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,…)
### Motif/Dwarf: Common Computational Methods

**Red Hot → Blue Cool**

<table>
<thead>
<tr>
<th></th>
<th>Embedded</th>
<th>SPEC DB</th>
<th>Game</th>
<th>ML</th>
<th>HPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Finite State Mach.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Combinational</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Graph Traversal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Structured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Dense Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Sparse Matrix</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Spectral (FFT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Dynamic Prog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>N-Body</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MapReduce</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Backtrack/ B&amp;B</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Graphical Models</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Unstructured Grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 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
    - 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 ...**

### 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 \( D_1/r_1 \) is small enough, stars inside smaller box can be replaced by their CM to compute the force on Vulcan
  - Boxes nest in boxes recursively

---

**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)

```plaintext
Procedure Quad_Tree_Build
Quad_Tree = {empty}
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 $\leq N \cdot$ maximum cost of Quad_Tree_Insert**
  - $= O( N \cdot$ maximum depth of Quad_Tree)

- **Uniform Distribution** of particles
  - Depth of Quad_Tree = $O( \log N )$
  - Cost $\leq O( N \cdot \log N )$

- **Arbitrary distribution of particles**
  - Depth of Quad_Tree = $O( \#$ bits in particle coords $) = O( b )$
  - Cost $\leq O( b \cdot N )$
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

- Good for low accuracy calculations:
  \[
  \text{RMS error} = \left( \frac{\sum_k |\text{approx } f(k) - \text{true } f(k)|^2}{\sum_k \text{true } f(k)^2} \right)^{1/2} \approx 1\%
  \]
  (other measures better if some true $f(k) \approx 0$)
- High Level Algorithm (in 2D, for simplicity)
  1) Build the QuadTree using QuadTreeBuild
     ... already described, cost = $O(N \log N)$ or $O(bN)$
  2) For each node = subsquare in the QuadTree, compute the CM and total mass (TM) of all the particles it contains
     ... "post order traversal" of QuadTree, cost = $O(N \log N)$ or $O(bN)$
  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

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

Step 3 of BH: compute force on each particle

For each particle, use as few nodes as possible to compute force, subject to accuracy constraint

Need criterion to decide if a node is far enough from a particle

- $D$ = side length of node
- $r$ = distance from particle to CM of node
- $\theta$ = user supplied error tolerance $< 1$
- Use CM and TM to approximate force of node on box if $D/r < \theta$
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
- \( r = \sqrt{(x_k - x_{CM})^2 + (y_k - y_{CM})^2 + (z_k - z_{CM})^2} \)
- \( G = \) gravitational constant
- Force on k \( \approx G \cdot m \cdot TM \cdot \frac{(x_{CM} - x_k, y_{CM} - y_k, z_{CM} - z_k)}{r^3} \)

Details of Step 3 of BH

- For each particle, traverse the QuadTree to compute the force on it for \( k = 1 \) to \( N \)
- \( f(k) = \text{TreeForce}(k, \text{root}) \)
  - \( \ldots \) compute force on particle k due to all particles inside root (except k)
  - \( \text{endfor} \)

Function \( f = \text{TreeForce}(k, n) \)
  - \( \ldots \) compute force on particle k due to all particles inside node n (except k)
  - \( f = 0 \)
  - if n contains one particle (not k) \( \ldots \) 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 \( D/r > \theta \) \( \ldots \) ok to approximate by CM and TM
      - compute f using formula from last slide
    - else
      - \( \ldots \) need to look inside node
        - for all children c of n
          - \( f = f + \text{TreeForce}(k, c) \)
        - 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
  - \( \text{Cost of TreeForce}(k, \text{root}) = O(\text{depth in QuadTree of leaf containing k}) \)
  - \( \text{Proof by Example (for } \theta > 1) \):
    - 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
    - \( \text{Cost} = O(\text{level of k}) \)
    - Total cost = \( O(D_{\text{max}} \text{ level of k}) = O(N \log N) \)
      - Strongly depends on \( \theta \)

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)

  - 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
  - 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

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

\[ \phi(z) = \text{potential due to } z_k, \quad k=1,\ldots,n \]
\[ = \sum_k m_k \log |z - z_k| \]
\[ = \text{Real}( \sum_k m_k \log (z - z_k) ) \]
\[ \quad \text{... since log } z = \log |z| + i\theta \]
\[ \quad \text{... drop Real()} \text{ from now on} \]
\[ = \sum_k m_k \log (z - z_k) \]
\[ \quad \text{... how logarithms work} \]
\[ = M \log(z) + \sum_k m_k \log (1 - z_k/z) \]
\[ \quad \text{... where } M = \sum_k m_k \]
\[ = M \log(z) - \sum_k m_k \sum_{k=1} z_k (z_k/z) \]
\[ \quad \text{... Taylor expansion converges if } |z_k/z| < 1 \]
\[ = M \log(z) - \sum_{k=1} z_{k-1} z_{k-2} m_k z_k z_k \]
\[ \quad \text{... swap order of summation} \]
\[ = M \log(z) - \sum_{k=1} z_{k-1} z_{k+1} \alpha_k \]
\[ \quad \text{... where } \alpha_k = \sum_k m_k z_k z_k \]
\[ \quad \text{... called Multipole Expansion} \]

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 = \(-\text{grad} \phi(x,y,z) = -(\partial \phi/\partial x, \partial \phi/\partial y, \partial \phi/\partial z) = -(x,y,z)/r^3 \)
- In 2D with x,y coordinates
  - Potential = \( \phi(x,y) = \log r = \log (x^2 + y^2)^{1/2} \)
  - Force = \(-\text{grad} \phi(x,y) = -(\partial \phi/\partial x, \partial \phi/\partial y) = -(x,y)/r^2 \)
- In 2D with z = x+iy coordinates, i = sqrt(-1)
  - Potential = \( \phi(z) = \log |z| = \text{Real}(\log z) \)
  - Force = \(-\text{grad} \phi(z) = -\partial \phi/\partial z = -z/|z|^2 \)
- Later: Kernel Independent FMM

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

\[ \phi(z) = \text{potential due to } z_k, \quad k=1,\ldots,n \]
\[ = \sum_k m_k \log |z - z_k| \]
\[ = \text{Real}( \sum_k m_k \log (z - z_k) ) \]
\[ \quad \text{... drop Real()} \text{ from now on} \]
\[ = M \log(z) - \sum_{k=1} z_{k-1} z_{k-2} \alpha_k \]
\[ \quad \text{... Taylor Expansion in 1/z} \]
\[ \quad \text{... where } M = \sum_k m_k = \text{Total Mass} \text{ and} \]
\[ \quad \alpha_k = 2k m_k z_k \]
\[ \quad \text{This is called a Multipole Expansion in } z \]
\[ = M \log(z) - \sum_{k=1} z_{k-1} z_{k-2} \alpha_k \]
\[ \quad \text{... bounded by geometric sum} \]

- Note that \( a_1 = \sum_k m_k z_k = CM/M \)
- So that \( M \) and \( a_1 \) terms have same info as Barnes-Hut

- Error (r) = O(h \{max \{ |z_k| / |z|^1\} \})
Error in Truncated 2D Multipole Expansion

\[ \text{error}(r) = O(\text{max}_k |z_k|/|z|^{r+1}) \]

Suppose \( \text{max}_k |z_k|/|z| < c < 1 \), so

\[ \text{error}(r) = O(c^{r+1}) \]

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

\( c = (D/\sqrt{2}) / (1.5*D) \approx .47 < .5 \)

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

\[ \phi(z) = M \cdot \log(z - z_n) \cdot \sum_{e \geq 1} (z - z_n)^e \alpha_e \]

Outer(n) = (M, \( \alpha_1 \), \( \alpha_2 \), ..., \( \alpha_r \) - \( z_n \))

- 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 \( \alpha_1 \) 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 \( n \)

\[ \sum_{\text{outer } \beta} (z - z_n)^e \]

\( z_n = \) center of node \( n \), a D-by-D box

Inner(n) = (\( \beta_0 \), \( \beta_1 \), ..., \( \beta_r \), \( z_n \))

Particles outside \( n \) must lie outside 3D-by-3D box centered at \( z_n \)

Top Level Description of FMM

(1) Build the QuadTree

(2) Call Build_Outer(root), to compute outer expansions of each node \( n \) in the QuadTree

... Traverse QuadTree from bottom to top, ... combining outer expansions of children ... to get 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
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?

- \( \phi_k(z) \sim M_k \log|z-z_k| + \sum_{e \geq 1} (z-z_k)^e \alpha_{e_k} \) expands around \( z_k \), \( k=1,2 \)
- First step: make them expansions around same point
- \( n_1 \) is a child (subsquare) of \( n_2 \)
- \( z_k = \text{center}(n_k) \) 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 \( \alpha_e \) and center \( z_2 \) which represents the same potential as Outer(\( n_1 \)) outside the black dashed box.

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

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

function \( M, \alpha_1, \ldots, \alpha_r, z_0 = \text{Build}_\text{Outer}(n) \) ... compute outer expansion of node \( n \)
if \( n \) if a leaf ... it contains 1 (or a few) particles
compute and return Outer(\( n \)) = \( M, \alpha_1, \ldots, \alpha_r, z_0 \) 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(\( c(k) \)) = \text{Build}_\text{Outer}(\( c(k) \))
Outer(\( n \)) = Outer(\( n \)) + Outer_shift(Outer(\( c(k) \)), \text{center}(\( n \)))
endfor
return Outer(\( n \))
end if

Cost = \( O(\# \text{nodes in QuadTree}) = O(N) \) same as for Barnes-Hut.

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(n1) = Inner_shift(Inner(n2), n1) for (r+1) x (r+1) matrix-vector multiply
° Need Inner(n4) = Convert(Outer(n3), n4) for (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 combing 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)

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 ( β1, ..., βr, z0) = Build_Inner(n)
  ... compute inner expansion of node n
  p = parent(n) ... 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 = $O(\text{# nodes in QuadTree})$
  = $O(N)$

Top Level Description of FMM

1. Build the QuadTree
2. Call Build_Outer(root), to compute outer expansions of each node n in the QuadTree
   ... Traverse QuadTree from bottom to top,
   ... combining outer expansions of children
   ... to get out outer expansion of parent
3. Call Build_Inner(root), to compute inner expansions of each node n in the QuadTree
   ... Traverse QuadTree from top to bottom,
   ... converting outer to inner expansions
   ... and combining them
4. For each leaf node n, add contributions of nearest particles directly into 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(N²) 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
  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
  • Assignment 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)

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

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. \( D(n)/r(n) < \theta \) and \( D(parent(n))/r(parent(n)) \geq \theta \)
    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 combing 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 \( \approx \) 5.8 Gflops
    - Decomposing domain 7 secs
    - Building the OctTree 7 secs
    - Tree Traversal 33 secs
    - Communication during traversal 6 secs
    - Force evaluation 54 secs
    - Load imbalance 7 secs
  - Rises to 160 secs as distribution becomes nonuniform
Performance Results - 2

- Cray T3E
  - Blackston, 1999
  - 10^-4 RMS error
  - General 80% efficient on up to 32 processors
  - Example: 50K particles, both uniform and nonuniform
    - preliminary results; lots of tuning parameters to set
- Uniform vs 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 >50 >500
  - vs O(n^2)

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

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 5-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
  - FFT used to build expansions; tunable

- Single-core, manually coded & tuned
  - Low-level: SIMD vectorization (x86)
  - Numerical: F SQRTPS + 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)

![Graph showing speedup](image)

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

---

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

![Graph showing force evaluation](image)

*Shape of curve changes as we introduce optimizations.*

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

---

**Cross-Platform Performance Comparison (Summary)**

GPU: NCSA Lincoln Cluster NVIDIA T10P + dual socket Xeon

![Graph showing performance relative to out-of-the-box Nehalem](image)

*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?

*Source: Richard Vuduc*
Traditional (Naïve $n^2$) Nbody Algorithm (using a 1D decomposition)

- Given $n$ particles, $p$ processors, $M = O(n/p)$ memory
- Each processor has $n/p$ particles
- Algorithm: shift copy of particles to the left $p$ times, calculating all pairwise forces
- Computation cost: $n^2/p$
- Communication bandwidth: $O(n)$ words
- Lower bound = $\Omega((n^2/p)/M) = \Omega(n)$, attained
- Communication latency: $O(p)$ 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)

- Divide $p$ into $c$ groups. Replicate particles in each group
- Make a copy of each group of $n*c/p$ particles
- Pass copy to the 0th…c-1th neighbor depending on row
- Main Algorithm: for $p/c^2$ steps
  - Compute pairwise 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)

- 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 $1 < c < p^{1/2}$

Communication Avoiding Version (2D decomposition is Limit)

- 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]
**Recall optimal sequential Matmul**

- **Naive 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)
  ```

- **Thm**: Picking $b = M^{1/2}$ attains lower bound:
  
  $\#\text{words}\_\text{moved} = \Omega(n^3/M^{1/2})$

- **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 $\Delta$**

  $\Delta = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}$

- **Solve LP for $x = [x_i, x_j, x_k]^T$:**
  
  $\max 1^\top x$ s.t. $\Delta x \leq 1$

  - Result: $x = [1/2, 1/2, 1/2]^T$, $1^\top x = 3/2 = S$

- **Thm**: $\#\text{words}\_\text{moved} = \Omega(n^3/M^{S-1}) = \Omega(n^3/M^{1/2})$

  Attained by block sizes $M^{x_i}, M^{x_j}, M^{x_k} = M^{1/2}, M^{1/2}, M^{1/2}$

---

**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?**
New Thm applied to Direct N-Body

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

$$\Delta = \begin{pmatrix}
  i & j \\
  1 & 0 \\
  0 & 1 \\
\end{pmatrix} F$$

* Solve LP for x = [x_i, x_j]^T: max 1^T x s.t. Δ x ≤ 1
  * Result: x = [1, 1], 1^T x = 2 = S
  * Thm: #words_moved = Ω(n^2/M^S-1) = Ω(n^2/M)
  * Attained by block sizes 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

Approach to generalizing lower bounds

* Matmul
  * for i=1:n, 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^2
    * Access locations indexed by (i,j), (i,k), (k,j)
  * General case
  * for i=1:n, for j=1:m, ... for ik = i3:i4
  * C(i+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_{D} (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 \( H < \mathbb{Z}^k \), \( \text{rank}(H) \leq \sum_j s_j \text{rank}(\varphi_j(H)) \)
- **Thm:** Given a program with array refs given by \( \varphi_j \), choose \( s_j \) to minimize \( s_{\text{HBL}} = \sum_j s_j \text{ subject to HBL-LP.} \)
  - Then \( \#\text{words}_\text{moved} = \Omega(\#\text{iterations}/M^{s_{\text{HBL}}}) \)
  - 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 \( H < \mathbb{Z}^k \), but still finitely many!
  - Thm: (bad news) Writing down all inequalities equivalent to Hilbert’s 10th problem over \( \mathbb{Q} \) (conjectured to be undecidable)
  - Thm: (good news) Can decidably write down a subset of the constraints with the same solution \( s_{\text{HBL}} \)
  - Thm: (better news) Can write it down explicitly in many cases of interest
    - Ex: when all \( \varphi_j = \{ \text{subset of indices} \} \)

**Is this bound attainable? (2/2)**

- Depends on loop dependencies
- Best case: none, or reductions (matmul)
- Thm: When all \( \varphi_j = \{ \text{subset of indices} \} \), dual of HBL-LP gives optimal tile sizes:
  - HBL-LP: minimize \( 1^Ts \) s.t. \( s^T\Delta \geq 1^T \)
  - Dual-HBL-LP: maximize \( 1^Tx \) s.t. \( \Delta^Tx \leq 1 \)
  - Then for sequential algorithm, tile \( i_j \) by \( M^{x_i} \)
- Ex: Matmul: \( s = [1/2, 1/2, 1/2]^T = x \)
- Extends to unimodular transforms of indices

**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
Proof of Communication Lower Bound on $C = A \cdot B$ (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 $C = A \cdot B$ 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 $F \cdot \# segments \geq T = \text{total flops} = 2n^3$, so $\# segments \geq T / F$
- So $\# loads & stores = M \cdot \# segments \geq M \cdot T / F$

Proof of Communication Lower Bound on $C = A \cdot B$ (2/5)

- 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 $n \times n \times n$ cube
  - One $n \times n$ face represents $A$
    - each $1 \times 1$ subsquare represents one $A(i,k)$
  - One $n \times n$ face represents $B$
    - each $1 \times 1$ subsquare represents one $B(k,j)$
  - One $n \times n$ face represents $C$
    - each $1 \times 1$ subsquare represents one $C(i,j)$
  - Each $1 \times 1 \times 1$ subcube represents one $C(i,j) = A(i,k) \cdot B(k,j)$
    - May be added directly to $C(i,j)$, or to temporary accumulator

Proof of Communication Lower Bound on $C = A \cdot B$ (3/5)

- 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 $C = A \cdot B$ (4/5)

- Thm (Loomis & Whitney, 1949)
  - $\#$ cubes in 3D set $\leq \frac{\text{area}(\text{A shadow}) \cdot \text{area}(\text{B shadow}) \cdot \text{area}(\text{C shadow})}{2}$
  - # cubes in black box with side lengths $x$, $y$ and $z$ = Volume of black box
    - $= x \cdot y \cdot z$
    - $= (x \cdot y \cdot z)^{1/2}$
    - $= (\#A_{\square} \cdot \#B_{\square} \cdot \#C_{\square})^{1/2}$

Thm (Loomis & Whitney, 1949)
- # cubes in 3D set = Volume of 3D set
  - $\leq \frac{\text{area}(\text{A shadow}) \cdot \text{area}(\text{B shadow}) \cdot \text{area}(\text{C shadow})}{2}$
**Proof of Communication Lower Bound on C = A·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 \( \leq (2M \cdot 2M \cdot 2M)^{1/2} = (2M)^{3/2} \equiv F \)
° # Segments \( \geq \lceil 2n^3 / F \rceil \)
° # Loads & Stores = M \cdot #Segments \( \geq M \cdot 2n^3 / F \)
° \( \geq n^3 / (2M)^{1/2} - M = \Omega(n^3 / M^{1/2}) \)

• Parallel Case: apply reasoning to one processor out of P
  • # Adds and Muls \( \geq 2n^3 / P \) (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
    \( \geq M \cdot (2n^3 / P) / F = M \cdot (2n^3 / P) / (2M)^{3/2} = n^2 / (2P)^{1/2} \)