# Divide and Conquer

- Divide the problem into several subproblems of equal size. Recursively solve each subproblem in parallel. Merge the solutions to the various subproblems into a solution for the original problem.

# Divide and Conquer

- Divide the problem into several subproblems of equal size. Recursively solve each subproblem in parallel. Merge the solutions to the various subproblems into a solution for the original problem.
- Dividing the problem is usually straightforward. The effort here often lies in combining the results effectively in parallel.

# Divide and Conquer Examples

- Top-down recursive mergesort.
- Gravitational N-body problem.

# Top-down Mergesort

```
MergeSort(A, low, high)

if low < high
    then  mid ← ⌊(low + high)/2⌋
          MergeSort (A, low, mid)
          MergeSort (A, mid + 1, high)
          Merge (A, low, mid, high)
```

# Top-down Mergesort

MergeSort(A, low, high)

1. **if** low $<$ high
2. **then** mid $\leftarrow \lfloor(\text{low} + \text{high})/2\rfloor$
3. MergeSort (A, low, mid)
4. MergeSort (A, mid $+$ 1, high)
4. Merge (A, low, mid, high)

- Top-down parallelization would be to create two processes that each handle one of the two recursive sort calls. The original process waits for them to finish and then merges the results.

# Top-down Mergesort

MergeSort(A, low, high)

1. **if** low $<$ high
2. **then** mid $\leftarrow \lfloor (\text{low} + \text{high})/2 \rfloor$
3. MergeSort (A, low, mid)
4. MergeSort (A, mid $+ 1$, high)
4. Merge (A, low, mid, high)

- Top-down parallelization would be to create two processes that each handle one of the two recursive sort calls. The original process waits for them to finish and then merges the results.
- Only feasible on a shared memory system. Would still have to limit the number of processes.

# N-Body Problem

- The N-body problem is concerned with determining the effects of forces between “bodies.” (astronomical, molecular dynamics, fluid dynamics etc)

# N-Body Problem

- The N-body problem is concerned with determining the effects of forces between "bodies." (astronomical, molecular dynamics, fluid dynamics etc)
- Gravitational N-body Problem. To simulate the positions and movements of the bodies in space that are subject to gravitational forces from other bodies using the Newtonian laws of physics.

# Gravitational N-body Problem

One of the deepest optical views showing early galaxies starting to form. The image is from the Hubble Telescope operated by NASA.

# Gravitational N-body Problem

A swarm of ancient stars.

# Gravitational N-body problem

- Given two bodies with masses $m_a$ and $m_b$, the gravitational force is given by
$$F = G\frac{m_a m_b}{r^2},$$
where G is the gravitational constant (which is $6.67259(\pm 0.00030) \times 10^{-11} kg^{-1} m^3 s^{-2}$) and $r$ is the distance between the bodies.

# Gravitational N-body problem

- Given two bodies with masses $m_a$ and $m_b$, the gravitational force is given by
$$F = G\frac{m_a m_b}{r^2},$$
where G is the gravitational constant (which is $6.67259(\pm 0.00030) \times 10^{-11} kg^{-1} m^3 s^{-2}$) and $r$ is the distance between the bodies.
- A body will accelerate according to Newton's second law:
$$F = ma$$
As a result of the gravitational forces all bodies will move to new positions and have new velocities.

# Gravitational N-body problem

- Given two bodies with masses $m_a$ and $m_b$, the gravitational force is given by
$$F = G\frac{m_a m_b}{r^2},$$
where G is the gravitational constant (which is $6.67259(\pm 0.00030) \times 10^{-11} kg^{-1} m^3 s^{-2}$) and $r$ is the distance between the bodies.
- A body will accelerate according to Newton's second law:
$$F = ma$$
As a result of the gravitational forces all bodies will move to new positions and have new velocities.
- For a precise numeric description, differential equations would be used (with $F = m\, dv/dt$ and $v = dx/dt$). However an exact closed form solution is not known for $n > 3$. Instead a discrete event-driven simulation is done.

# Simulating the Gravitational N-body Problem

- Suppose the time steps are $t_0, t_1, t_2, \ldots$. Let the time interval be $\Delta t$, which is as short as possible. Then we can compute the force and velocity in time interval $t+1$ as given below.

$$F = m\left(\frac{v^{t+1} - v^{t}}{\Delta t}\right) \rightarrow v^{t+1} = v^{t} + \frac{F\Delta t}{m}$$

# Simulating the Gravitational N-body Problem

- Suppose the time steps are $t_0, t_1, t_2, \ldots$. Let the time interval be $\Delta t$, which is as short as possible. Then we can compute the force and velocity in time interval $t+1$ as given below.

$$F = m\left(\frac{v^{t+1} - v^{t}}{\Delta t}\right) \rightarrow v^{t+1} = v^{t} + \frac{F\Delta t}{m}$$

- New positions for the bodies can be computed using the velocity as follows:

$$x^{t+1} - x^{t} = v\Delta t$$

# Simulating the Gravitational N-body Problem

- Suppose the time steps are $t_0, t_1, t_2, \ldots$. Let the time interval be $\Delta t$, which is as short as possible. Then we can compute the force and velocity in time interval $t+1$ as given below.

$$F = m\left(\frac{v^{t+1} - v^t}{\Delta t}\right) \rightarrow v^{t+1} = v^t + \frac{F\Delta t}{m}$$

- New positions for the bodies can be computed using the velocity as follows:

$$x^{t+1} - x^t = v\Delta t$$

- Once bodies move to new positions, the forces change and the computation has to be repeated. The velocity is not actually constant over $\Delta t$. Hence an approximate answer is obtained. A leap-frog computation can help smooth out the approximation. In a leap-frog computation the position and velocity are computed alternately.

# Simulating the Gravitational N-body Problem

- Suppose the time steps are $t_0, t_1, t_2, \ldots$. Let the time interval be $\Delta t$, which is as short as possible. Then we can compute the force and velocity in time interval $t+1$ as given below.

$$F = m\left(\frac{v^{t+1} - v^t}{\Delta t}\right) \rightarrow v^{t+1} = v^t + \frac{F\Delta t}{m}$$

- New positions for the bodies can be computed using the velocity as follows:

$$x^{t+1} - x^t = v\Delta t$$

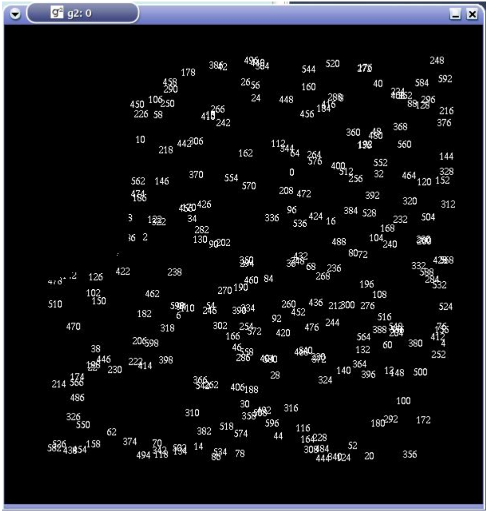
- Once bodies move to new positions, the forces change and the computation has to be repeated. The velocity is not actually constant over $\Delta t$. Hence an approximate answer is obtained. A leap-frog computation can help smooth out the approximation. In a leap-frog computation the position and velocity are computed alternately.

$$F^t = m\left(\frac{v^{t+1/2} - v^{t-1/2}}{\Delta t}\right), \rightarrow \; v^{t+1/2} = v^{t-1/2} + \frac{F\Delta t}{m}, \; x^{t+1} - x^t = v^{t+1/2}\Delta t$$

where positions are computed for $t, t+1, t+2, \ldots$ and the velocities are computed for $t+1/2, t+3/2, t+5/2, \ldots$.

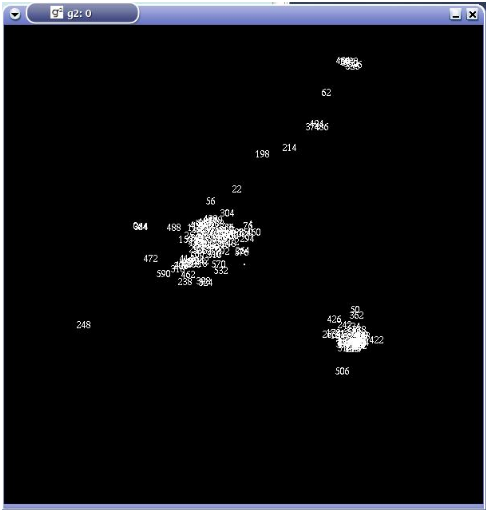
# N-body Simulation Example

Initial conditions: 300 bodies in a 2-dimensional space



# N-body Simulation Example

300 bodies after 500 steps of simulation



# Three-dimensional Space

- In 3-dimensional space, the position of two bodies $a$ and $b$ are given by $(x_a, y_a, z_a)$ and $(x_b, y_b, z_b)$ respectively. Then the distance between the bodies is:

$$
\begin{aligned}
r &= \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2} \\
F_x &= \frac{Gm_a m_b}{r^2}\left(\frac{x_b - x_a}{r}\right) \\
F_y &= \frac{Gm_a m_b}{r^2}\left(\frac{y_b - y_a}{r}\right) \\
F_z &= \frac{Gm_a m_b}{r^2}\left(\frac{z_b - z_a}{r}\right)
\end{aligned}
$$

# Three-dimensional Space

- In 3-dimensional space, the position of two bodies $a$ and $b$ are given by $(x_a, y_a, z_a)$ and $(x_b, y_b, z_b)$ respectively. Then the distance between the bodies is:

$$
\begin{aligned}
r &= \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2} \\
F_x &= \frac{G m_a m_b}{r^2} \left( \frac{x_b - x_a}{r} \right) \\
F_y &= \frac{G m_a m_b}{r^2} \left( \frac{y_b - y_a}{r} \right) \\
F_z &= \frac{G m_a m_b}{r^2} \left( \frac{z_b - z_a}{r} \right)
\end{aligned}
$$

- Similarly, the velocity is resolved in three directions.

# Three-dimensional Space

- In 3-dimensional space, the position of two bodies $a$ and $b$ are given by $(x_a, y_a, z_a)$ and $(x_b, y_b, z_b)$ respectively. Then the distance between the bodies is:

$$
\begin{aligned}
r &= \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2} \\
F_x &= \frac{G m_a m_b}{r^2}\left(\frac{x_b - x_a}{r}\right) \\
F_y &= \frac{G m_a m_b}{r^2}\left(\frac{y_b - y_a}{r}\right) \\
F_z &= \frac{G m_a m_b}{r^2}\left(\frac{z_b - z_a}{r}\right)
\end{aligned}
$$

- Similarly, the velocity is resolved in three directions.
- For simulation, we can use a fixed 3-dimensional space.

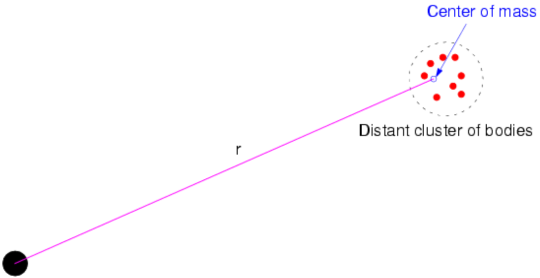
# Sequential Code for N-Body Problem

```
nbody(x, y, z, n)
    for (t=0; t<max; t++) {
        for (i=0; i<n; i++) {
            Fx ← compute_force_x(i)
            Fy ← compute_force_y(i)
            Fz ← compute_force_z(i)
            vx[i]_new ← vx[i] + Fx * dt/m
            vy[i]_new ← vy[i] + Fy * dt/m
            vz[i]_new ← vz[i] + Fz * dt/m
            x[i]_new ← x[i] + vx[i]_new * dt
            y[i]_new ← y[i] + vy[i]_new * dt
            z[i]_new ← z[i] + vz[i]_new * dt
        }
        for (i=0; i < n; i++) {
            x[i] ← x[i]_new, y[i] ← y[i]_new, z[i] ← z[i]_new
            v[i] ← v[i]_new
        }
    }
```

$\Theta(n^2)$ per iteration.

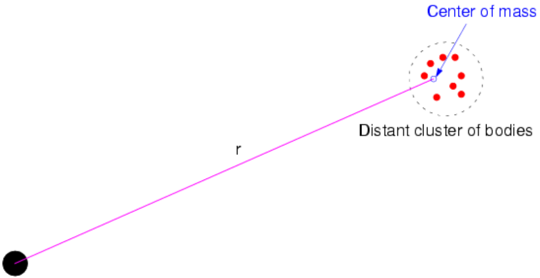
# Improving the Sequential Algorithm

- A cluster of distant bodies can be approximated as a single distant body with the total mass of the cluster sited at the center of the mass of the cluster.



# Improving the Sequential Algorithm

- A cluster of distant bodies can be approximated as a single distant body with the total mass of the cluster sited at the center of the mass of the cluster.




- When to use clustering? Suppose the original space is of dimension $d \times d \times d$,and the distance to the center of the mass of the cluster is $r$. Then we want to use clustering when

$$r \geq \frac{d}{\theta}, \quad \text{where } \theta \text{ is a constant, typically} \leq 1.0$$

# Parallel N-Body: Attempt I

- Each process is responsible for $n/p$ bodies, where $p$ is the total number of processes. Each process computes the new velocity and new position and then sends them to all other processes so they can compute the new force for the next round.

# Parallel N-Body: Attempt I

- Each process is responsible for $n/p$ bodies, where $p$ is the total number of processes. Each process computes the new velocity and new position and then sends them to all other processes so they can compute the new force for the next round.
- Even with clustering, the number of messages will be very high. Also computation of the force is still $O(n^2)$.

# Parallel N-Body: Attempt I

- Each process is responsible for $n/p$ bodies, where $p$ is the total number of processes. Each process computes the new velocity and new position and then sends them to all other processes so they can compute the new force for the next round.
- Even with clustering, the number of messages will be very high. Also computation of the force is still $O(n^2)$.
- Sequentially, there is a better algorithm (Barnes-Hut Algorithm) that is $O(n \lg n)$ on the average.

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.
  - Create a node to represent the cube for the space. Connect to parent if there is any. Next divide the cube representing the space into eight subcubes (four for a quadtree).

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.
  - Create a node to represent the cube for the space. Connect to parent if there is any. Next divide the cube representing the space into eight subcubes (four for a quadtree).
  - If a subcubes does not contain any body, it is eliminated.

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.
  - Create a node to represent the cube for the space. Connect to parent if there is any. Next divide the cube representing the space into eight subcubes (four for a quadtree).
  - If a subcubes does not contain any body, it is eliminated.
  - If a subcube contains one body, then create a leaf node representing that body.

# Barnes-Hut Algorithm

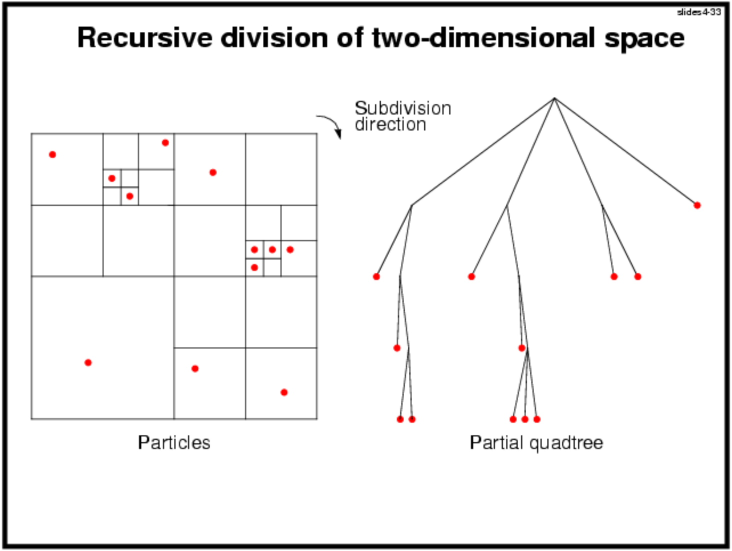
- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.
  - Create a node to represent the cube for the space. Connect to parent if there is any. Next divide the cube representing the space into eight subcubes (four for a quadtree).
  - If a subcubes does not contain any body, it is eliminated.
  - If a subcube contains one body, then create a leaf node representing that body.
  - If a subcube contains more than one body, then repeat this scheme recursively.

# Barnes-Hut Algorithm

- Uses a octtree data structure (quadtree for 2-dimensional space) to represent the 3-dimensional space.
- Using a better data structure cuts down the average run-time to $O(n \lg n)$ time!
- A octtree is a tree where each node has no more than eight child nodes. Similarly a quadtree is a tree where each node has no more than 4 child nodes. The octtree is built using the following divide-and-conquer scheme.
  - Create a node to represent the cube for the space. Connect to parent if there is any. Next divide the cube representing the space into eight subcubes (four for a quadtree).
  - If a subcubes does not contain any body, it is eliminated.
  - If a subcube contains one body, then create a leaf node representing that body.
  - If a subcube contains more than one body, then repeat this scheme recursively.
- After the construction of the tree, total mass and center-of-mass information is propagated from the bodies (leaf nodes) towards the root.

Reference: `http://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation`

# Barnes-Hut quadtree example



# Barnes-Hut Algorithm

**tree-nbody(n)**

```
for (t=0; t<max t++) {
    build_octtree() //builds tree top-down
    compute_mass() //works bottom-up on the tree
    compute_force()
    update() //update positions and velocities
}
```

# Barnes-Hut Algorithm

```
tree-nbody(n)
  for (t=0; t<max t++) {
    build_octtree() //builds tree top-down
    compute_mass() //works bottom-up on the tree
    compute_force()
    update() //update positions and velocities
  }
```

- The routines build_octtree(), compute_mass() and compute_force() take $O(n \lg n)$ time on an average.
- The total mass stored at each node is the sum of the total masses at its child nodes.

$$M = \sum_{i=0}^{7} m_i$$

- The center of mass is based on the positions and masses of the up to eight child nodes of each node.

$$x = \frac{1}{M} \sum_{i=0}^{7} m_i x_i$$

# Parallel N-Body: Attempt II

- We can partition the octtree among $p$ processes. Each process works on one subtree. The partitioning would have to be done deep enough to have $p$ subtrees. The top few levels can be duplicated on each process.

# Parallel N-Body: Attempt II

- We can partition the octtree among $p$ processes. Each process works on one subtree. The partitioning would have to be done deep enough to have $p$ subtrees. The top few levels can be duplicated on each process.
- However the octtree is, in general, very unbalanced. So any static partitioning scheme is not likely to be very effective. We will need to use some kind of dynamic load balancing but it may end up requiring a lot of messages.

# Parallel N-Body: Attempt II

- We can partition the octtree among $p$ processes. Each process works on one subtree. The partitioning would have to be done deep enough to have $p$ subtrees. The top few levels can be duplicated on each process.
- However the octtree is, in general, very unbalanced. So any static partitioning scheme is not likely to be very effective. We will need to use some kind of dynamic load balancing but it may end up requiring a lot of messages.
- There is another N-body algorithm that also runs in $O(n \lg n)$ time but uses a balanced tree by design. In fact, this algorithm was designed for parallel computing. This algorithm is known as **Orthogonal Recursive Bisection**.

# Orthogonal Recursive Bisection (ORB)

We will describe the orthogonal recursive bisection for the two-dimensional case. *Reference*: J. Salmon, Ph.D. Thesis.

- Find a vertical line that divides the area into two areas each with an equal number of bodies.

# Orthogonal Recursive Bisection (ORB)

We will describe the orthogonal recursive bisection for the two-dimensional case. *Reference*: J. Salmon, Ph.D. Thesis.

- Find a vertical line that divides the area into two areas each with an equal number of bodies.
- For each area, a horizontal line is found that divides it into two areas with an equal number of bodies.

# Orthogonal Recursive Bisection (ORB)

We will describe the orthogonal recursive bisection for the two-dimensional case. *Reference*: J. Salmon, Ph.D. Thesis.

- Find a vertical line that divides the area into two areas each with an equal number of bodies.
- For each area, a horizontal line is found that divides it into two areas with an equal number of bodies.
- Repeat above two steps until there are as many areas as processes. At that one process is assigned to each area.

How to find the vertical/horizontal line that bisects the set of points?

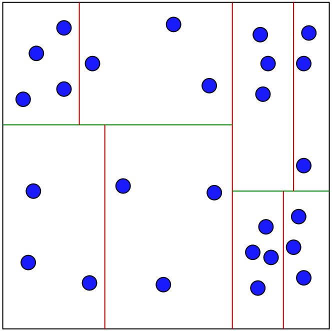
# Orthogonal Recursive Bisection (ORB)

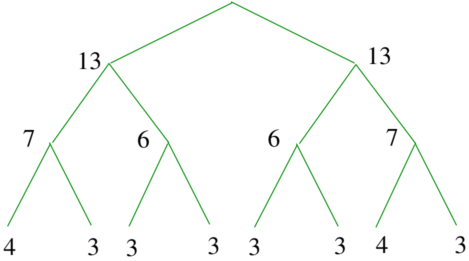
We will describe the orthogonal recursive bisection for the two-dimensional case. *Reference*: J. Salmon, Ph.D. Thesis.

- Find a vertical line that divides the area into two areas each with an equal number of bodies.
- For each area, a horizontal line is found that divides it into two areas with an equal number of bodies.
- Repeat above two steps until there are as many areas as processes. At that one process is assigned to each area.

How to find the vertical/horizontal line that bisects the set of points?
See chapter on (Medians and Order Statistics) in *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein.

# ORB example






26 bodies, 8 processes