1. **Shared Memory Implementation**

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

• Shared-memory
  – how can this be implemented in a scalable fashion?
  – n-body example

• Shared-memory multiprocessor performance and implementation issues
  – coherence
  – consistency
  – synchronization

• Example
  – Implementation of Barnes-Hut N-body algorithm
Shared memory multi-processors

- Main memory has a fixed access time
  - it has to serialize reads and writes
  - naïve implementation requires processors to serialize memory references among all processors
    - this doesn’t scale

- Instead processors maintain local caches of memory data
  - locality of reference
    - the unit of transfer to/from memory is a cache line (64 bytes)
    - L1 and L2 caches are local to the core
    - L3 is local to the socket
    - first touch principle for page faults
      - the page frame is allocated in the physical memory attached to the socket
Shared-memory multiprocessor implementation

• Objectives
  – Examine implementation issues in shared-memory multiprocessors
    • cache coherence
    • memory consistency
    • synchronization mechanisms

• Why?
  – Correctness
    • memory consistency (or lack thereof) can be the source of very subtle bugs
  – Performance
    • cache coherence and synchronization mechanisms can have profound performance implications
Coherence of memory location x

- Defined by three properties (assume x = 0 initially)

\[ \text{time} \]

(a) \[ P_1: \ W(x,1) \quad 1 = R(x) \]
   
   no intervening write of x
   
   by \( P_1 \) or other processor

(b) \[ P_1: \ W(x,1) \]
   \[ P_2: \quad 1 = R(x) \]
   
   sufficiently large
   
   interval and no
   
   other write of x

(c) \[ P_1: \ W(x,1) \quad a = R(x) \]
   \[ P_2: \ W(x,2) \quad a = R(x) \quad a \in \{1,2\} \]
   \[ P_3: \quad a = R(x) \quad \text{and has same value at all processors} \]
   
   sufficiently large
   
   interval and no other writes of x
Consistency Models

• The consistency problem
  – Performance motivates replication
    • Keep data in caches close to processors
  – Replication of read-only blocks is easy
    • No consistency problem
  – Replication of written blocks is hard
    • In what order do we see different write operations?
    • Can we see different orders when viewed from different processors?
  – Fundamental trade-offs
    • Programmer-friendly models perform poorly
Cache-coherent shared memory multiprocessor

- Implementations
  - shared bus
    - bus may be a “slotted” ring
  - scalable interconnect
    - fixed per-processor bandwidth

- Effect of CPU write on local cache
  - write-through policy – value is written to cache and to memory
  - write-back policy – value written in cache only; memory updated upon cache line eviction

- Effect of CPU write on remote cache
  - update – remote value is modified
  - invalidate – remote value is marked invalid
Bus-Based Shared-Memory protocols

- Invalidation protocol with write-back cache
  - Cache block can be in one of three states:
    - INVALID — The block does not contain valid data
    - SHARED — The block is a current copy of memory data
      - other copies may exist in other caches
    - EXCLUSIVE — The block holds the only copy of the correct data
      - memory may be incorrect, no other cache holds this block

- Handling exclusively-held blocks
  - Processor events
    - cache is block “owner”
      » reads and writes are local
  - Snooping events
    - on detecting a read-miss or write-miss from another processor to an exclusive block
      » write-back block to memory
      » change state to shared (on external read-miss) or invalid (on external write-miss)
Invalidation protocol: example

Shared

Excl
Invalid

Shared

Invalid
Excl
Invalid
Implementation: FSM per cache line

• Action in response to CPU event

Invalid -> Shared:
- CPU read
- Place read-miss on bus
- Eviction

Shared -> Invalid:
- CPU read
- Write-back block

Invalid -> Excl:
- CPU write
- Place write-miss on bus

Excl -> Invalid:
- CPU read
- CPU write

• Action in response to bus event

Invalid -> Shared:
- Write-miss for this block

Shared -> Invalid:
- Read-miss for this block
- Write-back block
**Intel cache coherence (skylake)**

- basically a directory-based protocol with 2 or 4 clusters
- each package (socket) is a cluster with p cores distributed across two slotted rings
Intel physical organization

- up to 4 sockets
- up to 28 cores per socket
- up to 56 thread contexts (28 threads and 28 hyperthreads)
Mapping OpenMP threads to hardware (1)

- Mapping threads to maximize data locality
  - `KMP_AFFINITY = "granularity=fine,compact"

Nearby thread-ids tend to share more lower-level cache

Note: we use a fictional machine with 2 sockets and 4 cores with hyperthreads to illustrate these mappings.
Mapping OpenMP threads to hardware (2)

- Mapping threads to maximize bandwidth without data locality
  - KMP_AFFINITY = “granularity=fine,scatter”

```
0 4 2 6 1 5 3 7 OpenMP thread id
```
Mapping OpenMP threads to hardware (3)

- Mapping threads to maximize data locality and equal thread progress
  - `KMP_AFFINITY = “granularity=fine,compact,1,0”`
  - `OMP_NUM_THREADS = 4`

![Diagram showing mapping of OpenMP threads to hardware]
Mapping OpenMP threads to hardware (4)

- Mapping threads to maximize bandwidth and equal thread progress
  - KMP_AFFINITY = “granularity=fine,scatter”
  - OMP_NUM_THREADS = 4
Coherence and Consistency

- **Coherence**
  - behavior of a single memory location
  - viewed from a single processor
  - read returns “most recent” written value

- **Consistency**
  - behavior of multiple memory locations read and written by multiple processors
  - viewed from one or more of the processors
  - read may not return the “most recent” value
    - What are the permitted ordering among reads and writes of several memory locations?
Example

• Case study: the Barnes-Hut algorithm
  – Study an important algorithm in scientific computing
    • efficient 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
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)

\[
\text{Ex: Gravitation } \mathbf{r}_{ij} = \| \mathbf{p}_i - \mathbf{p}_j \| \\
\quad f(i, j) = -G \cdot \frac{m_i \cdot m_j}{r_{ij}^2} \cdot \frac{\mathbf{p}_i - \mathbf{p}_j}{r_{ij}}
\]

the basic simulation algorithm:

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

Direct approach:
\( O(n^2) \) interactions per time-step
Reducing the number of interactions

Exploit combined effect of “distant” bodies

- 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

2D

a quadtree

an octree decomposition

3D

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
end function

Gravitation in 3D:

\[ F = \frac{G \cdot m_p \cdot m_q}{r_{pq}^2} \]
\[ \left[ \frac{x_p - x_q}{r_{pq}}, \frac{y_p - y_q}{r_{pq}}, \frac{z_p - z_q}{r_{pq}} \right] \]

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

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 \leq i \leq 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

Parallelism

* nested parallelism
  * over bodies
  * over recursively divided cells
load balance
different number of interactions for different bodies

Locality

nearby bodies interact with similar set of nodes in tree
Constructing the tree

- Small fraction $f$ of the total work
  - but sequential tree construction can limit overall speedup
    - Amdahl’s law: $SP < 1/f$

- 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

```plaintext
def function insert(p,T)
  if empty(T) then
    return p as singleton tree
  else
    determine child S of T in which p belongs
    $S' := insert(p,S)$
    return T with S replaced by $S'$
  endif

def function makeTree( P(1:n) )
  for i := 1 to n do
    $T := insert(P(i),T)$
    compute monopole approximation at each node
  endfor
```

```plaintext
def function makeTree( P(1:n) )
  for i := 1 to n do
    $T := insert(P(i),T)$
    compute monopole approximation at each node
  endfor
```

```plaintext
def function makeTree( P(1:n) )
  for i := 1 to n do
    $T := insert(P(i),T)$
    compute monopole approximation at each node
  endfor
```

```
```
The acceptance criterion

- when is a cell “distant enough”?

<table>
<thead>
<tr>
<th>Earth</th>
<th>r</th>
<th>d</th>
<th>Center of mass</th>
<th>Andromeda</th>
</tr>
</thead>
</table>

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: 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 monotone terms have been included in the force computation.

Effects of acceptance criterion ... on accuracy

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 $\rho = 0$, 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 distribution of bodies
    • 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:

SUBROUTINE stepSystem()
    CALL makeTree()
    !$OMP PARALLEL DO SCHEDULE(GUIDED,4)
    DO i = 1, n
        CALL gravCalc(i,root)
    END DO
    !$OMP END PARALLEL DO
    !$OMP PARALLEL DO
    ⟨integrate velocities and positions⟩
    !$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</th>
<th>force computation</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>
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
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
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%

```plaintext
stepSystem(P(1:n))

T := makeTree(P(1:n))
re-order P(1:n) according to T
forall 1 \leq i \leq n do
  f(i) := gravCalc(P(i),T)
(update velocities and positions)
```

Results on O2000 (evans) for 1M particles

<table>
<thead>
<tr>
<th>Processors</th>
<th>tree construction</th>
<th>force computation</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19.161</td>
<td>495.355</td>
<td>1.00</td>
</tr>
<tr>
<td>2</td>
<td>14.51</td>
<td>247.89</td>
<td>2.00</td>
</tr>
<tr>
<td>4</td>
<td>18.524</td>
<td>125.225</td>
<td>3.96</td>
</tr>
<tr>
<td>8</td>
<td>18.564</td>
<td>62.741</td>
<td>7.90</td>
</tr>
<tr>
<td>16</td>
<td>19.873</td>
<td>31.281</td>
<td>15.84</td>
</tr>
</tbody>
</table>

- graph of tree structure
- table showing performance metrics
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

```fortran
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 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>
<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
```

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!