CS 267: Applications of Parallel Computers

Tree-Structured Codes for N-Body Simulations

Kathy Yelick

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

Motivation

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

Outline

- Motivation
- Obvious algorithm on N bodies takes O(N^2) work
- How to reduce the number of particles in the force sum
  - We must settle for an approximate answer (say 2 decimal digits, or perhaps 16…)
- Basic Data Structures: Quad Trees and Oct Trees
- The Barnes-Hut Algorithm (BH)
  - An O(N log N) approximate algorithm for the N-Body problem
- The Fast Multipole Method (FMM)
  - An O(N) approximate algorithm for the N-Body problem
- Parallelizing BH, FMM and other algorithms
- Example applications
- Alternative approach: lots of hardware

Particle Simulation

\[
\begin{align*}
\text{t} & = 0 \\
\text{while} & \quad \text{t < t_final} \\
\text{for} & \quad i = 1 \text{ to } n \\
\text{compute} & \quad f(i) = \text{force on particle } i \\
\text{for} & \quad i = 1 \text{ to } n \\
\text{move} & \quad \text{particle i under force } f(i) \text{ for time } dt \\
\text{compute} & \quad \text{interesting properties of particles (energy, etc.)} \\
\text{t} & = t + dt \\
\text{end while}
\end{align*}
\]

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

Particle Simulation

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

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

Reducing the Number of Particles in the Sum

- Previous divide and conquer algorithms use same intuition
- Consider computing force on earth due to all celestial bodies
  - Look at night sky, \# terms in force sum \( \gg \) number of visible stars
  - One “star” is really the Andromeda galaxy, which is billions of stars
  - A lot of work if we compute this per star...
- OK to approximate all stars in Andromeda by a single point at its center of mass
  - D = size of box containing Andromeda, \( r = \text{distance of CM to Earth} \)
  - 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

**Quad Trees**

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

**Oct Trees**

- Similar data structure to subdivide 3D space
- Analogous to 2D Quad tree—each cube is divided into 8 sub-cubes

**Using Quad Trees and Oct Trees**

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

**Example of an Adaptive Quad Tree**

Adaptive quad tree where no space contains more than 1 particle

Child nodes enumerated counterclockwise from SW corner
Empty ones excluded

**Adaptive Quad Tree Algorithm**

```java
class QuadTree
build (particles)
QuadTree t = new QuadTree();
for each j in particles t.insert(j); ... loop over all N particles
insert(j); ... insert particle j in QuadTree
end build

insert(j); ... Try to insert particle j at node n in QuadTree
if this node is empty ... empty
add j as the (only) particle in this node
if this is an internal node (has 4 children) ... Internal
determine which child c contains particle j
insert(j); ... insert particle j in QuadTree
else (this quadtree contains 1 particle) ... leaf
add n’s 4 children to the QuadTree
let c be the child of the particle k already here
insert(k); ... insert particle k in QuadTree
let c be the child of n containing j
insert(j); ... insert particle j in QuadTree
end insert
```
Cost of Adaptive QuadTree Construction

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

1. Uniform distribution =>
   depth of QuadTree = \( O(\log N) \), so
   Cost = \( O(N \log N) \)

2. Arbitrary distribution =>
   depth of QuadTree = \( O(b) \),
   where \( b \) = #bits in particle coordinates, so
   Cost = \( O(bN) \)

Barnes-Hut Algorithm

• Good for low accuracy calculations:
  \[ \text{RMS error} = \left( \sum_k || \text{approx } f(k) - \text{true } f(k) ||^2 / || \text{true } f(k) ||^2 / N \right)^{1/2} \leq 1\%
  \]
  (other measures better if some true \( f(k) \) ~ 0)

1) Build the QuadTree using \( \text{QuadTree.build} \)
   => already described, cost = \( O(N \log N) \) or \( O(bN) \)

2) For each node \( n \) in the QuadTree, compute the
   CM and total mass \( \text{TM} \) 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: Compute CM and TM of Each Node

- Compute the CM = Center of Mass and TM = Total Mass
  - of all the particles in each node of the QuadTree
- Compute Mass()

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

Step 3: Compute Force on Each Particle

- For each node, can approximate force on particles outside the node
  due to particles inside node by using the node’s CM and TM
  => this will be accurate enough if the node is "far enough away" from the particle

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

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

• \( D = \) side length of node
• \( r = \) distance from particle to CM of node
• \( \theta = \) 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

• Use example of Gravity (1/r²)
• Given node \( n \) and particle \( k \), satisfying \( D/r < \theta \)
  \[ \begin{align*}
  & \text{Let } (x_k, y_k, z_k) \text{ be coordinates of } k, \text{ its mass} \\
  & \text{Let } (x_{CM}, y_{CM}, z_{CM}) \text{ be coordinates of CM} \\
  & \text{Define } \mathbf{r} = (x_k - x_{CM}, y_k - y_{CM}, z_k - z_{CM}) \\
  & \text{Then } \\
  & \text{G = gravitational constant} \\
  & \text{Force on } k = G \times m \times TM \times \frac{x_{CM} - x_k}{r^3} \frac{y_{CM} - y_k}{r^3} \frac{z_{CM} - z_k}{r^3}
  \end{align*} \]
Analysis of Step 3 of BH

- Correctness follows from recursive accumulation of force from each subtree
  - Each particle is accounted for exactly once, whether it is in a leaf or other node
- Complexity analysis
  - Cost of t.treeForce(k) = O(depth in t of leaf containing k)
  - Proof by Example (for $\theta > 1$):
    - For each undivided node, (except one containing k), $D/r < 1 < \theta$
    - There are 3 nodes at each level of the QuadTree
    - There is $O(1)$ work per node
    - Cost = $O($level of k$) = O(N \log N)$
  - Total cost = $O(\sum k$ level of k$) = O(N \log N)$
- Strongly depends on $\theta$

Sample BH calculation, assuming $\theta > 1$

Fast Multiple Method (FMM)

- FMM uses two kinds of expansions
  - Outer expansions represent potential outside node due to particles inside, analogous to (CM,TM)
  - Inner expansions represent potential inside node due to particles outside
    - Computing this for every leaf node is the computational goal of FMM
- First review potential, then return to FMM

Gravitational/Electrostatic Potential

- Force on particle at (x,y,z) due to one at origin = -(x,y,z)/r3
  - Instead of force, consider potential $\phi(x,y,z) = -1/r$
    - Potential satisfies 3D Poisson equation
      \[ \frac{d^2 \phi}{dx^2} + \frac{d^2 \phi}{dy^2} + \frac{d^2 \phi}{dz^2} = 0 \]
    - Force = -grad $\phi(x,y,z)$ = $-(d\phi/dx, d\phi/dy, d\phi/dz)$
- FMM will compute a compact expression for $\phi(x,y,z)$, which can be evaluated and/or differentiated at any point
- For simplicity, present algorithm in 2D instead of 3D
  - Force = $(x,y)/r^2 = -z/|z|^2$ where $z = x + iy$ (complex number)
  - Potential = log $|z|$ 
    - Potential satisfies 2D Poisson equation
      \[ \frac{d^2 \phi}{dx^2} + \frac{d^2 \phi}{dy^2} = 0 \]
    - Equivalent to gravity between “infinite parallel wires” instead of point masses

2D Multipole (Taylor Expansion in 1/z)

- $\phi(z) = \text{potential due to } zk, k=1,...,n$
  - $\sum k m_k \log |z - zk| \ldots \text{sum potential over all particles}$
  - $\text{Real}(\sum k m_k \log (z - zk)) \ldots$ since $\log z = \log |z| + i\theta$
  - $\text{drop Real})$ from now on
  - $\sum k m_k \cdot [\log(z) + \log (1 - zk/z)]$
    - how logarithms work
  - $M \cdot \log(z) + \sum k m_k \cdot \log (1 - zk/z)$
    - where $M = \sum k m_k$
  - $M \cdot \log(z) + \sum k m_k \cdot \sum e=1 (zk/z)^e$
    - Taylor expansion converges if $|zk/z| < 1$
  - $M \cdot \log(z) + \sum e=0 \ldots 1 z^e \log (zk/z^e)$
    - swap order of summation
  - $M \cdot \log(z) + \sum e=0 \ldots 1 z^e \log \theta$
    - where $\theta = \sum k m_k z_k^e$
  - $M \cdot \log(z) + \sum e=0 \ldots 1 |z|^e \log \theta$
    - where $\theta = \sum z_k \log \theta$

Summary of FMM

(1) Build the QuadTree
(2) Call Build_Outer(root), to compute outer expansions of each node n in the QuadTree
(3) Traverse the QuadTree from top to bottom, computing Inner(n) for each node n in QuadTree
  - still need to show how to convert outer to inner expansions
(4) For each leaf node n, add contributions of nearest particles directly into Inner(n)
  - since Inner(n) only includes potential from distant nodes
Parallelizing Hierarchical N-Body Codes

- Barnes-Hut, FMM and related algorithm have similar computational structure:
  1) Build the QuadTree
  2) Traverse QuadTree from leaves to root and build outer expansions (just (TM,CM) for Barnes-Hut)
  3) Traverse QuadTree from root to leaves and build any inner expansions
  4) Traverse QuadTree to accumulate forces for each particle

- One parallelization scheme will work for them all
  - Assign regions of space to each processor
  - Regions may have different shapes, to get load balance
  - Each region will have about N/p particles
  - Each processor will store part of QuadTree containing all particles 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
  - Given the LET, all force accumulations (step 4) are done in parallel, without communication
  - (Description based on SC97 paper by D. Blackston and T. Suel)

Programming Model in PBody

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

Load Balancing 1: ORB

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

Load Balancing 2: Costzones

- Called Costzones for Shared Memory
  - PhD thesis, J.P. Singh, Stanford, 1993
- Called "Hashed Oct Tree" for Distributed Memory
  - Warren and Salmon, Supercomputing 93
- We will use the name Costzones for both; also in Pbody
- Idea: partition QuadTree instead of space
  - Estimate work for each node, call total work W
  - Arrange nodes of QuadTree in some linear order (lots of choices)
  - Assign contiguous blocks of nodes with work W/p to processors
  - Works well in 3D

Linearly Ordering Nodes for Costzones

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

- Assign unique key to each node in QuadTree, then compute hash(key) to get a linear order
- key = interleaved bits of x,y coordinates of node, prefixed by 1
- 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, then determine how to best build QuadTree
- Random Sampling:
  - All processors send small random sample of their particles to Proc 1
  - Proc 1 builds small Quadtree serially, determines its Costzones, and broadcasts them to all processors
  - Other processors build part of Quadtree they are assigned by these Costzones
  - All processors know all Costzones; we need this later to compute LETs

Computing Locally Essential Trees (LETs)

- Warren and Salmon, 1992; Liu and Bhatt, 1994
- Every processor needs a subset of the whole QuadTree, called the LET, to compute the force on all particles it owns
- Shared Memory
  - Receiver Driven Protocol
  - Each processor reads part of QuadTree it needs from shared memory on demand, keeps it in cache
  - Drawback: cache memory appears to need to grow proportionally to P to remain scalable
- Distributed Memory
  - Sender driven protocol
  - Each processor decides which other processors need parts of its local subset of the Quadtree, and sends these subsets

Locally Essential Trees in Distributed Memory

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

Applications of Fast N-Body Algorithms

- Astrophysics and Celestial Mechanics
  - Intel Delta = 1992 supercomputer, 512 Intel i860s
  - 17 million particles, 600 time steps, 24 hours elapsed time
    - M. Warren and J. Salmon
  - Gordon Bell Prize at Supercomputing 92
  - Sustained 5.2 Gflops = 44K Flops/particle/time step
  - 1% accuracy
  - Direct method (17 Flops/particle/time step) at 5.2 Gflops would have taken 18 years, 6570 times longer
- Plasma Simulation
- Molecular Dynamics
- Electron-Beam Lithography Device Simulation
- Fluid Dynamics (vortex method)

Performance Results - 1

- 512 Proc Intel Delta
  - Warren and Salmon, Supercomputing 92
  - 8.8 M particles, uniformly distributed
  - 0.1% to 1.0% RMS error
  - 114 seconds = 5.8 Gflops
    - Decomposing domain 7 secs
    - Building the OctTree 7 secs
    - Tree Traversal 33 secs
    - Communication during traversal 6 secs
    - Force evaluation 54 secs
    - Load imbalance 7 secs
  - Rises to 160 secs as distribution becomes nonuniform
### Performance Results - 2

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

<table>
<thead>
<tr>
<th>Uniform</th>
<th>1 proc</th>
<th>4 procs</th>
<th>1 proc</th>
<th>4 procs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tree size</td>
<td>2745</td>
<td>2745</td>
<td>5729</td>
<td>5729</td>
</tr>
<tr>
<td>MaxDepth</td>
<td>4</td>
<td>4</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>Time(secs)</td>
<td>172.4</td>
<td>38.9</td>
<td>14.7</td>
<td>2.4</td>
</tr>
<tr>
<td>Speedup</td>
<td>4.4</td>
<td>&gt;50</td>
<td>6.1</td>
<td>&gt;500</td>
</tr>
</tbody>
</table>

### Alternate Approach: Hardware

**Grape-6 System**

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