multipole Method



## Step 3 of FMM: Compute Inner(n) for each n in QuadTree

---

... Compute Inner(n) for each node of the QuadTree

outer = **Build\_Inner( root )**

---

```
function (  $\beta_1, \dots, \beta_r, z_n$  ) = Build_Inner( n ) ... compute inner expansion of node n
    p = parent(n) ... p=nil if n = root
    Inner(n) = Inner_shift( Inner(p), center(n) ) ... Inner(n) = 0 if n = root
    for all i in Interaction_Set(n) ... Interaction_Set(root) is empty
        Inner(n) = Inner(n) + ConvertO2I( Outer(i), center(n) )
            ... add component by component
    end for
    for all children c of n ... complete preorder traversal of QuadTree
        Build_Inner( c )
    end for
```

**Cost =  $O(\# \text{ nodes in QuadTree})$**

**=  $O(N)$**

# Top Level Description of FMM

---

- (1) Build the QuadTree
- (2) Call `Build_Outer(root)`, to compute outer expansions of each node  $n$  in the QuadTree
  - ... Traverse QuadTree from bottom to top,
  - ... combining outer expansions of children
  - ... to get out outer expansion of parent
- (3) Call `Build_Inner(root)`, to compute inner expansions of each node  $n$  in the QuadTree
  - ... Traverse QuadTree from top to bottom,
  - ... converting outer to inner expansions
  - ... and combining them
- (4) *For each leaf node  $n$ , add contributions of nearest particles directly into  $\text{Inner}(n)$* 
  - ... if 1 node/leaf, then each particles accessed once,
  - ... so cost =  $O(N)$
  - ... final  $\text{Inner}(n)$  is desired output: expansion for potential at each point due to all particles

# Outline

---

- **Motivation**
  - Obvious algorithm for computing gravitational or electrostatic force 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 related algorithms**

# Parallelizing Hierarchical N-Body codes

---

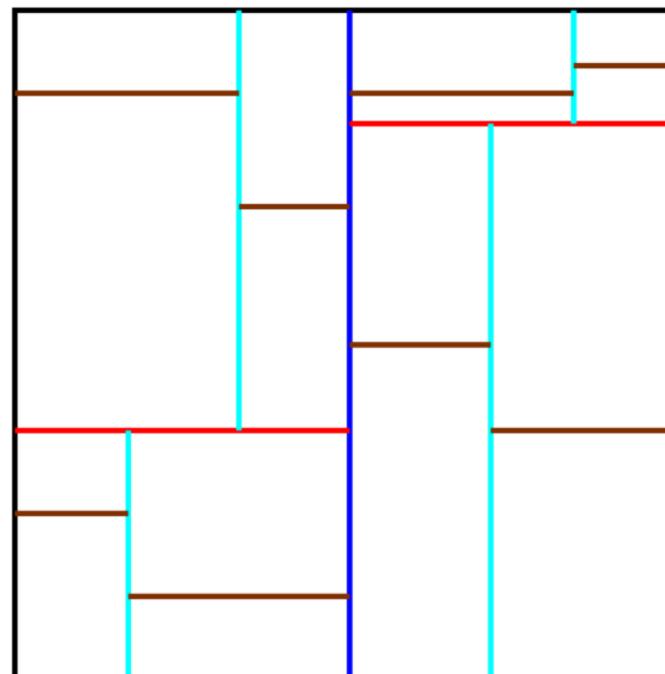
- Barnes-Hut, FMM and related algorithm have similar computational structure:
  - 1) Build the QuadTree
  - 2) Traverse QuadTree from leaves to root and build outer expansions (just (TM,CM) for Barnes-Hut)
  - 3) Traverse QuadTree from root to leaves and build any inner expansions (FMM only)
  - 4) Traverse QuadTree to accumulate forces for each particle
- One parallelization scheme will work for them all
  - Based on D. Blackston and T. Suel, Supercomputing 97
    - UCB PhD Thesis, David Blackston, “Pbody”
    - Autotuner for N-body codes
  - 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

## Load Balancing Scheme 1: 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

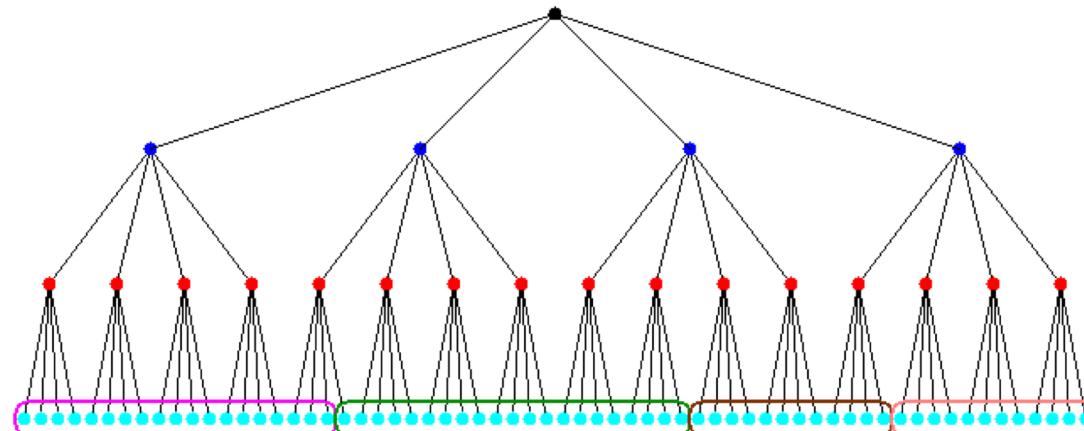
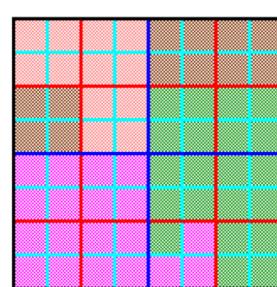
Partitioning  
for 16 procs:



## Load Balancing Scheme 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 (details on hidden slides)
  - Assign contiguous blocks of nodes with work  $W/p$  to processors: locality
  - **Works well in 3D**

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



## 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
- ° As particles move, may need to occasionally repeat construction, so should not be too slow

# 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  $j$ ; Does  $k$  need  $n$ ?
  - 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 (Interaction Set)

# Performance Results - 1

---

- ° **512 Proc Intel Delta**

- Warren and Salmon, Supercomputing 92, Gordon Bell Prize
- 8.8 M particles, uniformly distributed
- .1% to 1% RMS error, Barnes-Hut
- 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, running FMM
  - Blackston, 1999
  - $10^{-4}$  RMS error
  - Generally 80% efficient on up to 32 processors
  - Example: 50K particles, both uniform and nonuniform
    - preliminary results; lots of tuning parameters to set

	Uniform		Nonuniform	
	1 proc	4 procs	1 proc	4 procs
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

- Ultimate goal - portable, tunable code including all useful variants

## Performance Results - 3



# Optimizing and Tuning the Fast Multipole Method for Multicore and Accelerator Systems

**Georgia Tech**

– **Aparna Chandramowlishwaran, Aashay Shringarpure, Ilya Lashuk; George Biros, Richard Vuduc**

**Lawrence Berkeley National Laboratory**

– **Sam Williams, Lenny Oliker**

- Presented at IPDPS 2010
- Source: Richard Vuduc

## Summary

---

- ▶ **First cross-platform single-node multicore study of tuning the fast multipole method (FMM)**
  - ▶ Explores data structures, SIMD, mixed-precision, multithreading, and tuning
  - ▶ Show
    - ▶ **25x speedups on Intel Nehalem –**
      - ▶ 2-sockets × 4-cores/socket × **2-thr/core** = **16 threads**
    - ▶ **9.4x on AMD Barcelona**
      - ▶ 2-sockets × 4-cores/socket × **1-thr/core** = **8 threads**
    - ▶ **37.6x on Sun Victoria Falls**
      - ▶ 2-sockets × 8-cores/socket × **8-thr/core** = **128 threads**
- ▶ **Surprise? Multicore ~ GPU in performance & energy efficiency for the FMM**

# Optimizations tried (manual and autotuning)

---

- **Uses KIFMM = Kernel Independent FMM**
  - Applies to “any” kernel, not just gravity/electrostatics
  - Requires subroutine to evaluate kernel, builds own expansions
    - Ex: (modified) Laplace, Stokes
    - Approximate particles inside square/box by evenly spaced particles on circle/sphere
  - FFT used to build expansions; tunable

## ► Single-core, manually coded & tuned

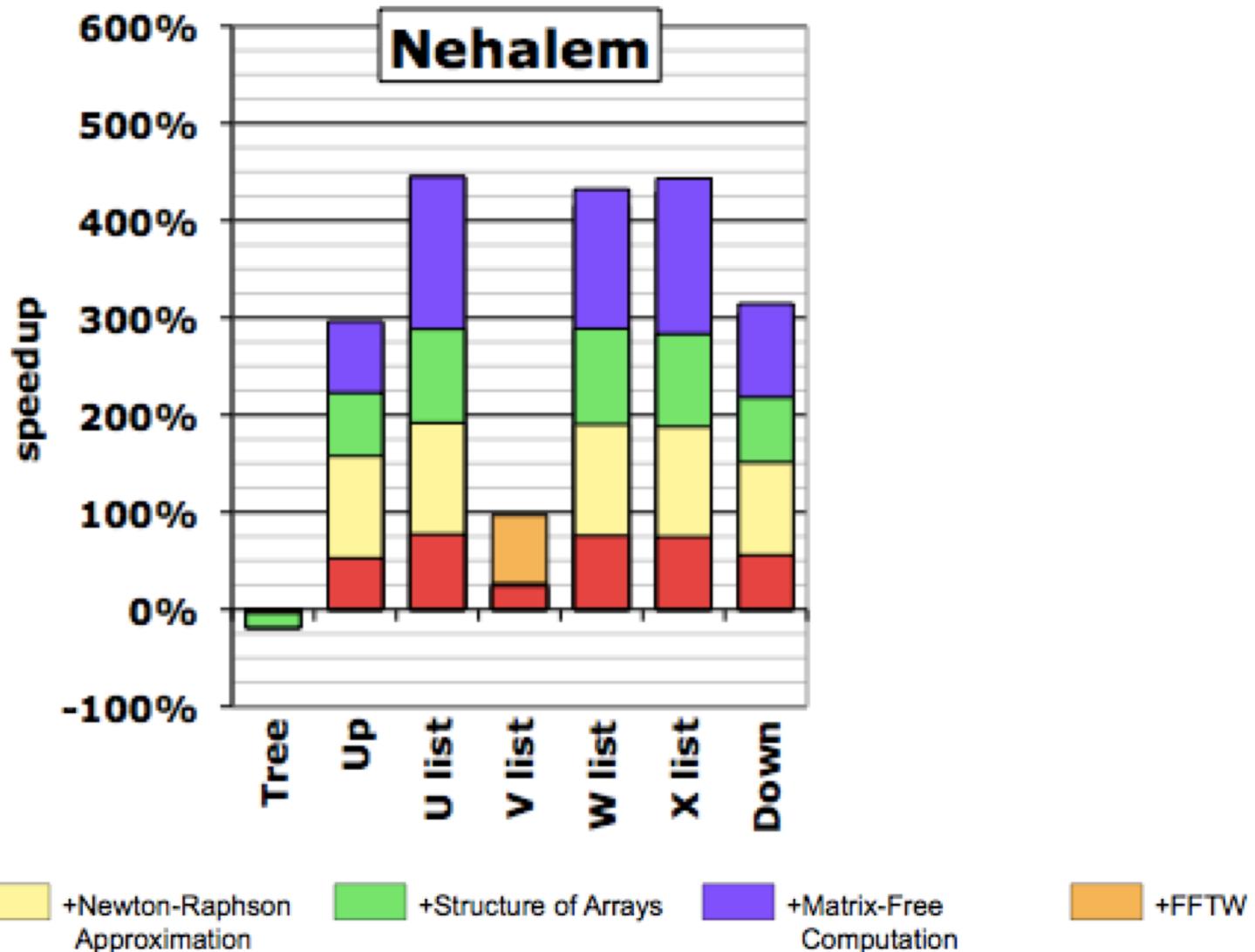
- **Low-level:** SIMD vectorization (x86)
- **Numerical:** `rsqrt ps` + Newton-Raphson (x86)
- **Data:** Structure reorg. (transpose or “SOA”)
- **Traffic:** Matrix-free via interprocedural loop fusion
- FFTW plan optimization

## ► OpenMP parallelization

## ► Algorithmic tuning of max particles per box, $q$

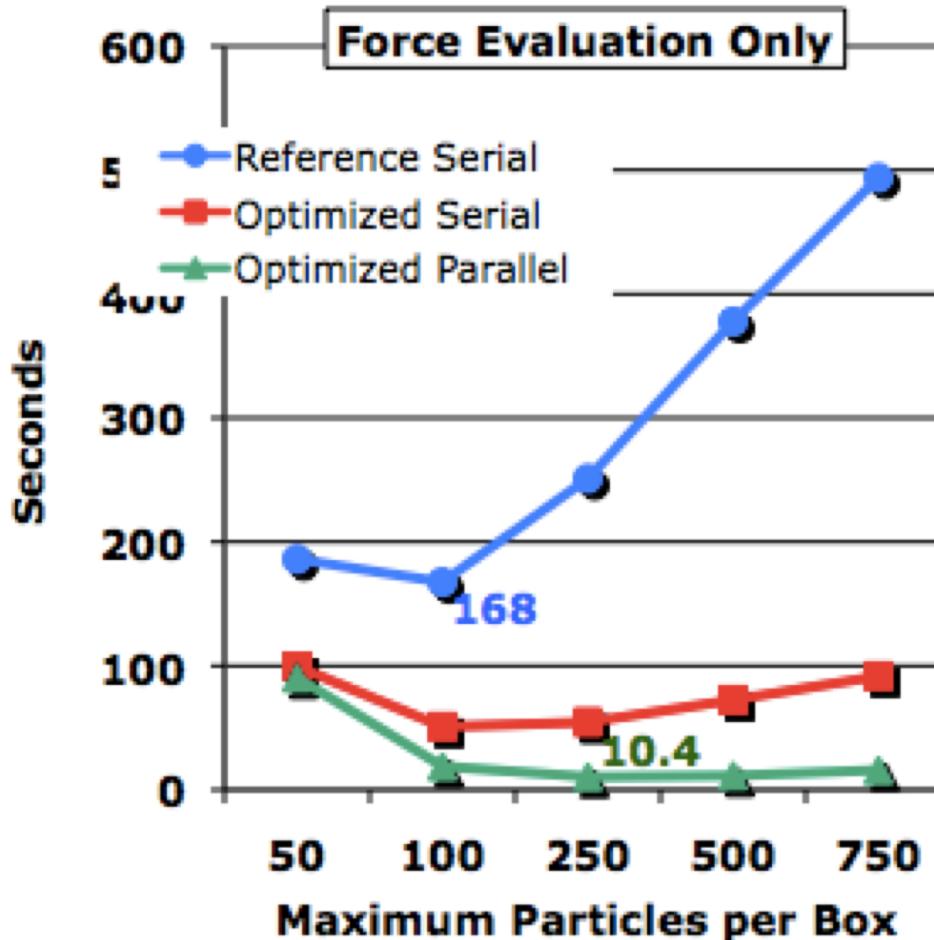
# Single-core Optimizations

## Double-Precision, Non-uniform (ellipsoidal)



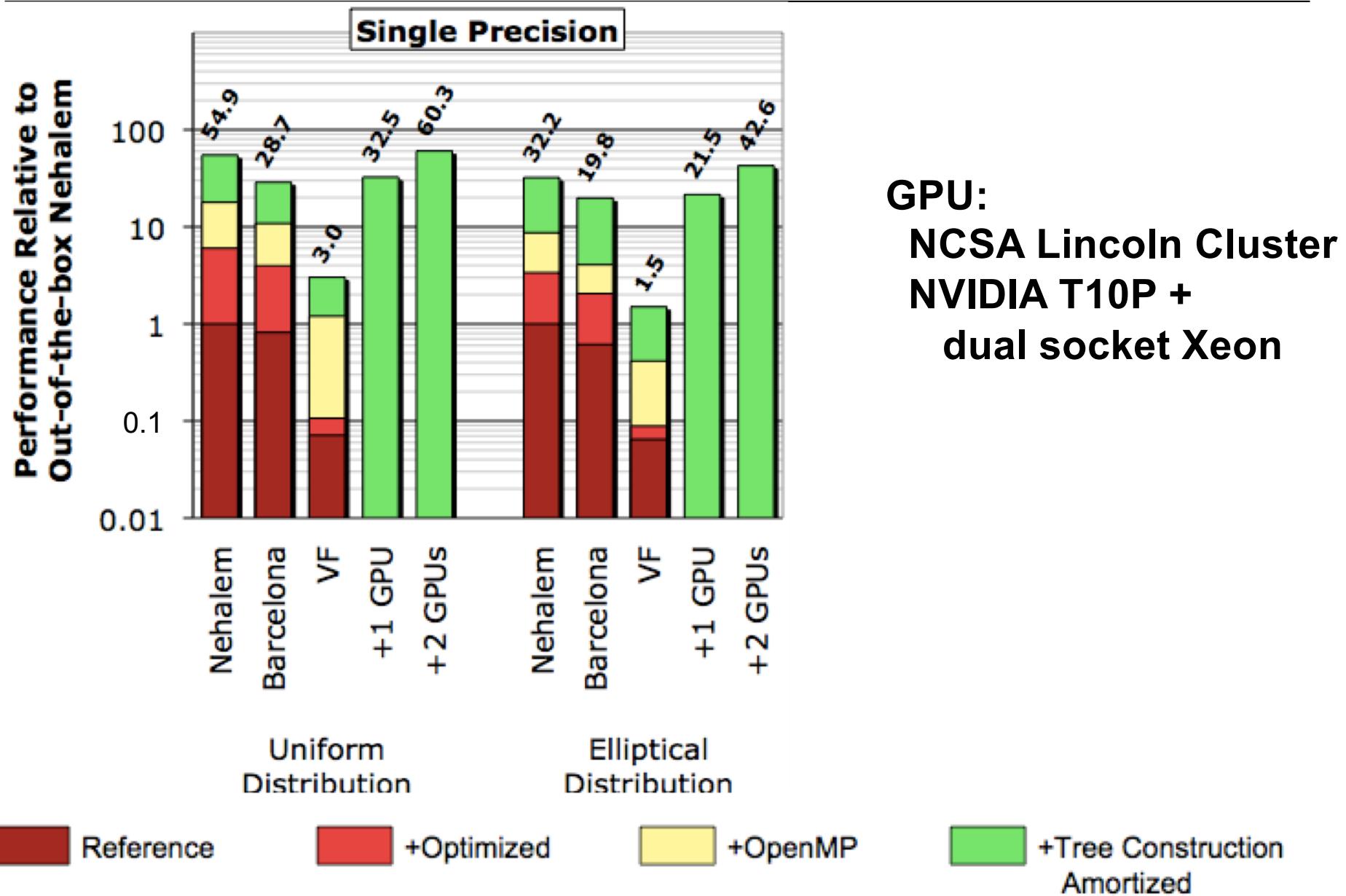
Reference: kifmm3d [Ying, Langston, Zorin, Biros]

# Algorithmic Tuning of $q = \text{Max pts / box}$ - Nehalem



***Shape of curve changes as we introduce optimizations.***

# Cross-Platform Performance Comparison (Summary)



**GPU:**  
NCSA Lincoln Cluster  
NVIDIA T10P +  
dual socket Xeon

***Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.***

# Minimizing Communication in N-Body Problem

---

- **Hierarchical Methods**

- Reducing arithmetic good for reducing communication too!
- Deriving communication lower bounds is an open problem
  - Answer is approximate, so lower bound may depend on desired accuracy
  - Lower bound may also depend on particle distribution
  - Open problem (probably hard)

- **Direct methods**

- Thm: Suppose  $p$  processors compute interactions among  $n$  particles, using local memories of size  $M$ . If each processor does an equal amount of work ( $n^2/p$  interactions) then the number of words that a processor must communicate is  $\Omega((n^2/p)/M)$ , and the number of messages is  $\Omega((n^2/p)/M^2)$
- If not computing all  $n^2$  interactions (eg cutoff distance), replace  $n^2$  by #interactions in Thm
- Attainable when  $n/p \leq M \leq n/p^{1/2}$ 
  - $M = n/p$  : #words =  $n$ , #messages =  $p$
  - $M = n/p^{1/2}$  : #words =  $n/p^{1/2}$ , #messages =  $O(1)$  (12x speedups)