CS 267 Applications of Parallel Computers

Hierarchical Methods for the N-Body problem

James Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr14

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

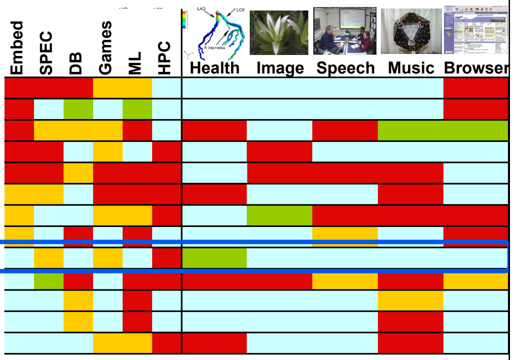
Particle Simulation

```
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
  - 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...

What do commercial and CSE applications have in common?

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



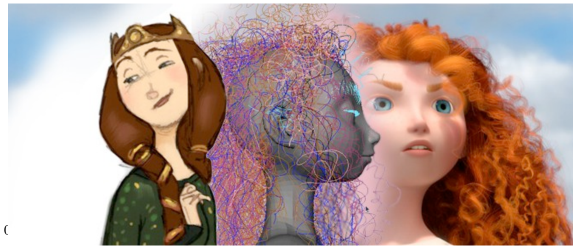
	Embed	SPEC	DB	Games	ML	HPC	Health	Image	Speech	Music	Browser
1 Finite State Mach.	🟥	🟥	🟥	🟨	🟨	🟦	🟦	🟦	🟦	🟦	🟥
2 Combinational	🟥	🟦	🟩	🟦	🟩	🟦	🟦	🟦	🟦	🟦	🟥
3 Graph Traversal	🟥	🟨	🟨	🟨	🟥	🟦	🟥	🟦	🟥	🟩	🟩
4 Structured Grid	🟥	🟥	🟦	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦
5 Dense Matrix	🟥	🟥	🟨	🟥	🟥	🟥	🟦	🟥	🟥	🟥	🟦
6 Sparse Matrix	🟨	🟨	🟦	🟥	🟥	🟥	🟥	🟦	🟦	🟥	🟦
7 Spectral (FFT)	🟨	🟦	🟦	🟨	🟨	🟥	🟦	🟩	🟥	🟥	🟥
8 Dynamic Prog	🟨	🟦	🟥	🟦	🟥	🟦	🟦	🟦	🟨	🟦	🟥
9 N-Body	🟦	🟨	🟦	🟨	🟦	🟥	🟩	🟦	🟦	🟦	🟦
10 MapReduce	🟦	🟩	🟥	🟦	🟥	🟥	🟥	🟥	🟨	🟥	🟨
11 Backtrack/ B&B	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟨	🟦
12 Graphical Models	🟦	🟦	🟨	🟦	🟥	🟦	🟦	🟦	🟦	🟥	🟦
13 Unstructured Grid	🟦	🟦	🟦	🟨	🟨	🟥	🟥	🟦	🟦	🟥	🟦

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. Yasoki et al
    - Gordon Bell Prize for Price/Performance at Supercomputing 2009
  - Sustained 20 Teraflops, or $8/Gigaflop

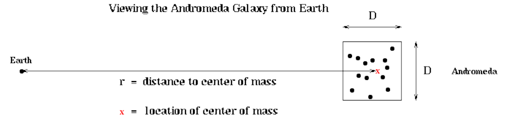
Applications (2/2)

- Molecular Dynamics
- Plasma Simulation
- Electron-Beam Lithography Device Simulation
- Hair …
  - www.fxguide.com/featured/brave-new-hair/
  - 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 ≥ 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"

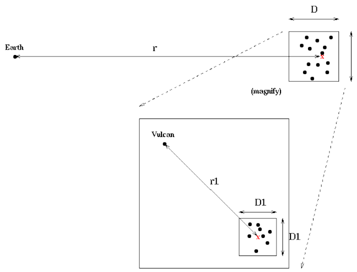


  - Idea not new: Newton approximated earth and falling apple by CMs

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

Replacing Clusters by their Centers of Mass Recursively



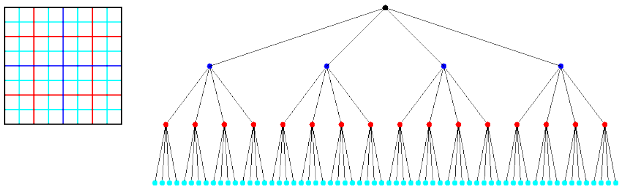
Outline

- 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

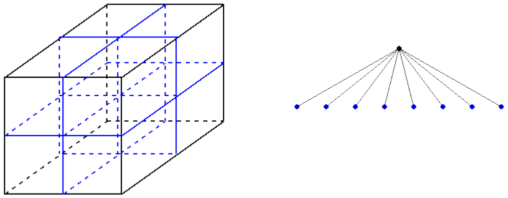
A Complete Quadtree with 4 Levels



Oct Trees

- Similar Data Structure to subdivide space

2 Levels of an Octree

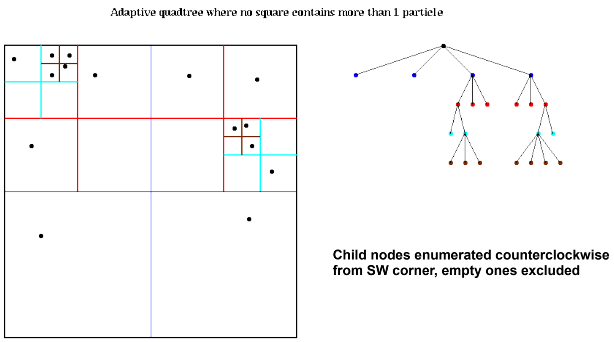


Using Quad Trees and Oct Trees

- 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

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 Constrution

- 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:**

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

~ 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)
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

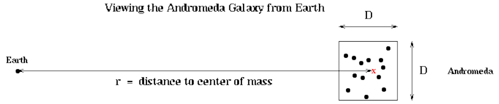
```
… Compute the CM = Center of Mass and TM = Total Mass of all the particles
… in each node of the QuadTree
( TM, CM ) = Compute_Mass( root )

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

Viewing the Andromeda Galaxy from Earth



r = distance to center of mass

x = location of center of mass

Computing force on a particle due to a node

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

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

Details of Step 3 of BH

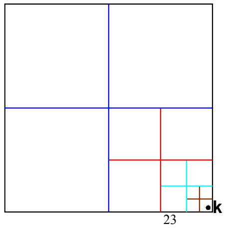
```
… for each particle, traverse the QuadTree to compute the force on it
for k = 1 to N
  f(k) = TreeForce( k, root )
         … compute force on particle k due to all particles inside root (except k)
endfor
```

```
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 formula on last slide
  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 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 $\theta>1$):
    - For each undivided node = square, (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(level of k)
  - Total cost = $O(\Sigma_k$ level of k) = O(N log N)
    - Strongly depends on θ

Sample Barnes-Hut Force calculation
For particle in lower right corner
Assuming theta > 1



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 = $-\text{grad}\ \phi(x,y,z) = -(d\phi/dx\ ,\ d\phi/dy\ ,\ d\phi/dz) = -(x,y,z)/r^3$
- In 2D with x,y coordinates
  - Potential = $\phi(x,y) = \log r = \log (x^2 + y^2)^{1/2}$
  - Force = $-\text{grad}\ \phi(x,y) = -(d\phi/dx\ ,\ d\phi/dy\ ) = -(x,y)/r^2$
- 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 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) (1/2)

$\phi(z)$ = 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 $\log z = \log |z|e^{i\theta} = \log |z| + i\theta$

… 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

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

… where $\alpha_e = \Sigma_k\, m_k\, z_k^e/e$ … called Multipole Expansion

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

$\phi(z)$ = 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

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

… where $M = \Sigma_k\, m_k$ = Total Mass and

… $\alpha_e = \Sigma_k\, m_k\, z_k^e /e$

… This is called a Multipole Expansion in z

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

… r = number of terms in Truncated Multipole Expansion

… and error(r) = $-\Sigma_{r<e} z^{-e}\, \alpha_e$

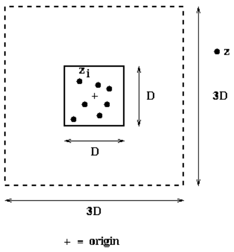
- Note that $\alpha_1 = \Sigma_k\, m_k\, z_k = CM*M$
  so that M and α_1 terms have same info as Barnes-Hut

- 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

Error outside larger box is O(c^(-r))



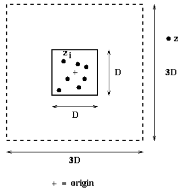
Outer(n) and Outer Expansion

$\phi(z) \sim M * \log(z - z_n) - \Sigma_{r \ge e \ge 1} (z-z_n)^{-e} \alpha_e$

- Outer(n) = (M, α_1 , α_2 , … , α_r , z_n)
  - Stores data for evaluating potential $\phi(z)$ outside node n due to particles inside n
  - z_n = center of node n
  - Error small for z outside dotted line in previous plot
  - Cost of evaluating $\phi(z)$ is O(r), independent of the number 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
  - M and α_1 same information as 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
  - $\Sigma_{0 \le e \le r} \beta_e * (z-z_n)^e$
  - z_n = center of node n, a D-by-D box
  - Inner(n) = (β_0 , β_1 , … , β_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

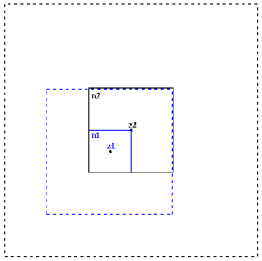
(3) Call Build_ Inner(root), to compute inner expansions of each node n in the QuadTree
- … Traverse QuadTree from top to bottom,
- … converting outer to inner expansions
- … and combining them

(4) For each leaf node n, add contributions of nearest particles directly into 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?
  - $\phi_k(z) \sim M_k * \log(z-z_k) - \Sigma_{r \geq e \geq 1} (z-z_k)^{-e} \alpha_{ek}$ 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 α_k and center z_2 which represents the same potential as Outer(n_1) outside the black dashed box

Using Outer_Shift to convert Outer(n1) to Outer(n2)



Outer_shift: Details (2/3)

- Given expansion centered at z_1 (= child)

  $\phi_1(z) = M_1 * \log(z-z_1) + \Sigma_{r \geq e \geq 1} (z-z_1)^{-e} \alpha_{e1}$
- Solve for M_2 and α_{e2} in expansion centered at z_2 (= parent)

  $\phi_1(z) \sim \phi_2(z) = M_2 * \log(z-z_2) + \Sigma_{r \geq e \geq 1} (z-z_2)^{-e} \alpha_{e2}$
- Get $M_2 = M_1$ and each α_{e2} is a linear combination of the α_{e1}
  - multiply r-vector of α_{e1} values by a fixed r-by-r matrix to get α_{e2}
- (M_2 , α_{12} , … , α_{r2} , z_2) = Outer_shift(Outer(n_1) , z_2)

  = desired Outer(n_2)

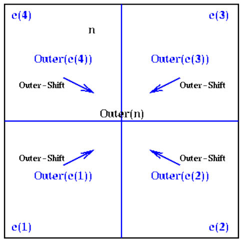
Step 2 of FMM: compute Outer(n) for each node n in QuadTree (3/3)

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

function ( M, $\alpha_1$,…,$\alpha_r$ , $z_n$) = 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, $\alpha_1$,…,$\alpha_r$ , $z_n$) 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))
             … just 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

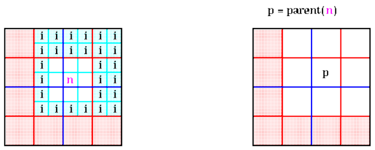
(4) For each leaf node n, add contributions of nearest particles directly into Inner(n)

- … 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)

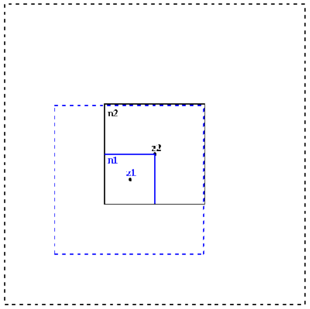
Interaction_Set(n) for the Fast Multipole Method



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

- Need Inner(n_1) = Inner_shift(Inner(n_2), n_1)

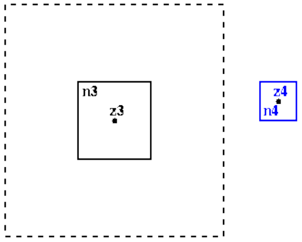
Converting Inner(n2) to Inner(n1)



n2 = parent(n1)

- Need Inner(n_4) = Convert(Outer(n_3), n_4)

Converting Outer(n3) to Inner(n4)



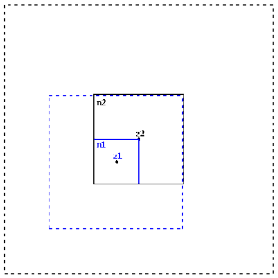
n3 in Interaction_set(n4)

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

- Inner(n_k) =

  (β_{0k} , β_{1k} , … , β_{rk} , z_k)

Converting Inner(n2) to Inner(n1)



- Inner expansion = $\Sigma_{0 \le e \le r}\, \beta_{ek} * (z-z_k)^e$
- Solve $\Sigma_{0 \le e \le r}\, \beta_{e1} * (z-z_1)^e = \Sigma_{0 \le e \le r}\, \beta_{e2} * (z-z_2)^e$

  for β_{e1} given z_1, β_{e2} , and z_2
  - (r+1) x (r+1) matrix-vector multiply

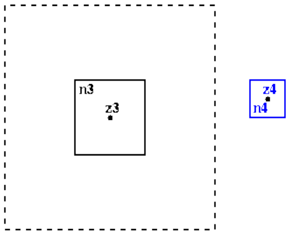
Step 3 of FMM: Inner(n_4) = Convert(Outer(n_3), n_4)

- Inner(n_4) =

  (β_0 , β_1 , … , β_r , z_4)
- Outer(n_3) =

  (M, α_1 , α_2 , … , α_r , z_3)

Converting Outer(n3) to Inner(n4)



- Solve $\Sigma_{0 \le e \le r}\, \beta_e * (z-z_4)^e = M*\log(z-z_3) + \Sigma_{0 \le e \le r}\, \alpha_e * (z-z_3)^{-e}$

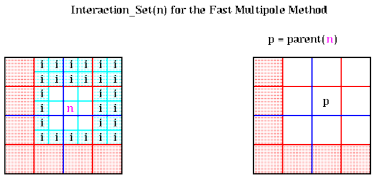
  for β_e given z_4 , α_e , and z_3
  - (r+1) x (r+1) matrix-vector multiply

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

° We will use Inner_shift and Convert to build each Inner(n) by combining expansions from other nodes

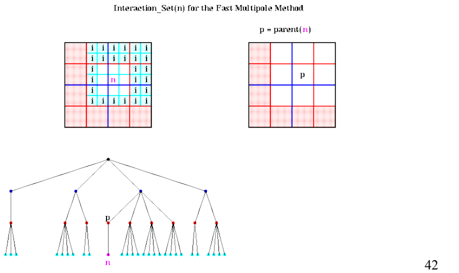
° Which other nodes?

- 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)


Interaction_Set(n) for the Fast Multipole Method

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


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 ( $\beta_1,…,\beta_r$ , $z_n$) = 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) + 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 Hierachical 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
- 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

---

Programming Model - BSP

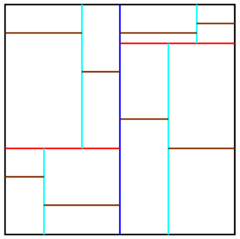
- BSP Model = Bulk Synchronous Programming Model
  - All processors compute; barrier; all processors communicate; barrier; repeat
- Advantages
  - easy to program (parallel code looks like serial code)
  - easy to port (MPI, shared memory, TCP network)
- Possible disadvantage
  - Rigidly synchronous style might mean inefficiency?
- OK with few processors; communication costs low
  - 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 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

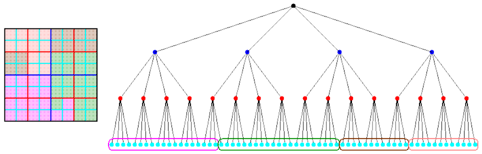
Partitioning for 16 procs:



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 (lots of choices)
  - Assign contiguous blocks of nodes with work W/p to processors
  - Works well in 3D

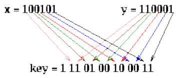
Using costzones to layout a quadtree on 4 processors
Leaves are color coded by processor color



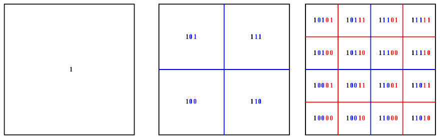
Linearly Ordering Quadtree nodes for Costzones (1/2)

- 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

Building a key for a hashed Quadtree



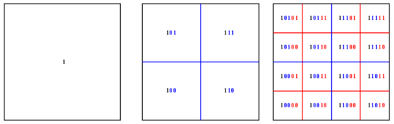
Assigning Keys to Quadtree Nodes



Linearly Ordering Quadtree nodes for Costzones (2/2)

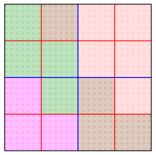
- Assign unique key to each node in QuadTree, then compute hash(key) to get a linear order
- key = interleaved bits of x,y coordinates of node, prefixed by 1

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
- 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 $D(parent(n))/r(parent(n)) \geq \theta$, 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

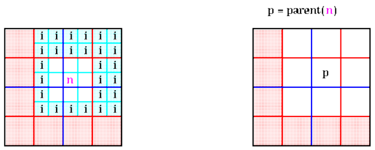
Recall Step 3 of FMM

° 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)

Interaction_Set(n) for the Fast Multipole Method



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
  - 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, multithreading, mixed-precision, 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

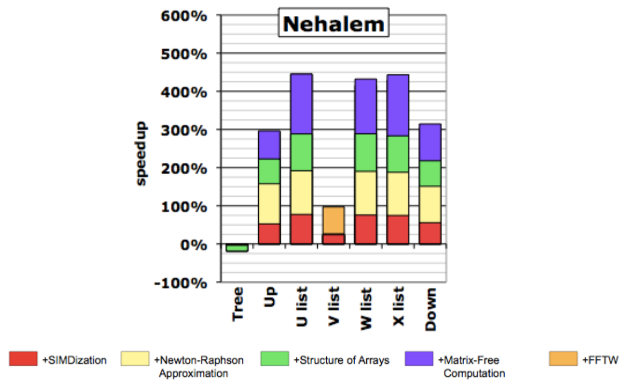
Source: Richard Vuduc

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
  - 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*

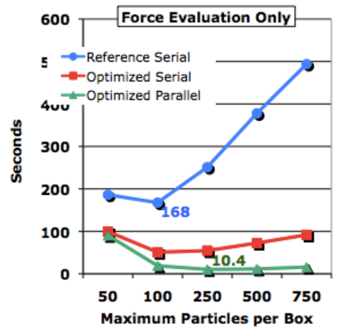
Source: Richard Vuduc

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



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

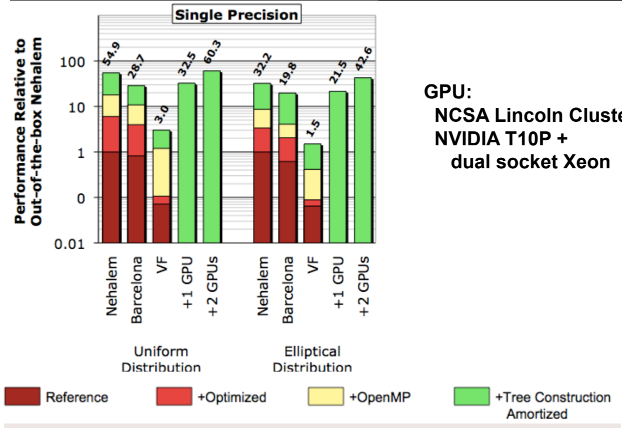
Algorithmic Tuning of *q* = Max pts / box - Nehalem



Shape of curve changes as we introduce optimizations.

Source: Richard Vuduc

Cross-Platform Performance Comparison (Summary)



GPU:
NCSA Lincoln Cluster
NVIDIA T10P +
dual socket Xeon

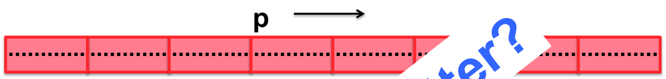
Nehalem outperforms 1-GPU case, a little slower than 2-GPU case.

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 (eg cutoff distance), replace n^2 by #interactions in Thm**
  - **For which values of M is this attainable?**

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

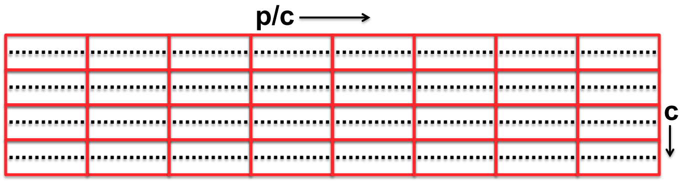


- Given n particles, p processors memory
- Each processor has n/p part
- Algorithm: shift copy of o the left p times, calculating all pairw
- Computation co
- Communica dth: O(n) words
  - Lower b p)/M) = Ω(n) , attained
- Communica atency: O(p) messages
  - Lower bound = $\Omega((n^2/p)/M^2) = \Omega(p)$, attained

Can we do better?

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

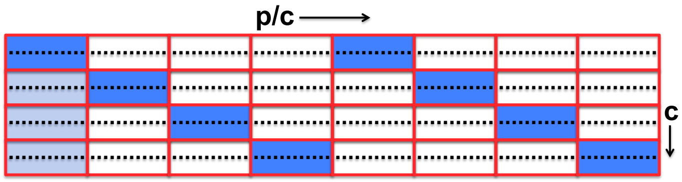
Driscoll, Georganas, Koanantakool, Solomonik, Yelick



- 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 0th…c-1st 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 cth neighbor
- Reduce across c to produce final value for each particle

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

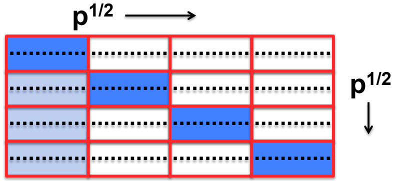
Driscoll, Georganas, Koanantakool, Solomonik, Yelick



- 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 $1 \le c \le p^{1/2}$

Communication Avoiding Version (2D decomposition is Limit)

Driscoll, Georganas, Koanantakool, Solomonik, Yelick

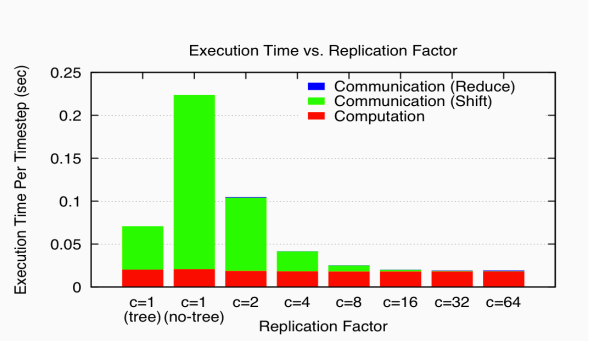


- Limit is when $c = p^{1/2}$
  - Memory: $M = O(n/p^{1/2})$
  - Startup/Finish: Latency: O(log c) = O(log p);
    Bandwidth $O(n/p^{1/2})$
- Main part of Algorithm has 1 step
  - Latency: O(1) Bandwidth: $O(n/p^{1/2})$

Same as "parallelizing in the force direction" in NAMD [Plimpton95]

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

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



11.8x speedup

How general are these communication lower bounds and optimal algorithms?

Recall optimal sequential Matmul

- Naïve code

    for i=1:n, for j=1:n, for k=1:n, C(i,j)+=A(i,k)*B(k,j)

- “Blocked” code

```
for i1 = 1:b:n,  for j1 = 1:b:n,   for k1 = 1:b:n
  for i2 = 0:b-1,  for j2 = 0:b-1,   for k2 = 0:b-1
    i=i1+i2,  j = j1+j2,  k = k1+k2
    C(i,j)+=A(i,k)*B(k,j)
```

b x b matmul

- Thm: Picking b = $M^{1/2}$ attains lower bound:

    #words_moved = $\Omega(n^3/M^{1/2})$

- Where does 1/2 come from?

New Thm applied to Matmul

- for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
- Record array indices in matrix Δ

$$\Delta = \begin{array}{cccc} i & j & k & \\ 1 & 0 & 1 & A \\ 0 & 1 & 1 & B \\ 1 & 1 & 0 & C \end{array}$$

- Solve LP for x = $[xi,xj,xk]^T$: max $1^T x$ s.t. $\Delta\, x \le 1$
    - Result: x = $[1/2, 1/2, 1/2]^T$, $1^T x = 3/2 = S$
- Thm: #words_moved = $\Omega(n^3/M^{S-1}) = \Omega(n^3/M^{1/2})$

    Attained by block sizes $M^{xi},M^{xj},M^{xk} = M^{1/2},M^{1/2},M^{1/2}$

New Thm applied to Direct N-Body

- for i=1:n, for j=1:n, F(i) += force(P(i) , P(j))
- Record array indices in matrix Δ

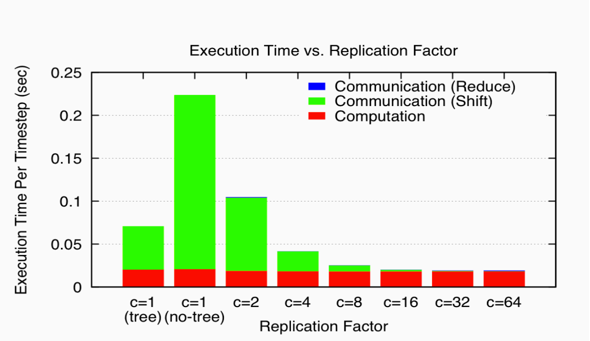
$$\Delta = \begin{array}{ccl} i & j & \\ 1 & 0 & F \\ 1 & 0 & P(i) \\ 0 & 1 & P(j) \end{array}$$

- Solve LP for $x = [xi,xj]^T$: max $1^T x$ s.t. $\Delta x \le 1$
  - Result: $x = [1,1]$, $1^T x = 2 = S$
- Thm: #words_moved = $\Omega(n^2/M^{S-1}) = \Omega(n^2/M^1)$

  Attained by block sizes $M^{xi}, M^{xj} = M^1, M^1$

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

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



11.8x speedup

New Thm applied to Random Code

- for i1=1:n, for i2=1:n, … , for i6=1:n

  A1(i1,i3,i6) += func1(A2(i1,i2,i4),A3(i2,i3,i5),A4(i3,i4,i6))

  A5(i2,i6) += func2(A6(i1,i4,i5),A3(i3,i4,i6))
- Record array indices in matrix Δ

$$\Delta = \begin{array}{cccccccl} i1 & i2 & i3 & i4 & i5 & i6 & \\ 1 & 0 & 1 & 0 & 0 & 1 & A1 \\ 1 & 1 & 0 & 1 & 0 & 0 & A2 \\ 0 & 1 & 1 & 0 & 1 & 0 & A3 \\ 0 & 0 & 1 & 1 & 0 & 1 & A3,A4 \\ 0 & 0 & 1 & 1 & 0 & 1 & A5 \\ 1 & 0 & 0 & 1 & 1 & 0 & A6 \end{array}$$

- Solve LP for $x = [x1,\ldots,x6]^T$: max $1^T x$ s.t. $\Delta x \le 1$
  - Result: $x = [2/7,3/7,1/7,2/7,3/7,4/7]$, $1^T x = 15/7 = s$
- Thm: #words_moved = $\Omega(n^6/M^{S-1}) = \Omega(n^6/M^{8/7})$

  Attained by block sizes $M^{2/7},M^{3/7},M^{1/7},M^{2/7},M^{3/7},M^{4/7}$

Approach to generalizing lower bounds

- Matmul

  for i=1:n, for j=1:n, for k=1:n,

  C(i,j)+=A(i,k)*B(k,j)

  => for (i,j,k) in S = subset of Z^3

  Access locations indexed by (i,j), (i,k), (k,j)
- General case

  for i1=1:n, for i2 = i1:m, … for ik = i3:i4

  C(i1+2*i3-i7) = func(A(i2+3*i4,i1,i2,i1+i2,…),B(pnt(3*i4)),…)

  D(something else) = func(something else), …

  => for (i1,i2,…,ik) in S = subset of Z^k

  Access locations indexed by “projections”, eg

  φ_C (i1,i2,…,ik) = (i1+2*i3-i7)

  φ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

General Communication Bound

- Def: Hölder-Brascamp-Lieb Linear Program (HBL-LP)

  for $s_1,\ldots,s_m$:

  for all subgroups $H < Z^k$, $\mathrm{rank}(H) \le \Sigma_j\, s_j{*}\mathrm{rank}(\varphi_j(H))$
- Thm: Given a program with array refs given by φ_j, choose s_j to minimize $s_{HBL} = \Sigma_j\, s_j$ subject to HBL-LP. Then

  #words_moved = Ω (#iterations/$M^{s_{HBL}-1}$)
  - Proof depends on recent result in pure mathematics by Christ/Tao/Carbery/Bennett

Is this bound attainable? (1/2)

- But first: Can we write it down?
  - One inequality per subgroup $H < Z^k$, but still finitely many!
  - Thm: (bad news) Writing down all inequalities equivalent to Hilbert's 10th problem over Q
    - conjectured to be undecidable
  - Thm: (good news) Can decidably write down a subset of the constraints with the same solution s_{HBL}
  - Thm: (better news) Can write it down explicitly in many cases of interest
    - Ex: when all φ_j = {subset of indices}

Is this bound attainable? (2/2)

- Depends on loop dependencies
- Best case: none, or reductions (matmul)
- Thm: When all φ_j = {subset of indices}, dual of HBL-LP gives optimal tile sizes:

  HBL-LP: minimize $1^T{*}s$ s.t. $s^T{*}\Delta \ge 1^T$

  Dual-HBL-LP: maximize $1^T{*}x$ s.t. $\Delta{*}x \le 1$

  Then for sequential algorithm, tile i_j by M^{x_j}
- Ex: Matmul: $s = [\ 1/2\ ,\ 1/2\ ,\ 1/2\ ]^T = x$
- Extends to unimodular transforms of indices

Intuition behind LP for matmul

- for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
- for i1= 1:M^{xi}:n, for j1=1:M^{xj}:n, for k1=1:M^{xk}:n

  for i2 = 0: M^{xi} -1, for j2 = 0: M^{xj} -1, for k2=0: M^{xk} -1

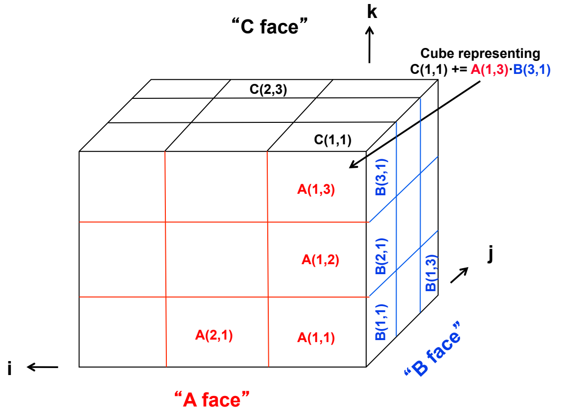
  C(i1+i2, j1+j2) += A(i1+i2,k1+k2)*B(k1+k2,j1+j2)
- How do we choose x = [xi,xj,xk]?
  - C(i,j) has blocks of size M^{xi} by M^{xj}, or M^{xi+xj} words, so $xi + xj \le 1$ to fit in fast memory of size M
  - Similarly A(i,k) requires $xi + xk \le 1$, B(k,j) requires $xk + xj \le 1$
  - Same as $\Delta\ x \le 1$
  - Number of inner 3 loop iterations = M^{xi} x M^{xj} x M^{xk} = $M^{xi + xj + xk}$
  - Goal: maximize number of inner 3 loop iterations given blocks of A,B,C in fast memory
  - Same as maximizing $s = xi + xj + xk = 1^T x$ s.t. $\Delta\ x \le 1$
  - Solution: x = [½, ½, ½], s = 3/2
  - Overall communication cost

    = number of times inner 3 loops executed * M = n^3/M^s * M = $n^3/M^{1/2}$

Proof of Communication Lower Bound on C = A·B (1/5)

- Proof from Irony/Toledo/Tiskin (2004)
- Think of instruction stream being executed
  - Looks like " … add, load, multiply, store, load, add, …"
    - Each load/store moves a word between fast and slow memory
  - We want to count the number of loads and stores, given that we are multiplying n-by-n matrices C = A·B using the usual $2n^3$ flops, possibly reordered assuming addition is commutative/associative
  - Assuming that at most M words can be stored in fast memory
- Outline:
  - Break instruction stream into segments, each with M loads and stores
  - Somehow bound the maximum number of flops that can be done in each segment, call it F
  - So F · # segments ≥ T = total flops = $2 \cdot n^3$, so # segments ≥ T / F
  - So # loads & stores = M · #segments ≥ M · T / F

Proof of Communication Lower Bound on C = A·B (2/5)



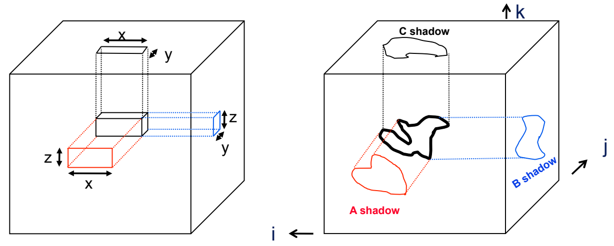
- If we have at most 2M "A squares", "B squares", and "C squares" on faces, how many cubes can we have?

Proof of Communication Lower Bound on C = A·B (3/5)

- Given segment of instruction stream with M loads & stores, how many adds & multiplies (F) can we do?
  - At most 2M entries of C, 2M entries of A and/or 2M entries of B can be accessed
- Use geometry:
  - Represent n^3 multiplications by n x n x n cube
  - One n x n face represents A
    - each 1 x 1 subsquare represents one A(i,k)
  - One n x n face represents B
    - each 1 x 1 subsquare represents one B(k,j)
  - One n x n face represents C
    - each 1 x 1 subsquare represents one C(i,j)
  - Each 1 x 1 x 1 subcube represents one C(i,j) += A(i,k) · B(k,j)
    - May be added directly to C(i,j), or to temporary accumulator

---

Proof of Communication Lower Bound on C = A·B (4/5)



cubes in black box with side lengths x, y and z
= Volume of black box
= x·y·z
= $(xz \cdot zy \cdot yx)^{1/2}$
= $(\#A\square s \cdot \#B\square s \cdot \#C\square s)^{1/2}$

(i,k) is in A shadow if (i,j,k) in 3D set
(j,k) is in B shadow if (i,j,k) in 3D set
(i,j) is in C shadow if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)
cubes in 3D set = Volume of 3D set
≤ (area(A shadow) · area(B shadow) · area(C shadow)) $^{1/2}$

Proof of Communication Lower Bound on $C = A \cdot B$ (5/5)

- Consider one “segment” of instructions with M loads, stores
- Can be at most 2M entries of A, B, C available in one segment
- Volume of set of cubes representing possible multiply/adds in one segment is $\le (2M \cdot 2M \cdot 2M)^{1/2} = (2M)^{3/2} \equiv F$
- # Segments $\ge \lfloor 2n^3 / F \rfloor$
- # Loads & Stores = $M \cdot$ #Segments $\ge M \cdot \lfloor 2n^3 / F \rfloor$

$$\ge n^3 / (2M)^{1/2} - M = \Omega(n^3 / M^{1/2})$$

- Parallel Case: apply reasoning to one processor out of P
  - # Adds and Muls $\ge 2n^3 / P$ (at least one proc does this)
  - $M = n^2 / P$ (each processor gets equal fraction of matrix)
  - # “Load & Stores” = # words moved from or to other procs $\ge M \cdot (2n^3 /P) / F = M \cdot (2n^3 /P) / (2M)^{3/2} = n^2 / (2P)^{1/2}$

Ongoing Work

- **Develop algorithm to compute lower bound in general**
- **Automate generation of approximate LPs**
- **Extend “perfect scaling” results for time and energy by using extra memory**
- **Have yet to find a case where we cannot attain lower bound – can we prove this?**
- **Incorporate into compilers**