COMP 633 - Parallel Computing

Lecture 8
September 14, 2017

SMM (3)

OpenMP Case Study:
The Barnes-Hut N-body Algorithm
Topics

• Case study: the Barnes-Hut algorithm
  – Study an important algorithm in scientific computing
    » n-body simulation with long range forces
  – Investigate parallelization and implementation in a shared memory multiprocessor
    » expression and management of parallelism
    » memory hierarchy tuning
N-body simulations: self-gravitating systems

N-Body Simulation of the Cold Dark Matter Cosmology
The *n*-body simulation problem

- **Simulate the evolution of a system of *n* bodies over time**
  - Pairwise interaction of bodies
    - force $f(i,j)$ on body $i$ due to body $j$
    - total force $f(i)$ on body $i$ due to all bodies
    - acceleration of body $i$ via $f = ma$
  - Numerical integration of body velocities and positions
    - timestep $\Delta t$

- **Non-negligible long-range forces**
  - for uniformly distributed bodies in 3D, total force due to all bodies at a given distance $r$ is constant
    - cannot ignore contribution of distant bodies

- **Examples**
  - astrophysics (gravity)
  - molecular dynamics (electrostatics)

---

The basic simulation algorithm:

```plaintext
while (t < t_{Final}) do
  forall 1 \leq i \leq n do
    \langle compute force $f(i)$ on body $i$ \rangle
  end
  \langle update velocity and position of all bodies \rangle
  t = t + \Delta t
end
```

Direct approach:

$O(n^2)$ interactions per time-step

---

Ex: Gravitation

$$r_{ij} = ||\mathbf{p}_i - \mathbf{p}_j||$$

$$f(i, j) = -G \cdot \frac{m_i \cdot m_j}{r_{ij}^2} \cdot \frac{\mathbf{p}_i - \mathbf{p}_j}{r_{ij}}$$

$$f(i) = \sum_{j \neq i} f(i, j)$$
Reducing the number of interactions

Exploit combined effect of “distant” bodies

Formally

- **Monopole approximation** of the force on the earth due to interaction with all masses in the *Andromeda* galaxy

\[ f(b_{\text{earth}}) \approx -G \frac{m_{\text{earth}} M (p_{\text{earth}} - c)}{r^3} \]

- Monopole approximation saves work if it can be reused with multiple bodies

- Accuracy of approximation improves with
  - increasing \( r \)
  - decreasing \( d \)
  - order of the approximation
    - Monopole, dipole, quadropole, …
  - uniformity of body distribution

apply this idea *recursively*:
- determines control-structure
- requires hierarchical decomposition of space
Hierarchical decomposition of space

- A quadtree
- An octree decomposition
- An adaptive quadtree
The Barnes-Hut algorithm

**stepSystem():**

// P(i) is coordinates and mass of body i
T := makeTree(P(1:n))
forall 1 ≤ i ≤ n do
    f(i) = gravCalc(P(i), T)
⟨ update velocities and positions ⟩

**function gravCalc(body p, treenode q):**

if ("q is a leaf") then
    ⟨ return body-body interaction (p, q) ⟩
else
    if ("p is distant enough from q") then
        ⟨ return body-cell interaction (p, q) ⟩
    else
        forall q' ∈ nonemptyChildren(q) do
            accumulate gravCalc(p, q')
        ⟨ return accumulated interaction ⟩
    end if
end if

interaction in the case of gravitation:

\[
F = G \frac{m_p \cdot m_q}{r_{pq}^2} \cdot \begin{bmatrix} x_p - x_q \over r_{pq} \\ y_p - y_q \over r_{pq} \\ z_p - z_q \over r_{pq} \end{bmatrix}
\]

\[ r_{pq} = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2 + (z_p - z_q)^2} \]

**body-body interaction:** use masses of bodies and distance between them.

**body-cell interaction:** use mass of body and mass of cell and distance between body and center of mass of cell.

force is additive; individual contributions can be accumulated.
The Barnes-Hut algorithm - Performance issues

stepSystem(P(1:n))
    -- P(1:n) is sequence of bodies
    T := makeTree(P(1:n))
    forall 1 ≤ i ≤ n do
        f(i) := gravCalc(P(i), T)
        ⟨update velocities and positions⟩

function gravCalc(p, q)
    if (“q is a leaf”) then
        ⟨return body-body interaction⟩
    else
        if (“p is distant enough from q”) then
            ⟨return body-cell interaction⟩
        else
            forall q’ ∈ nonemptyChildren(q) do
                accumulate gravCalc(p, q’)
                ⟨return accumulated interaction⟩
        end if
    end if
end if
Constructing the tree

- **Small fraction \( f \) of the total work**
  - but sequential tree construction can limit overall speedup
    - Amdahl's law: \( SP < \frac{1}{f} \)

- **Computing monopole approximation for each cell**
  - Post-order traversal of tree
    - At leaves, monopole coincides with single body
    - At interior nodes, monopole is weighted sum of all children’s monopoles

```
function makeTree( P(1:n) )
    for i := 1 to n do
        T := insert(P(i),T)
        \( \langle \text{compute monopole approximation at each node} \rangle \)
    endfor

function insert(p,T)
    if empty(T) then
        \( \langle \text{return p as singleton tree} \rangle \)
    else
        \( \langle \text{determine child S of T in which p belongs} \rangle \)
        S' := insert(p,S)
        \( \langle \text{return T with S replaced by S'} \rangle \)
    endif
```

![Tree diagram](image)
The acceptance criterion

- when is a cell “distant enough”?

original criterion used by Barnes-Hut:

\[
\frac{d}{r} < \theta \equiv r > \frac{d}{\theta}
\]

where usually

\[0.7 \leq \theta \leq 1.0\]

- problem: detonating galaxy anomaly

(one) solution: \textit{add distance between center of mass (cm) and geometric center of cell (c)}

\[
r > \frac{d}{\theta} + |cm - c|
\]
Effects of acceptance criterion … on runtime

Fig. 3.—Scaling of CRAY X-MP CPU time (CPU seconds per step per particle) for spherical, isotropic Plummer models, as a function of the number of particles, for values of the clumping parameter $\theta$ in the range $0 \leq \theta \leq 1.5$. Only monopole terms have been included in the force computation.

Effects of acceptance criterion ... on accuracy

\[ \log_{10} \text{relative error (\%)} \]

\[ N = 1024 \]
\[ N = 4096 \]
\[ N = 16,384 \]
\[ N = 32,768 \]

Fig. 6.—Magnitude of the typical error (in percent) in the tree force computation, relative to a direct sum, as a function of $\theta$, for selected values of the particle number $N$. The calculations have assumed spherical, isotropic Plummer models with softening parameter $\varepsilon = 0$, and only monopole terms have been included in the force computations.


1% accuracy sufficient for most astrophysical simulations. Different techniques with better error control necessary for other systems (*fast multipole methods*).
Effect of body distribution … on total work

For fixed $n$

- uniform distributions generate high interaction work (shallow trees)
- non-uniform distributions generate higher tree construction and lower interaction work
Complexity of Barnes-Hut

• **Tree building**
  – cost of tree construction depends on particle distribution
    » cost of body insertion \( \propto \) distance to root
    » for a uniform distribution of \( n \) particles, sequential construction of the tree is \( O(n \log n) \) time
  – In a simulation, tree could be maintained rather than reconstructed each time step

• **Force calculation (uniform distribution of bodies in 2D)**
  – consider computing the force acting on a body in the lower right corner
  – if \( \theta = 1.0 \) the 3 undivided top-level squares will satisfy the acceptance criterion
  – The remaining square does not satisfy the criterion, hence we descend into the next level
  – each level of the tree incurs a constant amount of work while descending along the path to the lower right corner
  – for a uniform distribution of \( n \) bodies, the length of the path is \( O(\log_4 n) \)
  – computing the forces on \( n \) bodies is \( O(n \log n) \) work
  – non-uniform distribution more difficult to analyze

• **Accuracy and complexity are difficult to control**
Implementation issues - parallelization

• parallelization of the force computation loop:

```fortran
SUBROUTINE stepSystem()
  CALL makeTree()
  !$OMP PARALLEL DO SCHEDULE(GUIDED,4)
  DO i = 1, n
    CALL gravCalc(i,root)
  END DO
  !$OMP END PARALLEL DO
  !$OMP PARALLEL DO
  !$OMP END PARALLEL DO
  ⟨integrate velocities and positions⟩
  !$OMP END PARALLEL DO
END SUBROUTINE stepSystem
```

• observations:
  – force computation scales reasonably up to 16 processors
  – dynamic scheduling important
  – single processor performance not impressive

<table>
<thead>
<tr>
<th>Processors</th>
<th>Tree Construction (sec)</th>
<th>Force Computation (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25.759</td>
<td>1568.854</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>27.444</td>
<td>809.294</td>
<td>1.94</td>
</tr>
<tr>
<td>4</td>
<td>29.028</td>
<td>416.174</td>
<td>3.77</td>
</tr>
<tr>
<td>8</td>
<td>24.334</td>
<td>196.997</td>
<td>7.96</td>
</tr>
<tr>
<td>16</td>
<td>26.066</td>
<td>120.664</td>
<td>13.00</td>
</tr>
</tbody>
</table>

Results on O2000 (evans) for 1M particles

0
200
400
600
800
1000
1200
1400
1600
1800
sec

Processors
Implementation issues - tuning of gravCalc (1)

- **performance analysis of gravCalc shows**
  - poor cache reuse (90% L1 and 88% L2)
  - poor use of floating point units
  - poor reuse of subexpressions
  compiler can’t generate good code?

- **manual tuning of gravCalc**
  - inline computation of acceptance criterion
  - inline computation of interaction
  - reuse distance vector (body-cell)
  - fuse loops
  **significant performance improvement!**

- **observations:**
  - 2.5 times faster
  - good scaling
  - better use of FPUs and better prediction
  cache reuse (93% L1 and 94% L2) still bad

**RECURSIVE SUBROUTINE** gravCalc(p,q)

```plaintext
IF ("q is a body") THEN
  (compute body-body interaction; accumulate)
ELSE
  IF ("p is distant enough from q") THEN
    (compute body-cell interaction; accumulate)
  ELSE
    DO q’ ∈ nonemptyChildren(q)
       CALL gravCalc(p,q’)
    END DO
  END IF
END IF
END SUBROUTINE gravCalc
```

**Results on O2000 (evans) for 1M particles**

<table>
<thead>
<tr>
<th>Processors</th>
<th>tree construction</th>
<th>force computation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.066</td>
<td>639.961</td>
</tr>
<tr>
<td>2</td>
<td>17.878</td>
<td>315.785</td>
</tr>
<tr>
<td>4</td>
<td>19.527</td>
<td>164.764</td>
</tr>
<tr>
<td>8</td>
<td>15.323</td>
<td>79.049</td>
</tr>
<tr>
<td>16</td>
<td>13.686</td>
<td>44.678</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processors</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>2.03</td>
</tr>
<tr>
<td>4</td>
<td>3.88</td>
</tr>
<tr>
<td>8</td>
<td>8.10</td>
</tr>
<tr>
<td>16</td>
<td>14.32</td>
</tr>
</tbody>
</table>
Implementation issues - tuning of gravCalc (2a)

• how can we improve cache reuse?
  – neighboring bodies in space will most likely interact with the same cells and bodies!

• sort bodies according to some spatial order:
  – precompute spatial order such as Morton order or Peano-Hilbert order
  – or simply order bodies as they are encountered during a depth-first treewalk of T
  – Sorted bodies may also speed up subsequent tree rebuilding

Morton order  Peano-Hilbert order  Tree order
Implementation issues - tuning of gravCalc (2b)

• observations:
  – 30-40% increase in performance
  – very good scaling
  – L2 reuse now up at 99.8%
  – L1 still at 93%

\[
\text{stepSystem}(P(1:n))
\]

\[
T := \text{makeTree}(P(1:n))
\]

\[
\text{re-order } P(1:n) \text{ according to } T
\]

\[
\text{forall } 1 \leq i \leq n \text{ do}
\]

\[
f(i) := \text{gravCalc}(P(i), T)
\]

\[
\langle \text{update velocities and positions} \rangle
\]

Results on O2000 (evans) for 1M particles

\[
\begin{array}{c|c|c|c|c|c}
1 & 2 & 4 & 8 & 16 \\
\hline
\text{force computation} & 495.355 & 247.89 & 125.225 & 62.741 & 31.281 \\
\text{speedup} & 1.00 & 2.00 & 3.96 & 7.90 & 15.84 \\
\end{array}
\]
Implementation issues - tuning of gravCalc (3)

How can we improve L1 reuse?

– interact a group of bodies with a cell or body!
– walk the tree and compute forces for a set of neighboring bodies

```
RECURSIVE SUBROUTINE gravCalc(set P,node q)
  IF (“q is a body”) THEN
    DO p ∈ P
      ⟨compute body-body interaction; accumulate⟩
    END DO
  ELSE
    P’ = ∅
    DO p ∈ P
      IF (“p is distant enough from q”) THEN
        ⟨compute body-cell interaction; accumulate⟩
      ELSE
        P’ = P’ ⊔ {p}
      END IF
    END DO
  IF (P’.NE. ∅) THEN
    DO q’ ∈ nonemptyChildren(q)
      CALL gravCalc(P’,q’)
    END DO
  END IF
END SUBROUTINE gravCalc
```

Results on O2000 (evans) for 1M particles

```
<table>
<thead>
<tr>
<th>Processors</th>
<th>Tree Construction (sec)</th>
<th>Force Computation (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.041</td>
<td>421.391</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>19.471</td>
<td>205.309</td>
<td>2.05</td>
</tr>
<tr>
<td>4</td>
<td>19.824</td>
<td>104.438</td>
<td>4.03</td>
</tr>
<tr>
<td>8</td>
<td>18.605</td>
<td>51.828</td>
<td>8.13</td>
</tr>
<tr>
<td>16</td>
<td>13.716</td>
<td>25.805</td>
<td>16.33</td>
</tr>
</tbody>
</table>
```

observations:

- 20-40% increase in performance
- L1 reuse now at 99.7%
  (32 bodies per group)
- L2 down slightly at 96%
- ordered particles essential
Another technique to improve L1 reuse

– allow leaf-cells to contain more than 1 body
– compute the body-body interactions in a doubly nested loop.

```
RECURSIVE SUBROUTINE gravCalc(set P, node q)
    P' = ∅
    DO p ∈ P
        IF (“p is distant enough from q”) THEN
            (compute body-cell interaction; accumulate)
        ELSE
            IF (“q is a leaf”) THEN
                DO p ∈ P, q’ ∈ q
                    (compute body-body interaction; accumulate)
            END DO
        ELSE
            P’ = P’ ∪ {p}
        END IF
    END IF
    END DO
    IF (P’.NE.∅) THEN
        DO q’ ∈ nonemptyChildren(q)
            CALL gravCalc(P’,q’)
        END DO
    END IF
END SUBROUTINE gravCalc
```

Results on O2000 (evans) for 1M particles

- force computation: 378.345, 189.231, 94.996, 47.866, 23.809
- speedup: 1.00, 2.00, 3.98, 7.90, 15.89

observations:

- 10% increase in performance

this algorithm will perform strictly more work than the previous versions! More particles per leaf potentially causes more body-body interactions and fewer body-cell interactions to be computed.
Implementation issues - summary

• Shared memory model
  – enables relatively simple parallelization of basic algorithm using OpenMP
  – shared memory model critical in dynamic load balancing

• Performance tuning
  – overall these optimizations lead to 4-5 times faster single-processor performance
  – Linear or superlinear parallel speedup to 16 processors
  – optimizing serial performance is essential for obtaining good parallel performance
  – last two optimization are instances of exposing parallelism to improve serial performance

• Observations
  – the better the performance of gravCalc the more seriously the serial tree-construction affects the overall speedup
    » when makeTree time is included in speedup
      • speedup drops from 13.00 to 10.8 for p = 16 in first version
      • speedup drops from 15.89 to 11.74 for p = 16 on last version
  – parallel tree construction algorithms!