Lecture 10:
Particle Methods

December 4, 2020

Reminders

- Homework 2 due tonight! 23:59
- Homework 3 out now (last HW), due January 8, 2021 23:59

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

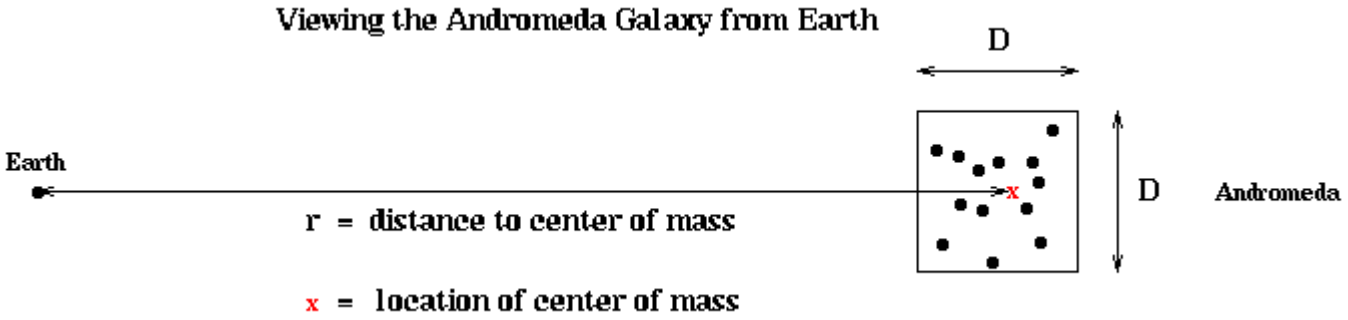
Particle Simulation

```
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
  - Nearest_neighbor_force requires interacting with a few neighbors, so still O(n)
    - van der Waals, bouncing balls
  - 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^2), but we can do better...

Reducing the number of particles in the force sum

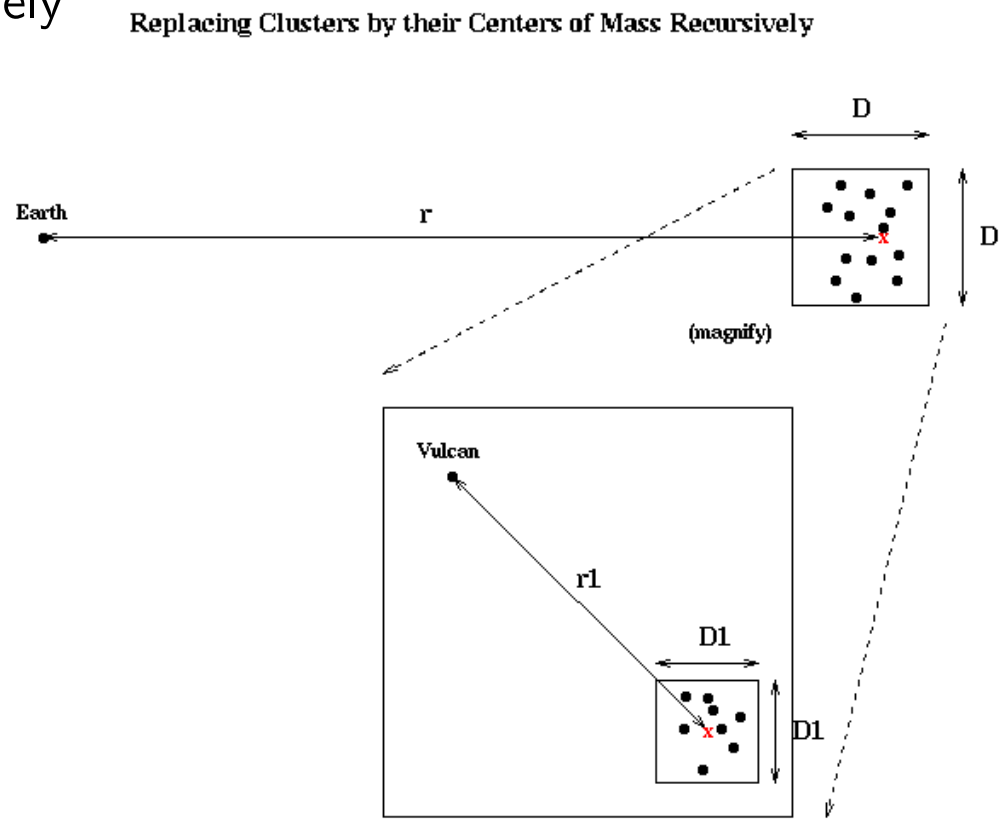
- All later divide and conquer algorithms use same intuition
- Consider computing force on earth due to all celestial bodies
  - Look at night sky, # terms in force sum ≥ number of visible stars
  - Oops! One "star" is really the Andromeda galaxy, which contains billions of real stars
    - Seems like 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 D1/r1 is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
  - Boxes nest in boxes recursively

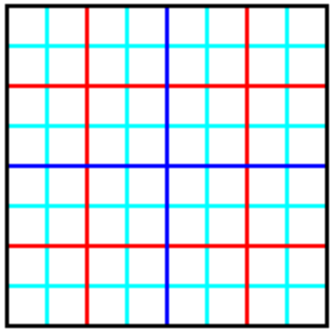


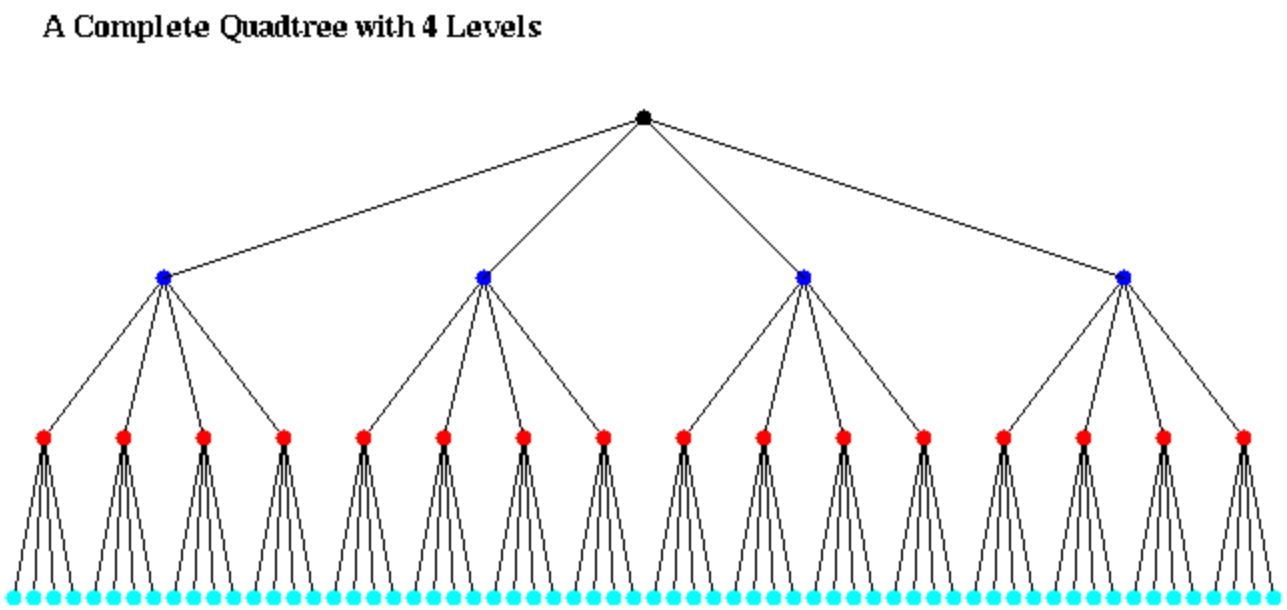
Outline

- 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

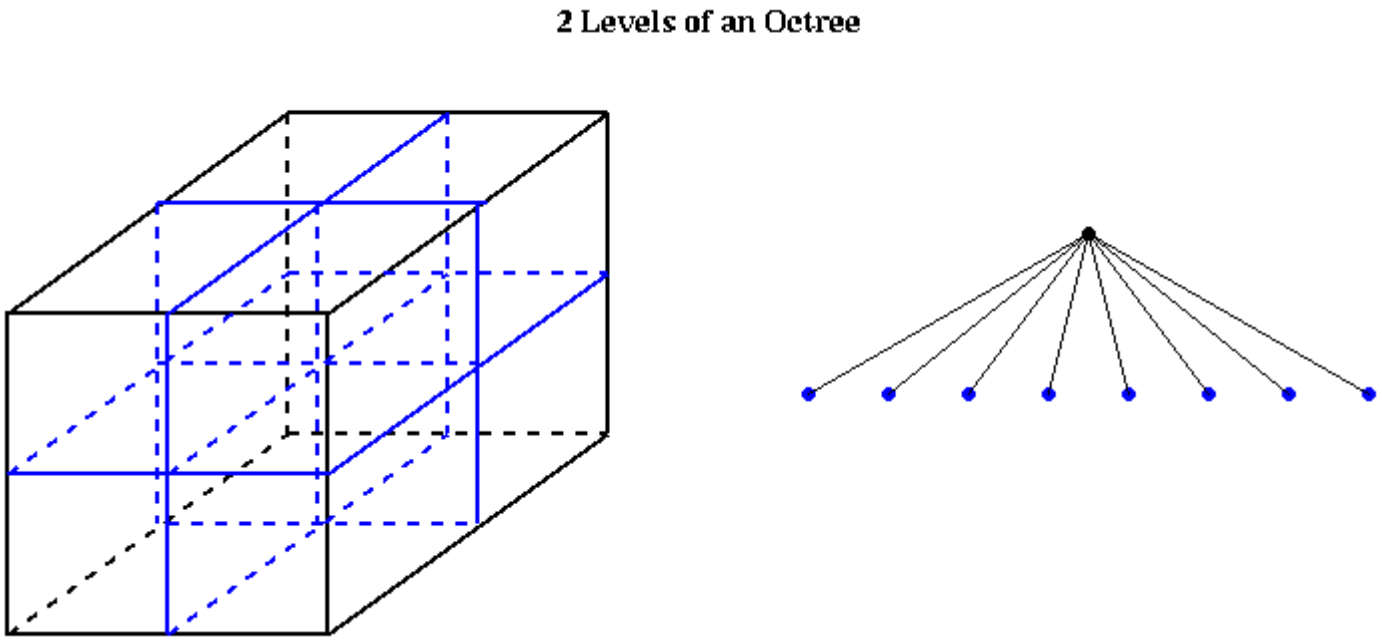
- Data structure to subdivide the plane
  - Nodes can contain coordinates of center of box, side length
  - Eventually also coordinates of CM, total mass, etc.
- In a complete quad tree, each nonleaf node has 4 children





Oct Trees

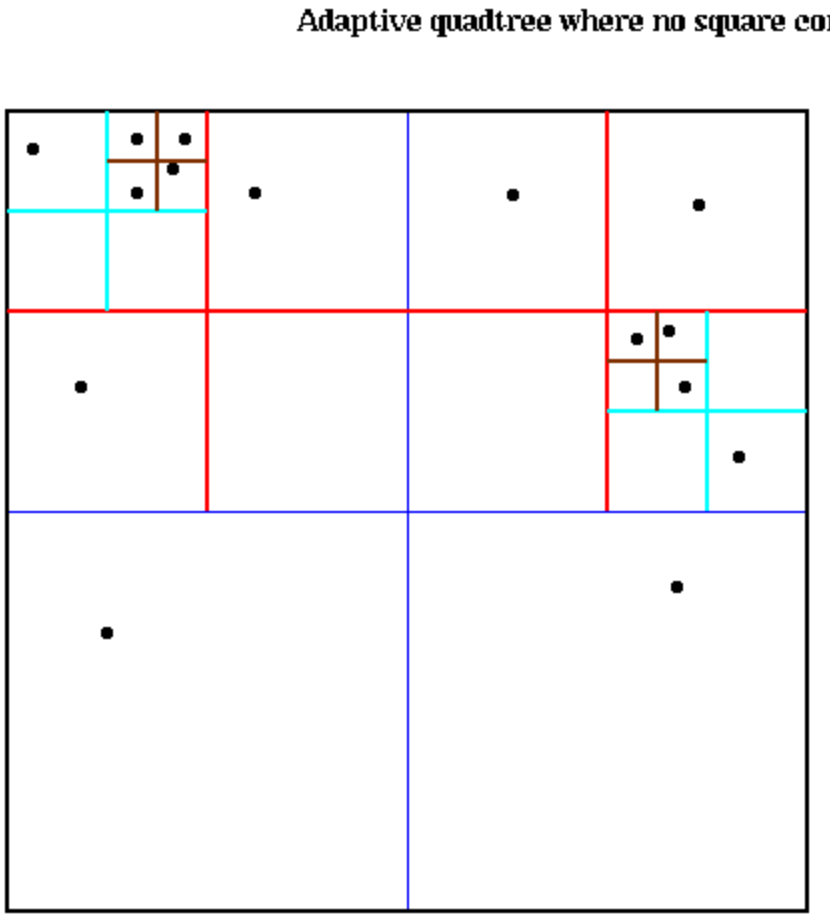
- Similar Data Structure to subdivide space

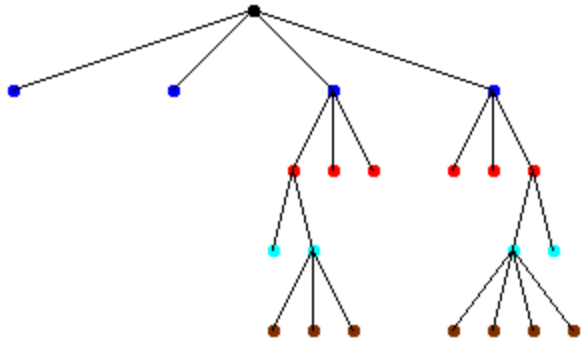


Using Quad Trees and Oct Trees

- 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





Child nodes enumerated counterclockwise from SW corner, empty ones excluded

In practice, have q>1 particles/square; tuning parameter

Adaptive Quad Tree Algorithm (Oct Tree analogous)

```
Procedure Quad_Tree_Build
   Quad_Tree = {emtpy}
   for j = 1 to N                          … loop over all N particles
      Quad_Tree_Insert(j, root)            … insert particle j in QuadTree
   endfor
   …  At this point, each leaf of Quad_Tree will have 0 or 1 particles
   …  There will be 0 particles when some sibling has 1
   Traverse the Quad_Tree eliminating empty leaves  … via, say Breadth First Search
```

```
Procedure Quad_Tree_Insert(j, n) … Try to insert particle j at node n in Quad_Tree
   if n an internal node                … n has 4 children
      determine which child c of node n contains particle j
      Quad_Tree_Insert(j, c)
  else if n contains 1 particle   …  n is a leaf    Easy change for  q > 1 particles/leaf
      add n's 4 children to the Quad_Tree
      move the particle already in n into the child containing it
      let c be the child of n containing j
      Quad_Tree_Insert(j, c)
   else                                …  n empty
      store particle j in node n
   end
```

Cost of Adaptive Quad Tree Construction

- Cost ≤ N * maximum cost of Quad_Tree_Insert
  = O(N * maximum depth of Quad_Tree)

  - Uniform Distribution of particles
    - Depth of Quad_Tree = O(log N)
    - Cost ≤ O(N * log N)
  - Arbitrary distribution of particles
    - Depth of Quad_Tree = O(# bits in particle coords) = O(b)
    - Cost ≤ O(b N)

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:

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

$$\sim 1\%$$

(other measures better if some true f(k) ~ 0)

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

```
1) Build the QuadTree using QuadTreeBuild
     … 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)
    … (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 2 of BH: compute CM and total mass of each node

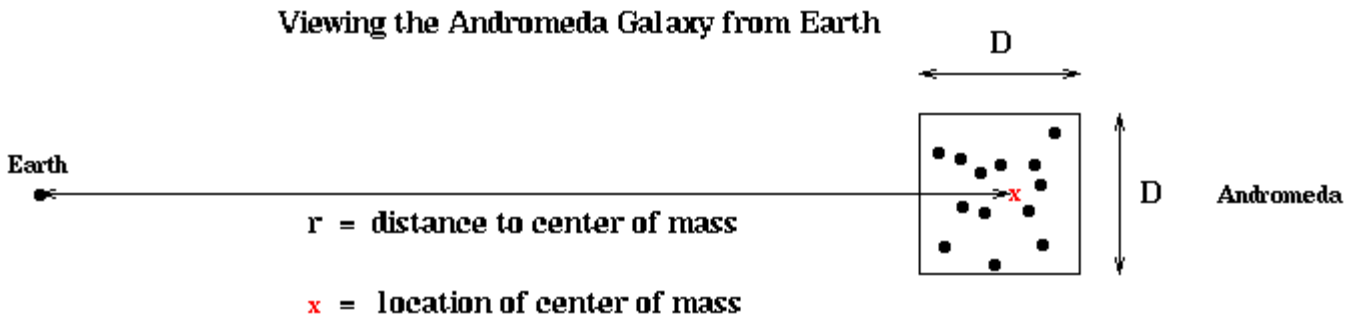
```
… Compute the CM = Center of Mass and TM = Total Mass of all the particles
… in each node of the QuadTree
( TM, CM ) = Compute_Mass( root )
```

```
function ( TM, CM ) = Compute_Mass( n )    … compute the CM and TM of node n
      if n contains 1 particle
           … the TM and CM are identical to the particle's mass and location
           store (TM, CM) at n
           return (TM, CM)
      else       … "post order traversal": process parent after all children
           for all children c(j) of n  … j = 1,2,3,4
                 ( TM(j), CM(j) ) = Compute_Mass( c(j) )
           endfor
           TM = TM(1) + TM(2) + TM(3) + TM(4)
                  … the total mass is the sum of the children's masses
           CM = ( TM(1)*CM(1) + TM(2)*CM(2) + TM(3)*CM(3) + TM(4)*CM(4) ) / TM
                  … the CM is the mass-weighted sum of the children's centers of mass
           store ( TM, CM ) at n
           return ( TM, CM )
       end if
```

Cost = O(# nodes in QuadTree) = O(N log N) or O(b N)

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



Computing force on a particle due to a node

- Suppose node n, with CM and TM, and particle k, satisfy $D/r < q$
- 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

- $r = \left((x_k - x_{CM})^2 + (y_k - y_{CM})^2\right)^{1/2}$
- G = gravitational constant
- Force on k

$$G \times m \times TM \times (x_{CM} - x_k, y_{CM} - y_k, z_{CM} - z_k)/r^3$$

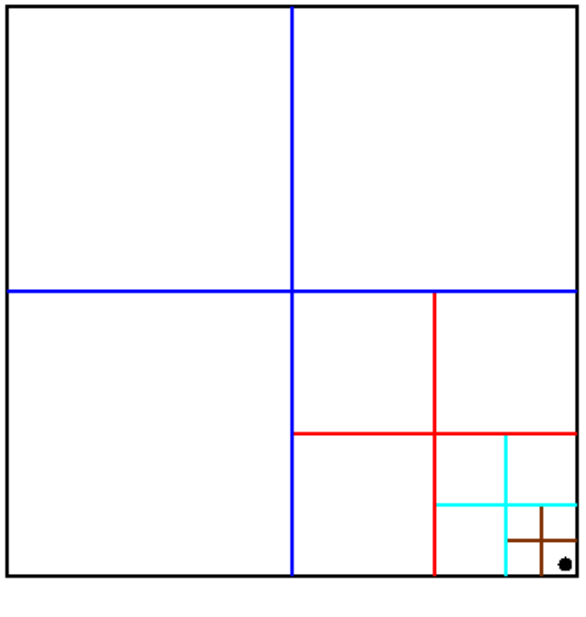
Details of Step 3 of BH

```
… for each particle, traverse the QuadTree to compute the force on it
for 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  < q      … 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 + TreeForce ( k, c )
             end for
        end if
   end if
```

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 TreeForce(k, root) = O(depth in QuadTree of leaf containing k)
  - Proof by Example (for $q > 1$):
    - For each undivided node = square, (except one containing k), $D/r < 1 < q$
    - There are 3 nodes at each level of the QuadTree
    - There is O(1) work per node
    - Cost = O(level of k)
  - Total cost = $O(S_k$ level of k) = O(N log N)
    - Strongly depends on q



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

- 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 f(x,y,z) which can be evaluated and/or differentiated at any point
- In 3D with x,y,z coordinates
  - Potential = $f(x,y,z) = -\frac{1}{r} = -1/(x^2+y^2+z^2)^{\frac{1}{2}}$
  - Force = -grad $f(x,y,z) = -\left(\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}\right) = -(x,y,z)/r^3$
- In 2D with x,y coordinates
  - Potential = f(x,y) = log r = log $(x^2 + y^2)^{1/2}$
  - Force = -grad f(x,y) = - (df/dx , df/dy) = -(x,y)/r^2
- In 2D with z = x+iy coordinates, i = sqrt(-1)
  - Potential = f(z) = log |z| = Real(log z)

    ... because log z = log |z|e^{iq} = log |z| + iq
  - Drop 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)

$f(z)$ = potential due to $z_{k,\ k=1,\ldots,n}$

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

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

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

... drop Real() from now on

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

... how logarithms work

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

... where $M = \Sigma_k\ m_k$

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

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

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

... swap order of summation

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

... where $a_e = \Sigma_k\ m_k\ z_k^e/e$ ... called Multipole Expansion

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

$f(z)$ = potential due to $z_{k,\ k=1,\ldots,n}$

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

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

... drop Real() from now on

$= M * \log(z) - \Sigma_{e\geq 1}\ z^{-e}\ a_e$ ... Taylor Expansion in 1/z

... where $M = \Sigma_k\ m_k$ = Total Mass and

... $a_e = \Sigma_k\ m_k\ z_k^e\ /e$

... This is called a Multipole Expansion in z

$= M * \log(z) - \Sigma_{r\geq e\geq 1}\ z^{-e}\ a_e + \text{error}(\ r\ )$

... r = number of terms in Truncated Multipole Expansion

... and $\text{error}(\ r\ ) = -\Sigma_{r<e} z^{-e}\ a_e$

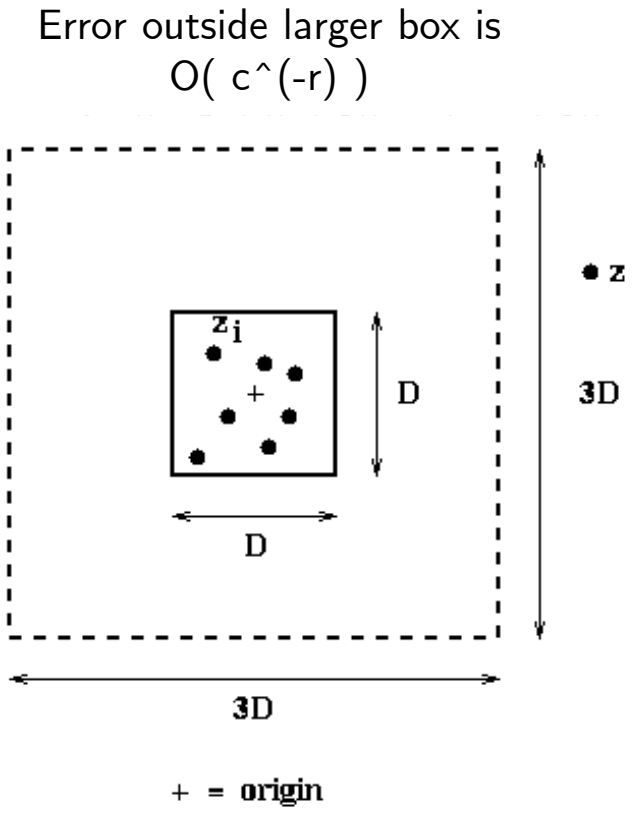
- Note that $a_1 = \Sigma_k\ m_k\ z_k = CM*M$
  so that M and a_1 terms have same info as Barnes-Hut

- $\text{error}(\ r\ ) = O(\ \{\max_k |z_k|\ /|z|\}^{r+1}\ )$ ... bounded by geometric sum

Error in Truncated 2D Multipole Expansion

- error(r) = $O(\{\max_k |z_k| / |z|\}^{r+1})$
- Suppose $\max_k |z_k| / |z| \le c < 1$, so error(r) $= O(c^{r+1})$
- Suppose all particles z_k lie inside a D-by-D square centered at origin
- Suppose z is outside a 3D-by-3D square centered at the origin
- c = (D/sqrt(2)) / (1.5*D) ~ .47 < .5
  - each term in expansion adds 1 bit of accuracy
  - 24 terms enough for single precision, 53 terms for double precision

- In 3D, can use spherical harmonics or other expansions



Outer(n) and Outer Expansion

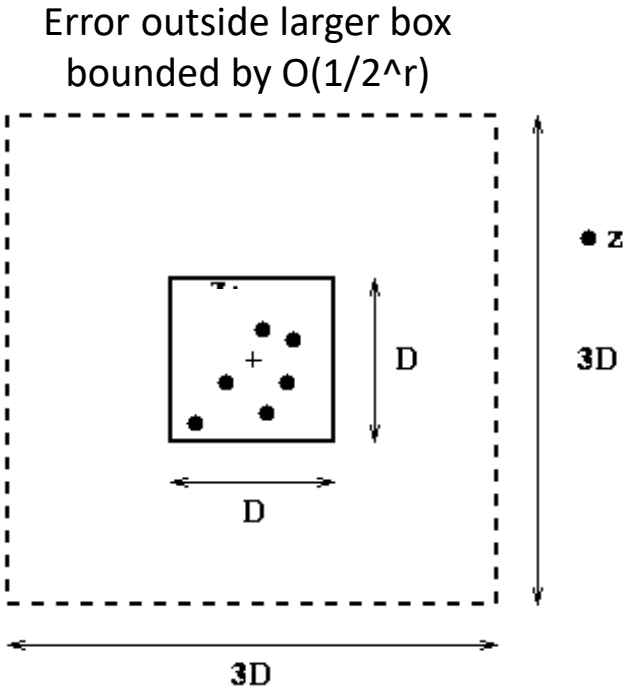
- Same idea for expansion around node n with center z_n :

  $f(z) \sim M * \log(z - z_n) - \Sigma_{r \geq e \geq 1} (z-z_n)^{-e} a_e$

° Outer(n) = ($M, a_1, a_2, \ldots, a_r, z_n$)
  - Stores data for evaluating f(z) outside node n due to particles inside n
  - Error small for z outside dotted line
  - Cost of evaluating f(z) is O(r), independent of #particles inside n
  - Cost grows linearly with desired number of bits of precision ~r

° Will be computed for each node in QuadTree

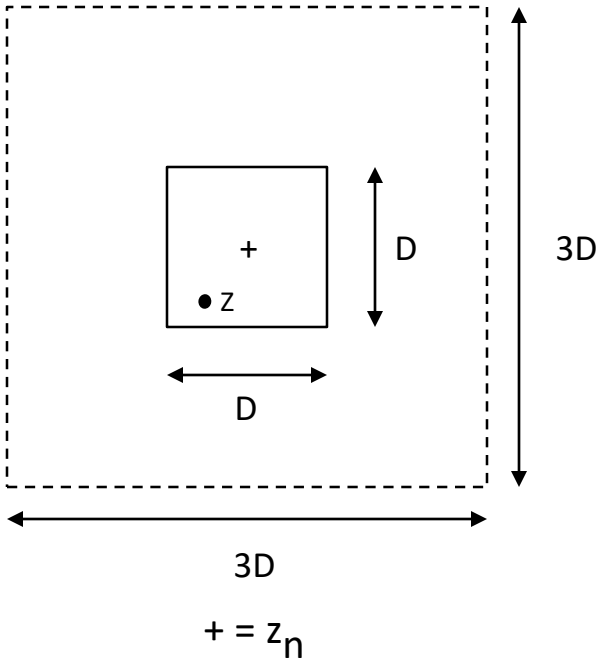
° Analogous to (TM,CM) in Barnes-Hut



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

- $\Sigma_{0 \le e \le r} b_e * (z-z_n)^e$
- z_n = center of node n, a D-by-D box
- Inner(n) = (b_0 , b_1 , ... , b_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

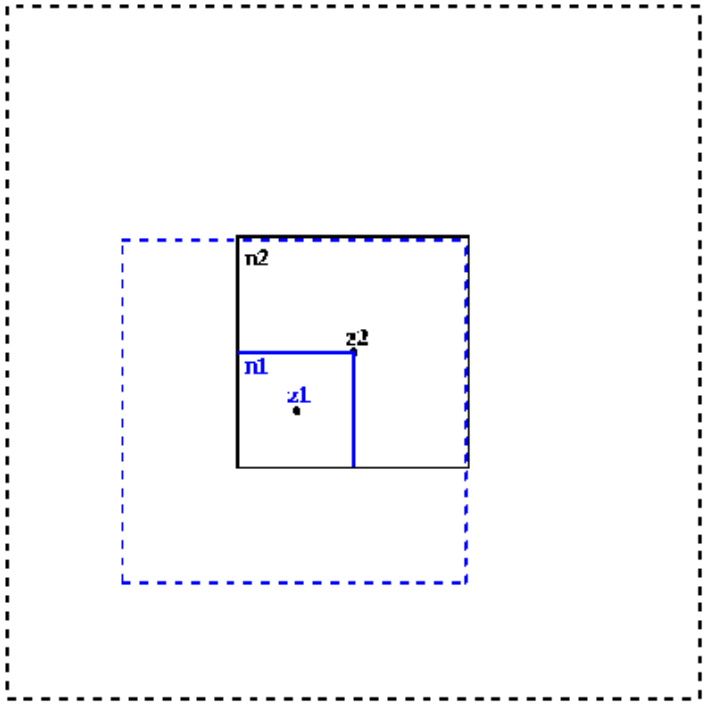
... and combining them

(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: Outer_shift: converting Outer(n_1) to Outer(n_2) (1/3)

- For step 2 of FMM (as in step 2 of BH) we want to compute Outer(n) cheaply from Outer(c) for all children c of n
- How to combine outer expansions around different points?
  - $f_k(z) \sim M_k * \log(z-z_k) - \sum_{r \geq e \geq 1} (z-z_k)^{-e} a_{ek}$ expands around z_k , k=1,2
  - First step: make them expansions around same point
- n_1 is a child (subsquare) of n_2
- z_k = center(n_k) for k=1,2
- Outer(n_1) expansion accurate outside blue dashed square, so also accurate outside black dashed square
- So there is an Outer(n_2) expansion with different a_k and center z_2 which represents the same potential as Outer(n_1) outside the black dashed box



Outer_shift: Details (2/3)

- Given expansion centered at z_1 (= child)

$$f_1(z) = M_1 * \log(z-z_1) + \Sigma_{r \geq e \geq 1} (z-z_1)^{-e} a_{e1}$$

- Solve for M_2 and a_{e2} in expansion centered at z_2 (= parent)

$$f_1(z) \sim f_2(z) = M_2 * \log(z-z_2) + \Sigma_{r \geq e \geq 1} (z-z_2)^{-e} a_{e2}$$

- Get $M_2 = M_1$ and each a_{e2} is a linear combination of the a_{e1}
  - multiply r-vector of a_{e1} values by a fixed r-by-r matrix to get a_{e2}
- $(M_2 , a_{12} , \ldots , a_{r2} , z_2) =$ Outer_shift$(\text{Outer}(n_1) , z_2)$

$$= \text{desired Outer}(n_2)$$

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

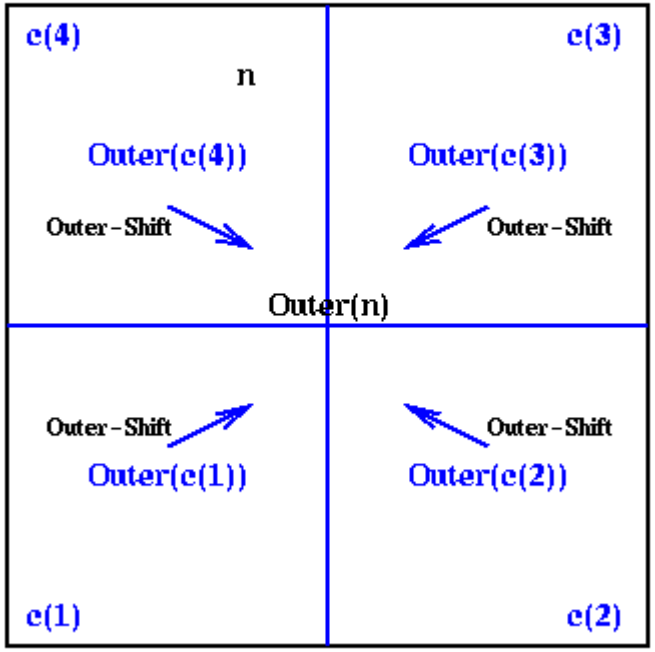
... Compute Outer(n) for each node of the QuadTree
outer = Build_Outer(root)

---

function (M, a_1,...,a_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, a_1,...,a_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)
        ... then add component by component
    endfor
    return Outer(n)
end if

Cost = O(# 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

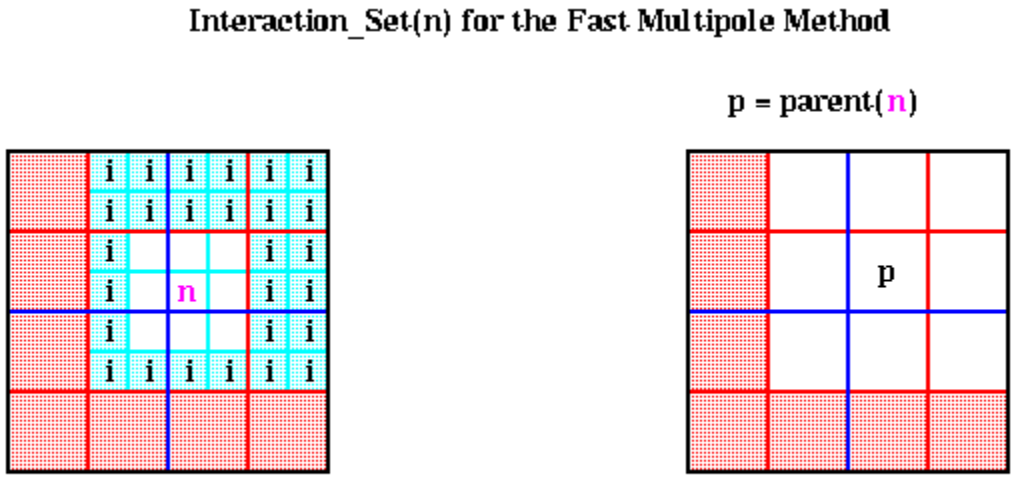
... and combining them

(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

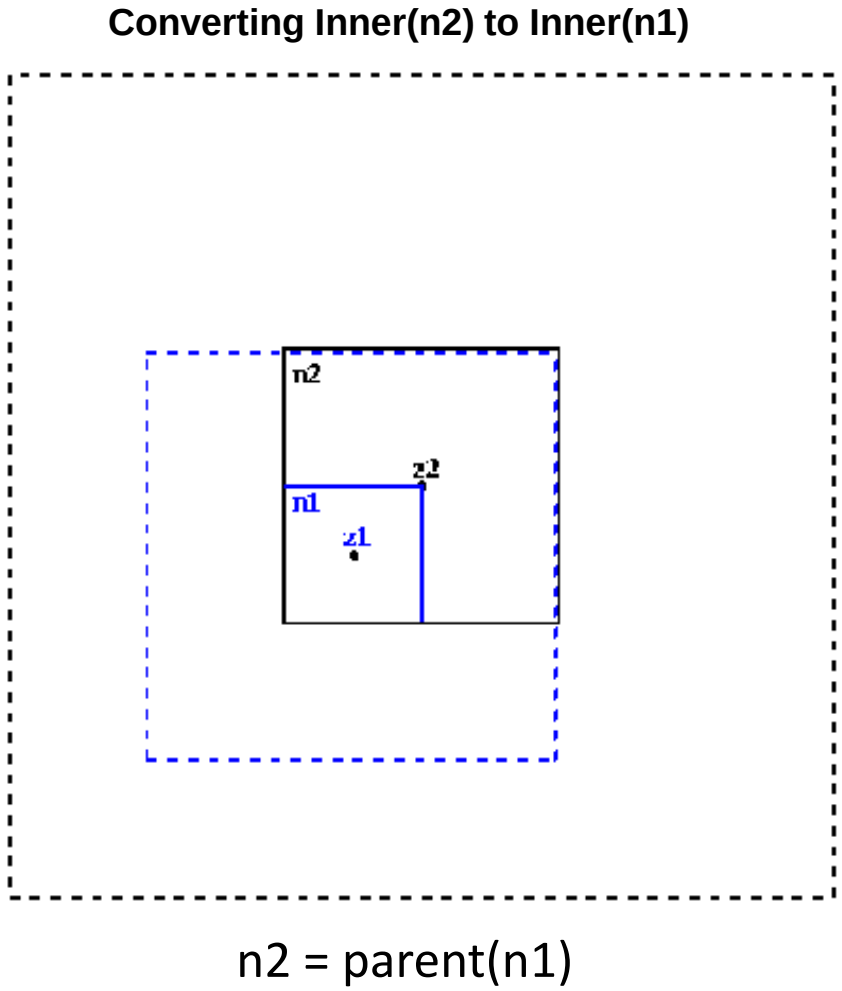
Step 3 of FMM: Computing Inner(n) from other expansions

- Which other expansions?
  - As few as necessary to compute the potential accurately
  - Inner expansion of p = 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)

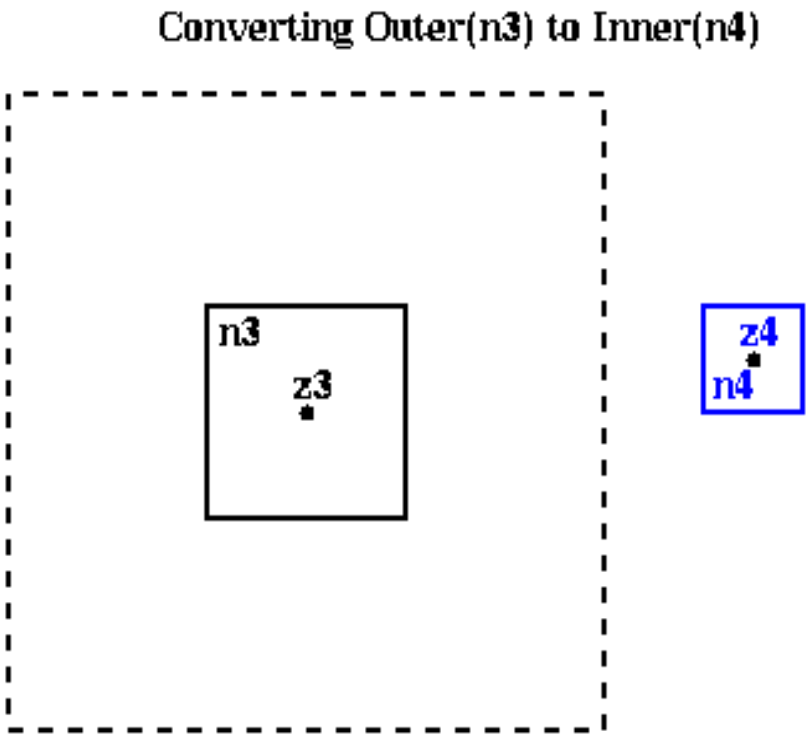


Step 3 of FMM: Compute Inner(n) for each n in QuadTree

- Need Inner(n_1) = Inner_shift(Inner(n_2), n_1)
- Need Inner(n_4) = Convert(Outer(n_3), n_4)



n2 = parent(n1)

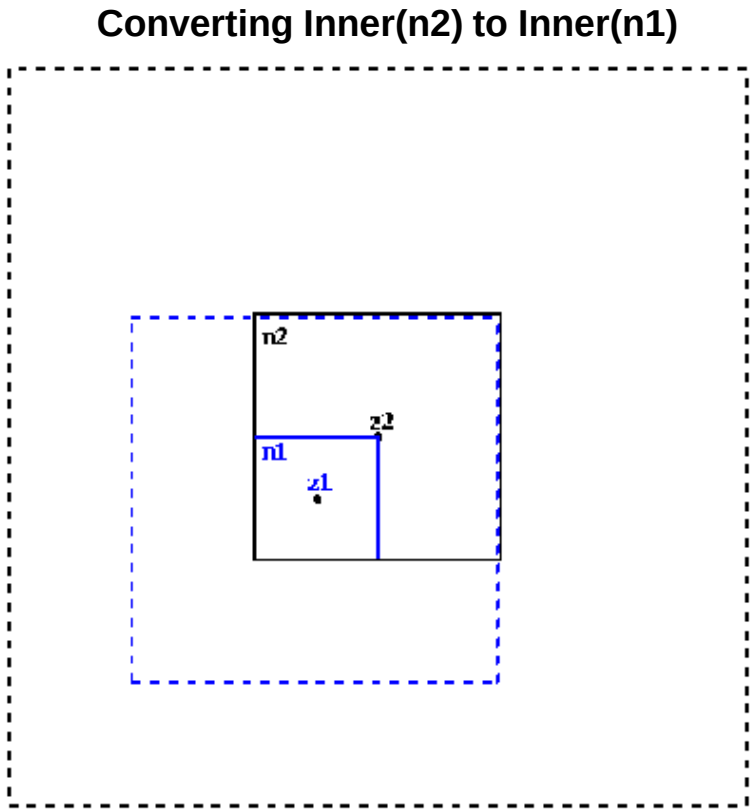


n3 in Interaction_set(n4)

Step 3 of FMM:Inner(n_1) = Inner_shift(Inner(n_2), n_1)

- Inner(n_k) =

  (b_{0k} , b_{1k} , ... , b_{rk} , z_k)



- Inner expansion = $\Sigma_{0 \le e \le r} b_{ek} * (z\text{-}z_k)^e$
- Solve $\Sigma_{0 \le e \le r} b_{e1} * (z\text{-}z_1)^e = \Sigma_{0 \le e \le r} b_{e2} * (z\text{-}z_2)^e$ for b_{e1} given z_1, b_{e2} , and z_2
  - (r+1) x (r+1) matrix-vector multiply

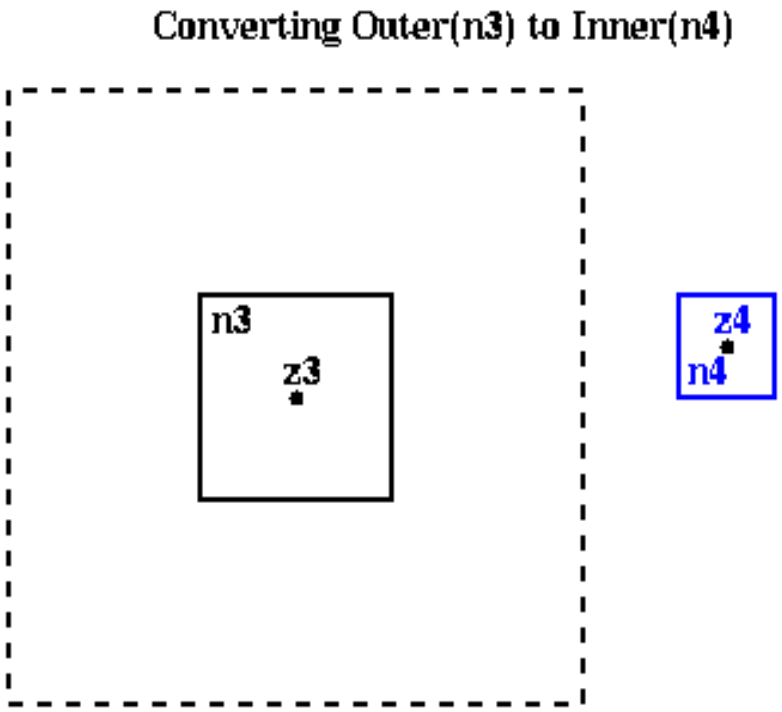
Step 3 of FMM: Inner(n_4) = Convert(Outer(n_3), n_4)

- Inner(n_4) =

  (b_0 , b_1 , ... , b_r , z_4)

- Outer(n_3) =

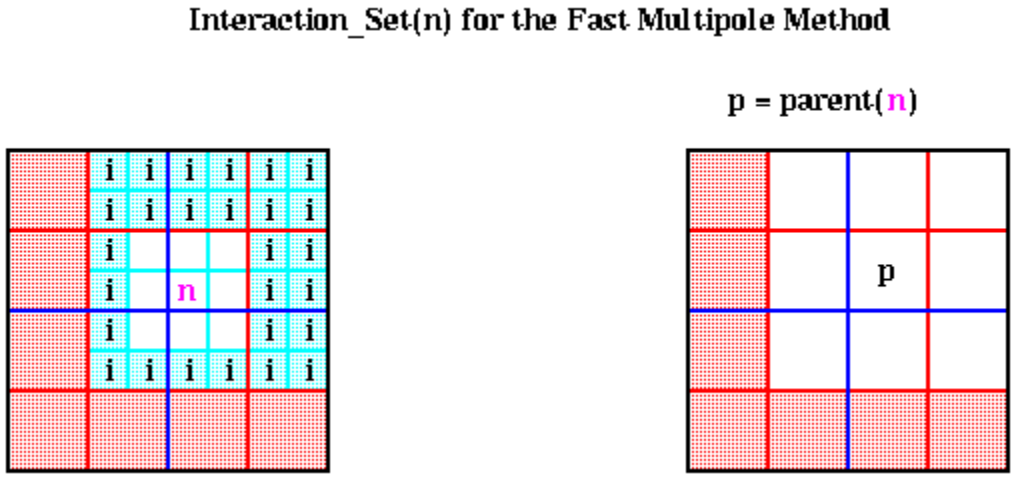
  (M, a_1 , a_2 , ... , a_r , z_3)



- Solve $\Sigma_{0\le e\le r}\ b_e * (z-z_4)^e = M*\log(z-z_3) + \Sigma_{0\le e\le r}\ a_e * (z-z_3)^{-e}$ for b_e given z_4 , a_e , and z_3
  - (r+1) x (r+1) matrix-vector multiply

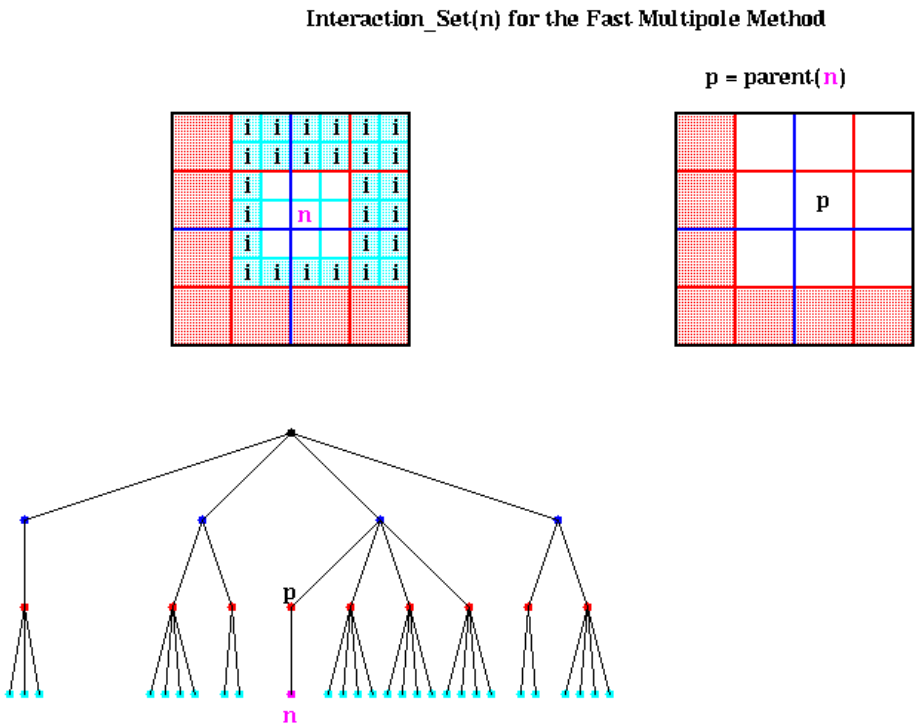
Step 3 of FMM: Computing Inner(n) from other expansions

- 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)



Step 3 of FMM: Interaction Set

- Interaction Set = { nodes i that are children of a neighbor of parent(n), such that i is not itself a neighbor of n}
- For each i in Interaction Set , Outer(i) is available, so that Convert(Outer(i),center(n)) gives contribution to Inner(n) due to particles in i
- Number of i in Interaction Set is at most $6^2 - 3^2 = 27$ in 2D
- Number of i in Interaction Set is at most $6^3 - 3^3 = 189$ in 3D



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 (  b1,...,br , zn) = 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) + Convert( 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(# 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 Inner(n)*
... if 1 node/leaf, then each particles accessed once,
... so cost = O(N)
... final 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

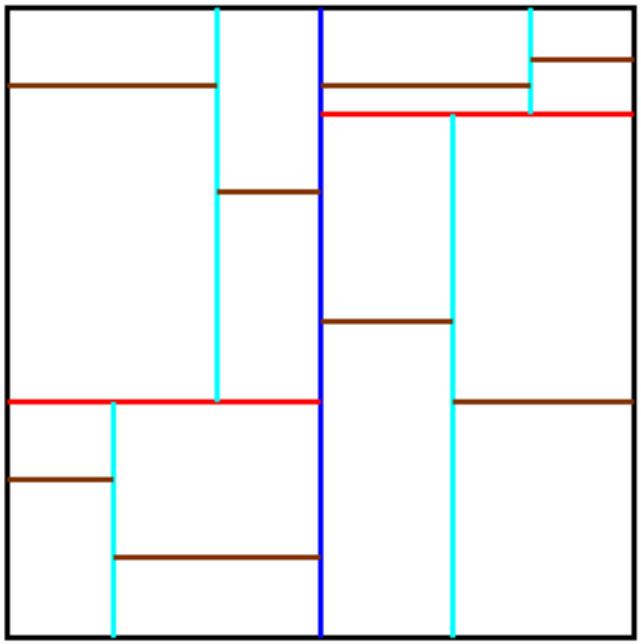
Parallelizing Hierarchical N-Body codes

- One parallelization scheme will work for them all
  - Based on D. Blackston and T. Suel, Supercomputing 97
  - 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)

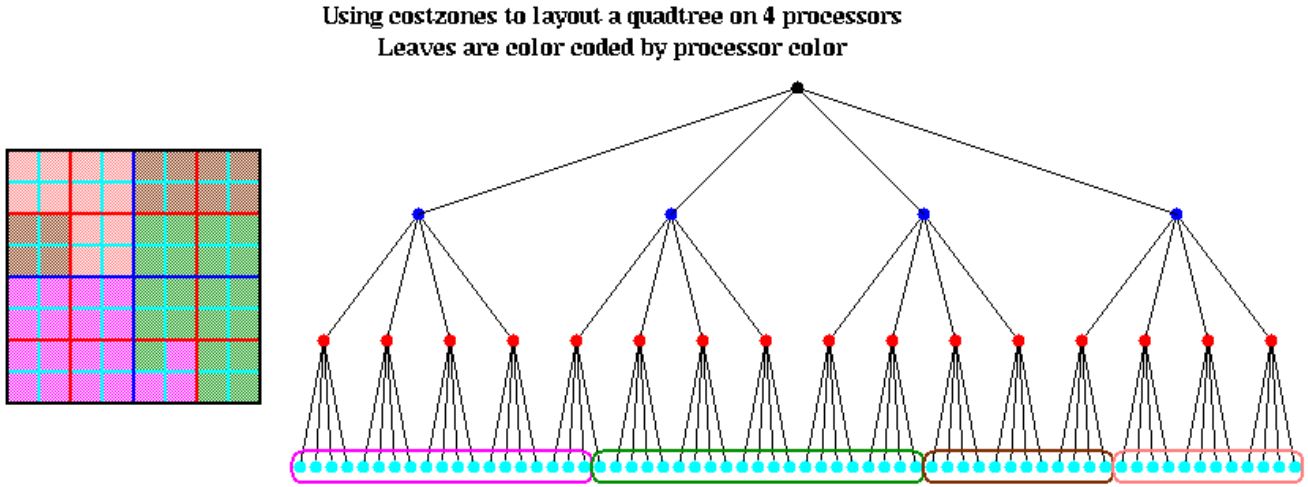
- Warren and Salmon, Supercomputing 92
- Recursively split region along axes into regions containing equal numbers of particles
- Works well for 2D, not 3D

Partitioning for 16 procs:



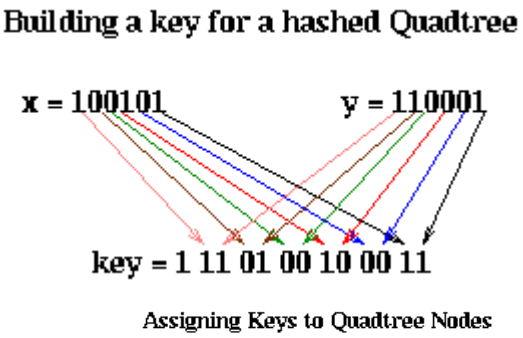
Load Balancing Scheme 2: Costzones

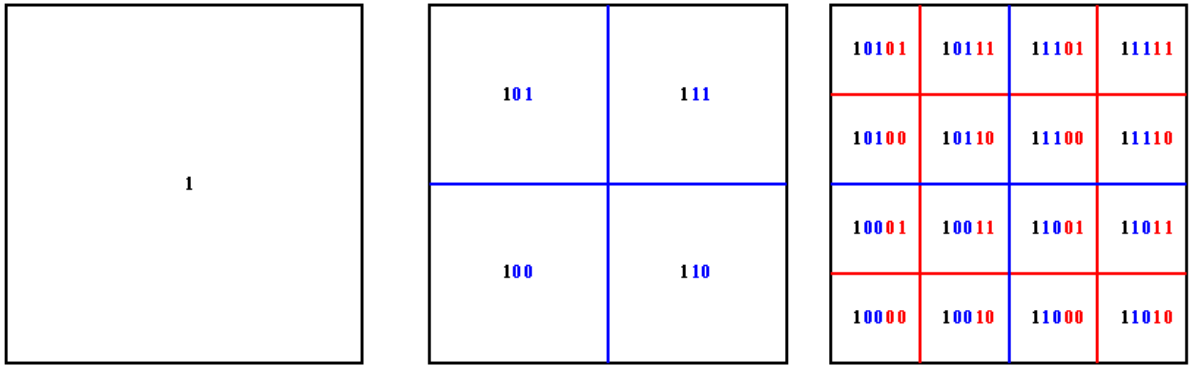
- Called Costzones for Shared Memory
  - PhD thesis, J.P. Singh, Stanford, 1993
- Called "Hashed Oct Tree" for Distributed Memory
  - Warren and Salmon, Supercomputing 93
- We will use the name Costzones for both
- Idea: partition QuadTree instead of space
  - Estimate work for each node, call total 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



Linearly Ordering Quadtree nodes for Costzones (1/2)

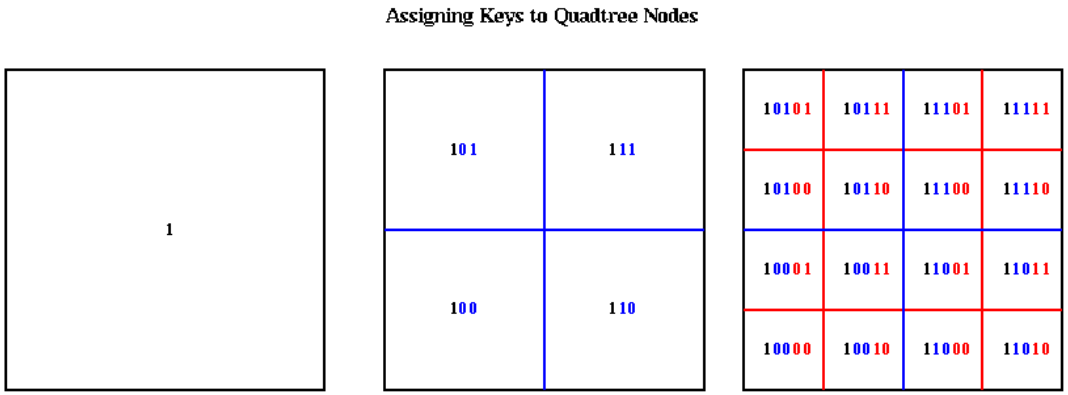
- Hashed QuadTrees (Warren and Salmon)
- Assign unique key to each node in QuadTree, then compute hash(key) to get integers that can be linearly ordered
- If (x,y) are coordinates of center of node, interleave bits to get key
  - Put 1 at left as "sentinel"
  - Nodes near root of tree have shorter keys



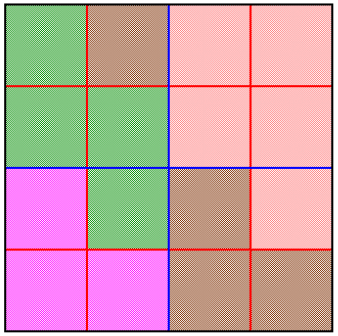


Linearly Ordering Quadtree nodes for Costzones (2/2)

- 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



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

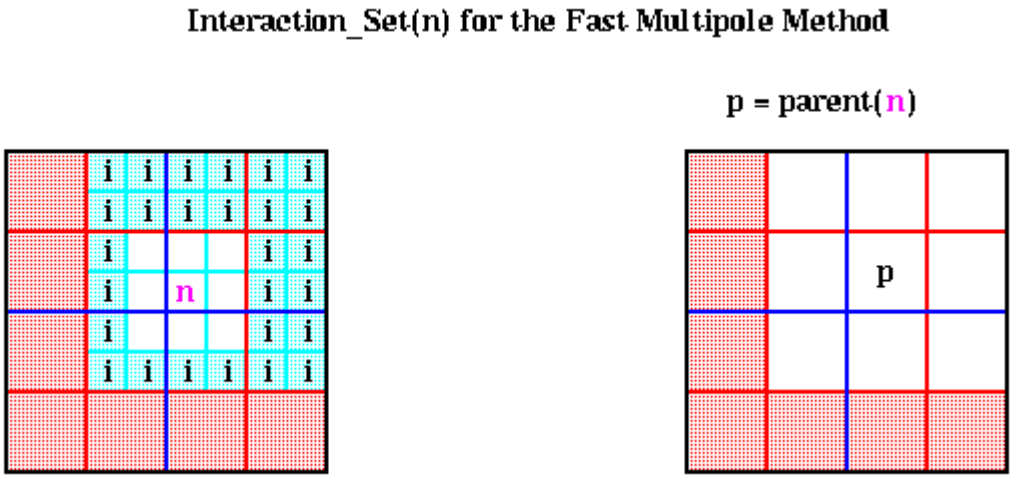
    (1) $D(n)/r(n) < q$ and $D(\text{parent}(n))/r(\text{parent}(n)) \geq q$, or

    (2) $D(n)/r(n) \geq q$

    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)

Recall Step 3 of FMM

- We will use Inner_shift and Convert 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)
  - Convert(Outer(i), center(n)) will account for potential from particles in boxes at same level in Interaction Set (nodes labeled i below)



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

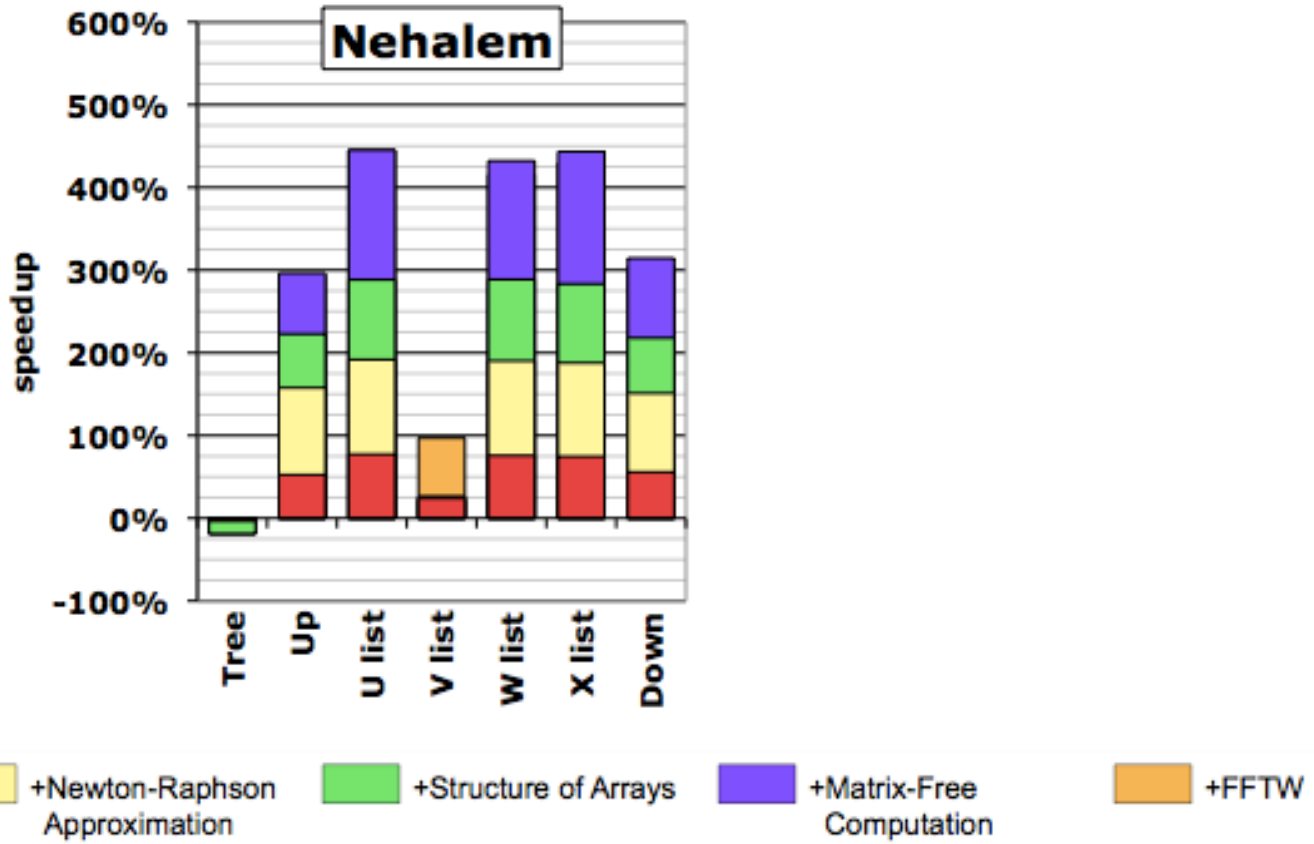
Performance Results - 3

- *Chandramowlishwaran, et al., IPDPS*, 2010
- 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 x 4-cores/socket x 2-thr/core = 16 threads
    - 9.4x on AMD Barcelona
      - 2-sockets x 4-cores/socket x 1-thr/core = 8 threads
    - 37.6x on Sun Victoria Falls
      - 2-sockets x 8-cores/socket x 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*: rsqrtps + 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

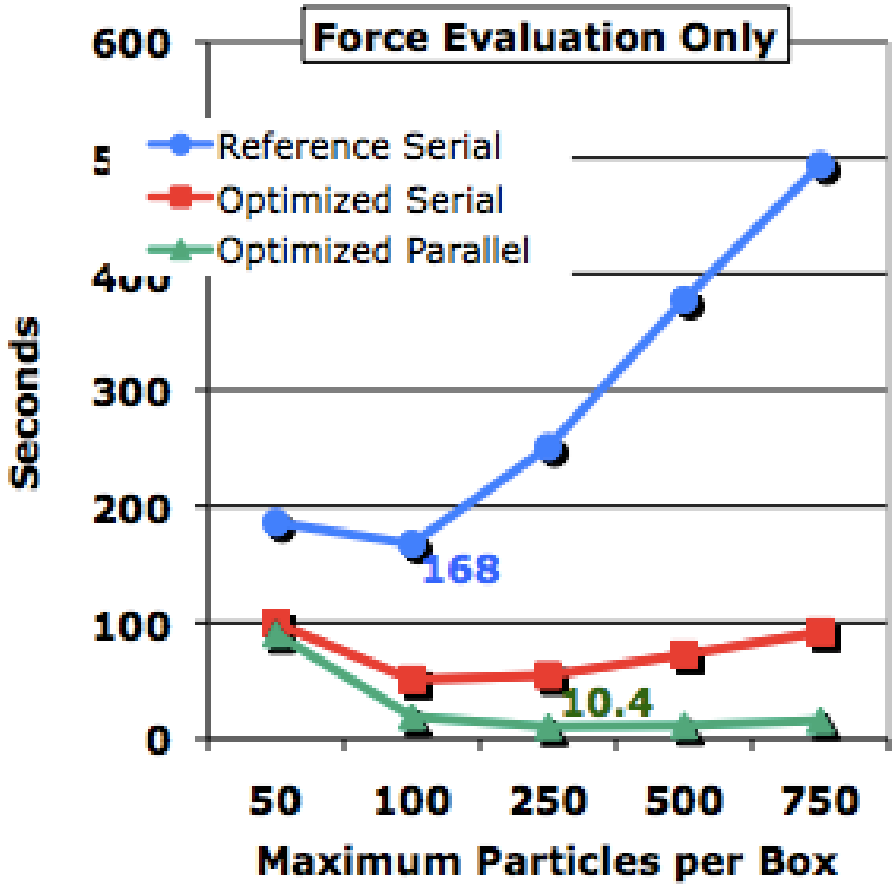
Single-core Optimizations
Double-Precision, Non-uniform (ellipsoidal)



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

Source: Richard Vuduc

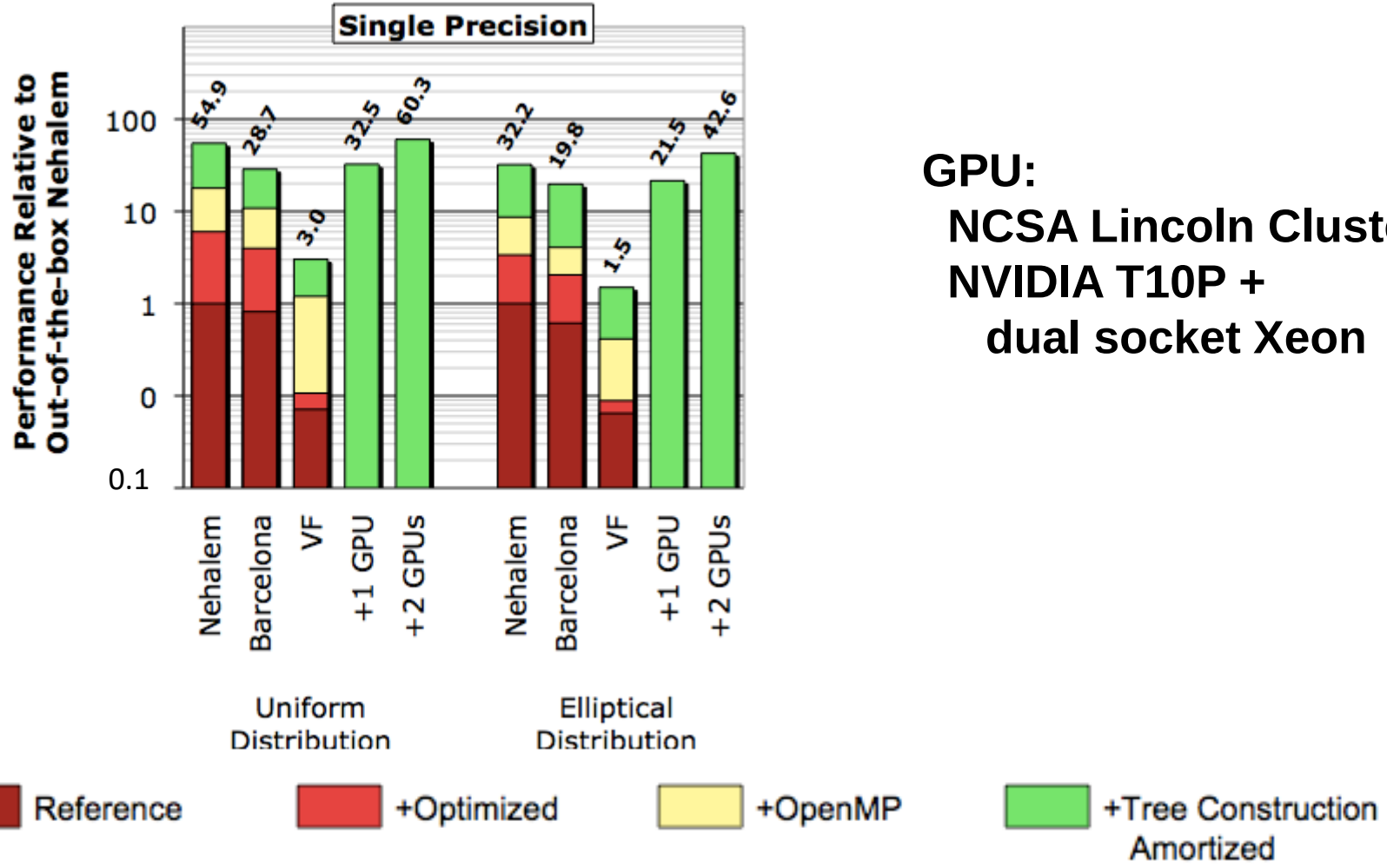
Algorithmic Tuning of q = Max pts / box - Nehalem



Shape of curve changes as optimizations are introduced.

Source: Richard Vuduc

Cross-Platform Performance Comparison (Summary)



GPU:
NCSA Lincoln Cluster
NVIDIA T10P +
dual socket Xeon

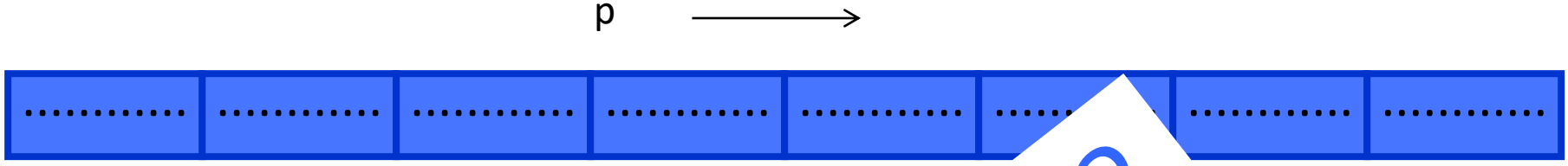
Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.

Source: Richard Vuduc

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 (e.g. cutoff distance), replace n^2 by #interactions in Thm
  - Attainable when $n/p <= M <= n/p^{1/2}$
    - $M = n/p$: #words = n, #messages = p
    - $M = n/p^{1/2}$: #words = $n/p^{1/2}$, #messages = O(1) (12x speedups)

Traditional (Naïve n^2) Nbody Algorithm (using a 1D decomposition)

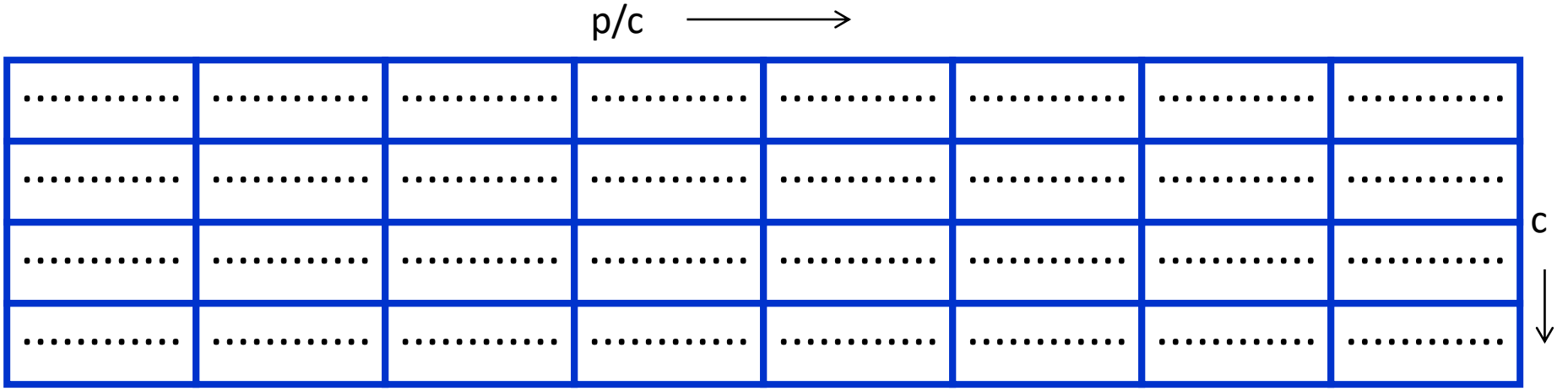


- Given n particles, p processors, M=O(
- Each processor has n/p particles
- Algorithm: shift copy of partic
- Computation cost: $n^2/$
- Communication b
  - Lower bound =
- Communica
  - Lower bound

Can we do better?

Communication Avoiding Version (using a "1.5D" decomposition: assume memory for c copies)

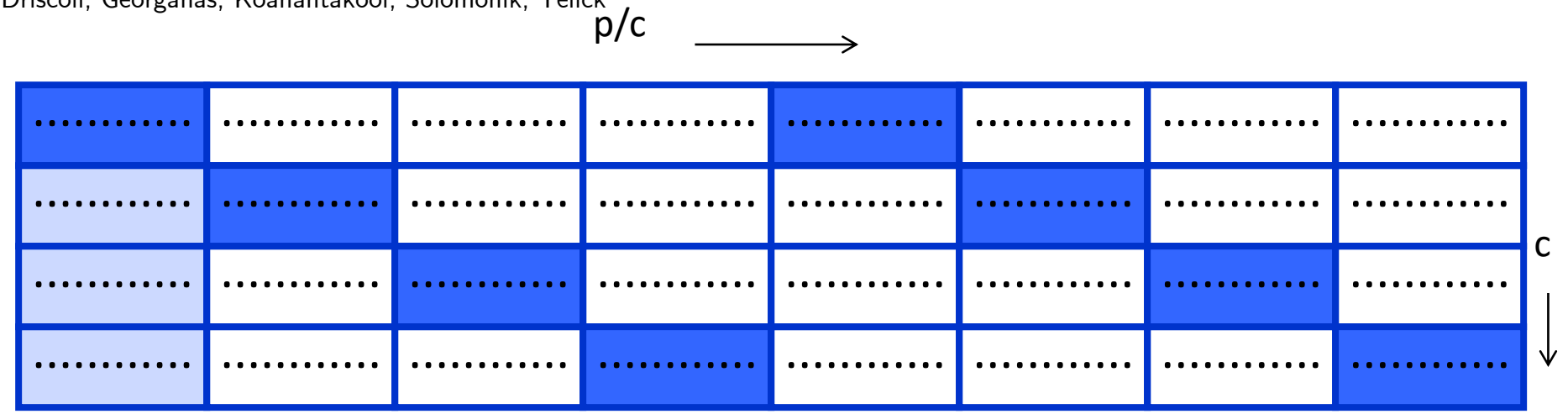
Driscoll, Georganas, Koanantakool, Solomonik, Yelick



- Divide p into c groups. Start with all n particles on p/c processors
- Make a copy of each group of n*c/p particles
- Pass copy to the 0^{th}...c-1^{st} neighbor depending on row
- Main Algorithm: for p/c^2 steps
  - Compute pairwise interactions for owned vs. shifted particles
  - Shift copy of n*c/p particles to c^{th} neighbor
- Reduce across c to produce final value for each particle

Communication Avoiding Version (using a "1.5D" decomposition: assume memory for c copies)

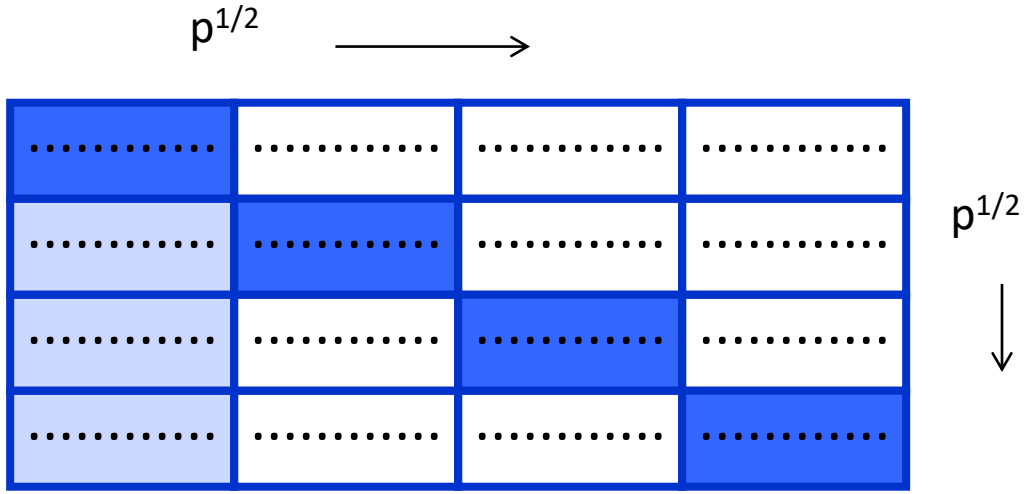
Driscoll, Georganas, Koanantakool, Solomonik, Yelick



- Divide p into c groups. Replicate particles in each group.
  - Memory: M = O(n*c/p) particles per processor
- Make, pass copies: Latency: O(log c) Bandwidth: O(n*c/p)
- Main Algorithm: for p/c^2 steps
  - Per step, Latency: O(1) Bandwidth: O(n*c/p)
  - Overall, Latency: $O(p/c^2) = O((n^2/p)/M^2)$
    Bandwidth: $O(n/c) = O((n^2/p)/M)$
- Attains Bandwidth, latency lower bound for $1 \leq c \leq p^{1/2}$

Communication Avoiding Version (2D decomposition is Limit)

Driscoll, Georganas, Koanantakool, Solomonik, Yelick



- Limit is when $c = p^{1/2}$
  - Memory: $M = O(n/p^{1/2})$
  - Startup/Finish: Latency: $O(\log c) = O(\log p)$;
    Bandwidth $O(n/p^{1/2})$
- Main part of Algorithm has 1 step
  - Latency: $O(1)$
  - Bandwidth: $O(n/p^{1/2})$

N-Body Speedups on IBM-BG/P (Intrepid)
8K cores, 32K particles

K. Yelick, E. Georganas, M. Driscoll, P. Koanantakool, E. Solomonik