N-BODY SOLVERS

PRESENTED BY

VIDYA NARASIMHAN

WHAT IS A N-BODY SOLVER ?

- **Problem** : Find the positions and velocities of a collection of interacting particles over a period of time
- **Definition** : An *n*-body solver is a program that finds the solution to an *n*-body problem by simulating the behavior of the particles.
- **Input to System** : mass, position, and velocity of each particle at the start of the simulation
- **Output of the System** : position and velocity of each particle at a sequence of user-specified times, or simply the position and velocity of each particle at the end of a user-specified time period.

APPLICATIONS

- Astro-Physics : To know the positions and velocities of a collection of stars,
- Chemistry : To know the positions and velocities of a collection of molecules or atoms

THE PROBLEM

- let's write an *n*-body solver that simulates the motions of planets or stars.
- We'll use Newton's second law of motion and his law of universal gravitation to determine the positions and velocities .

$$\mathbf{f}_{qk}(t) = -\frac{G m_q m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3}\left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right].$$

EXPANSION

- $\boldsymbol{f}_{\boldsymbol{qk}}$ **(t) :** force on particle *'q'* exerted by particle 'k' .
- $\boldsymbol{s}_{\boldsymbol{q}}\ (t)$: Position of particle 'q' at time 't' .
- s_k (t) : Position of particle 'k' at time 't' .
- G : Gravitational constant (6.673 → 10 11m3/(kg · s2))
- m_q and m_k : Mass of particles 'q' and 'k'
- $|s_q(t) - s_k(t)|$: Distance from particle *k* to particle *q*

TOTAL FORCE ON A PARTICLE

$$\mathbf{F}_q(t) = \sum_{\substack{k=0 \\ k \neq q}}^{n-1} \mathbf{f}_{qk} = -Gm_q \sum_{\substack{k=0 \\ k \neq q}}^{n-1} \frac{m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3} \left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right].$$

- F_q(t) = m_q(t) * a_q(t) = m_q(t) * s_q'' (t)

$$\mathbf{s}_q''(t) = -G \sum_{\substack{j=0 \\ j \neq q}}^{n-1} \frac{m_j}{\left|\mathbf{s}_q(t) - \mathbf{s}_j(t)\right|^3} \left[\mathbf{s}_q(t) - \mathbf{s}_j(t)\right].$$

INPUT AND OUTPUT

- Required Output : find the positions and velocities at the times $t = 0, \Delta t, 2\Delta t, \ldots, T\Delta t$,
- Given Input : n, the number of particles, Δt, T, and, for each particle, its mass, its initial position, and its initial velocity.
- Assumption : In a fully general solver, the positions and velocities would be three-dimensional vec- tors, but in order to keep things simple, we'll assume that the particles will move in a plane, and we'll use two-dimensional vectors instead.
-

SERIAL SOLUTION

In outline, a serial *n*-body solver can be based on the following pseudocode:

```
Get input data;
for each timestep {
   if (timestep output) Print positions and velocities of
       particles;
   for each particle q
      Compute total force on q;
   for each particle q
      Compute position and velocity of q;
}
Print positions and velocities of particles;
```

SERIAL SOLUTION

```
for each particle q {
   for each particle k != q {
      x_diff = pos[q][X] - pos[k][X];
      y_diff = pos[q][Y] - pos[k][Y];
      dist = sqrt(x_diff*x_diff + y_diff*y_diff);
      dist_cubed = dist*dist*dist;
      forces[q][X] -= G*masses[q]*masses[k]/dist_cubed * x_diff;
      forces[q][Y] -= G*masses[q]*masses[k]/dist_cubed * y_diff;
   }
}
```

$$
\begin{bmatrix}
0 & \mathbf{f}_{01} & \mathbf{f}_{02} & \cdots & \mathbf{f}_{0,n-1} \\
-\mathbf{f}_{01} & 0 & \mathbf{f}_{12} & \cdots & \mathbf{f}_{1,n-1} \\
-\mathbf{f}_{02} & -\mathbf{f}_{12} & 0 & \cdots & \mathbf{f}_{2,n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
-\mathbf{f}_{0,n-1} & -\mathbf{f}_{1,n-1} & -\mathbf{f}_{2,n-1} & \cdots & 0
\end{bmatrix}.
$$

```
for each particle q
   forces[q] = 0;
for each particle q {
   for each particle k > q {
      x_diff = pos[q][X] − pos[k][X];
      y_diff = pos[q][Y] − pos[k][Y];
      dist = sqrt(x_diff*x_diff + y_diff*y_diff);
      dist_cubed = dist*dist*dist;
      force_qk[X] = G*masses[q]*masses[k]/dist_cubed * x_diff;
      force_qk[Y] = G*masses[q]*masses[k]/dist_cubed * y_diff

      forces[q][X] += force_qk[X];
      forces[q][Y] += force_qk[Y];
      forces[k][X] −= force_qk[X];
      forces[k][Y] −= force_qk[Y];
   }
}
```

Program 6.1: A reduced algorithm for computing *n*-body forces

USING TANGENT TO APPROXIMATE A FUNCTION

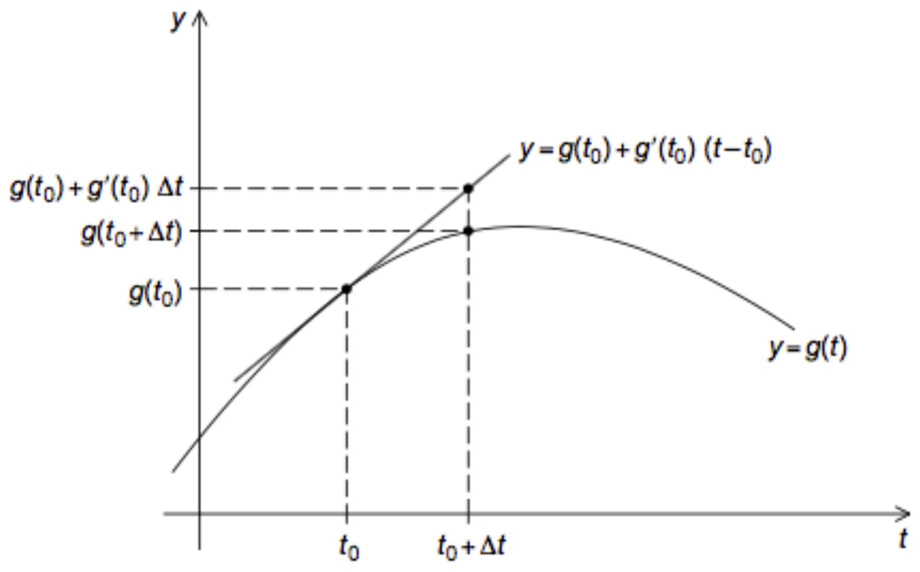


FIGURE 6.1

Using the tangent line to approximate a function

$$g(t+\Delta t) \approx g(t_0) + g'(t_0)(t+\Delta t - t) = g(t_0) + \Delta t g'(t_0).$$

$$\mathbf{s}_q(\Delta t) \approx \mathbf{s}_q(0) + \Delta t\,\mathbf{s}'_q(0) = \mathbf{s}_q(0) + \Delta t\,\mathbf{v}_q(0),$$

$$\mathbf{v}_q(\Delta t) \approx \mathbf{v}_q(0) + \Delta t\,\mathbf{v}'_q(0) = \mathbf{v}_q(0) + \Delta t\,\mathbf{a}_q(0) = \mathbf{v}_q(0) + \Delta t \frac{1}{m_q}\mathbf{F}_q(0).$$

```
pos[q][X] += delta_t*vel[q][X];
pos[q][Y] += delta_t*vel[q][Y];
vel[q][X] += delta_t/masses[q]*forces[q][X];
vel[q][Y] += delta_t/masses[q]*forces[q][Y];
```

DATA STRUCTURE

- For each particle we need to know ,
  - Mass
  - Position
  - Initial velocity
  - Accelaration
  - Total force acting on it
- For each particle it suffices to store its mass and the current value of its position, velocity, and force.
- We could store these four variables as a struct and use an array of structs to store the data for all the particles.

PARALLELIZING THE *N*-BODY SOLVERS

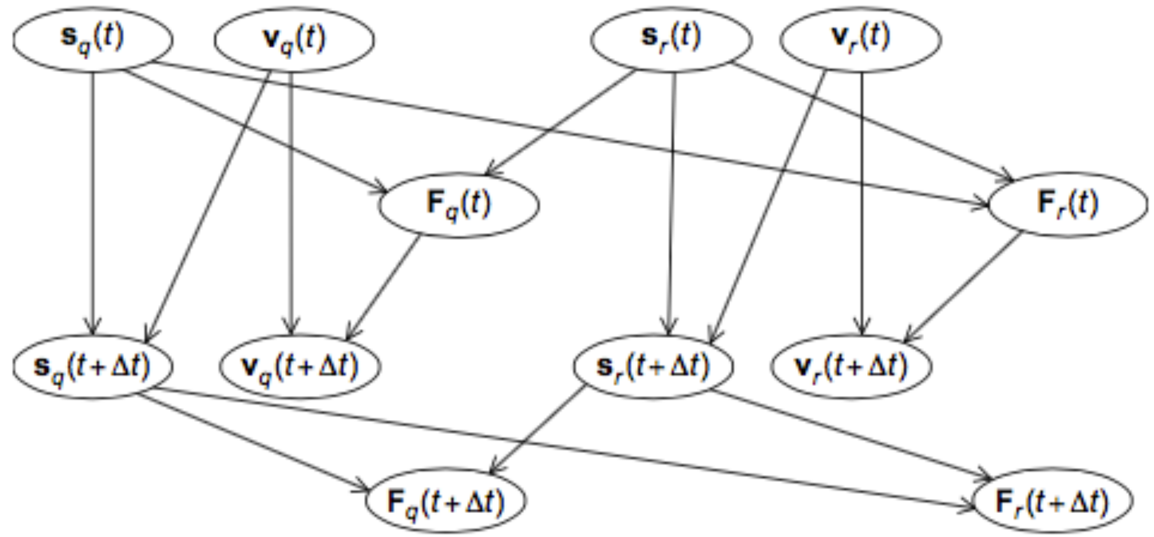
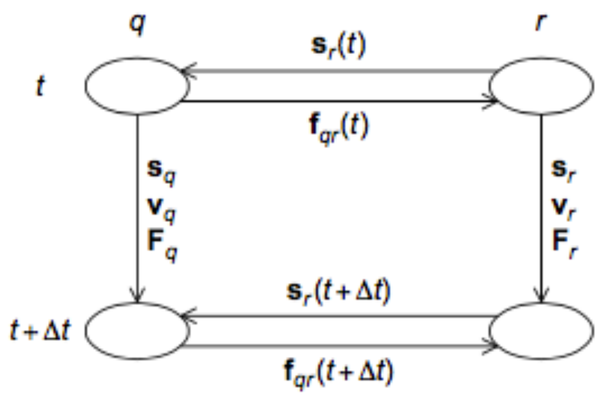


FIGURE 6.3

Communications among tasks in the basic *n*-body solver



HOW TO SPLIT THE TASK BETWEEN THREADS ?

- If we have *n* particles and *T* timesteps, then there will be *nT* tasks in both the basic and the reduced algorithm.
- Astrophysical *n*-body problems typically involve thousands or even millions of par- ticles, so *n* is likely to be several orders of magnitude greater than the number of available cores.
- However, *T* may also be much larger than the number of available cores. So, in principle, we have two “dimensions” to work with when we map tasks to cores
- Attempting to assign tasks associated with a single particle at different timesteps to different cores won’t work very well. Before estimating $\mathbf{s}q(t+1t)$ and $\mathbf{v}q(t+1t)$, Euler’s method must “know” $\mathbf{s}q(t)$, $\mathbf{v}q(t)$, and $\mathbf{a}q(t)$.

- Before estimating **s**q(t+1t) and **v**q(t+1t), Euler's method must "know" **s**q(t), **v**q(t), and **a**q(t). Thus, if we assign particle q at time t to core $c0$, and we assign particle q at time $t + \Delta t$ to core $c1 \neq c0$, then we'll have to communicate **s**q(t),**v**q(t), and **F**q(t) from $c0$ to $c1$.
- Of course, if particle q at time t and particle q at time t + 1t are mapped to the same core, this communication won't be necessary, so once we've mapped the task consisting of the calculations for particle q at the first timestep to core $c0$, we may as well map the subsequent computations for particle q to the same cores, since we can't simultaneously execute the computations for particle q at two different timesteps.
- Thus, mapping tasks to cores will, in effect, be an assignment of particles to cores.
- Assignment of particles to cores that assigns roughly n/thread count particles to each core will do a good job of balancing the workload among the cores

n-body solver using OpenMP

The n-body problem

- Find the positions and velocities of a collection of interacting particles over a period of time
- For example, **an astrophysicist** might want to know the positions and velocities of a collection of stars, while **a chemist** might want to know the positions and velocities of a collection of molecules or atoms

n-body solvers

An n-body solver is a program that finds the solution to an n-body problem by simulating the behavior of the particles.

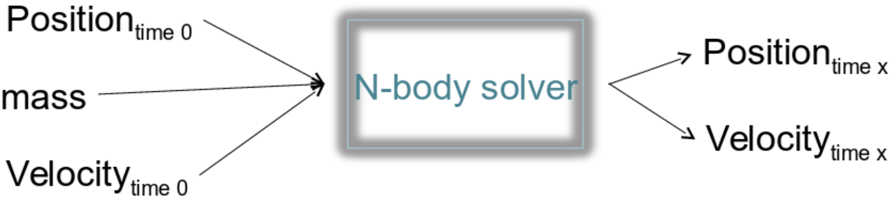


Figure: n-body solver

n-body solvers example

- An n-body solver that simulates the motions of planets or stars
- We use Newton's second law of motion and law of universal gravitation

$$\mathbf{f}_{qk}(t) = -\frac{G m_q m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3}\left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right].$$

Figure: Force on a particle q exerted by a particle k

G- gravitational constant

n-body solvers serial implementation

```
Get input data;
for each timestep {
   if (timestep output) Print positions and velocities of
       particles;
   for each particle q
      Compute total force on q;
   for each particle q
      Compute position and velocity of q;
}
Print positions and velocities of particles;
```

Figure: Serial pseudocode

Total force on a particle

The total force on a particle can be found by adding the forces due to all the particles

$$\mathbf{F}_q(t) = \sum_{\substack{k=0 \\ k\neq q}}^{n-1} \mathbf{f}_{qk} = -Gm_q \sum_{\substack{k=0 \\ k\neq q}}^{n-1} \frac{m_k}{\left|\mathbf{s}_q(t) - \mathbf{s}_k(t)\right|^3} \left[\mathbf{s}_q(t) - \mathbf{s}_k(t)\right].$$

```
for each particle q {
   for each particle k != q {
      x_diff = pos[q][X] - pos[k][X];
      y_diff = pos[q][Y] - pos[k][Y];
      dist = sqrt(x_diff*x_diff + y_diff*y_diff);
      dist_cubed = dist*dist*dist;
      forces[q][X] -= G*masses[q]*masses[k]/dist_cubed * x_diff;
      forces[q][Y] -= G*masses[q]*masses[k]/dist_cubed * y_diff;
   }
}
```

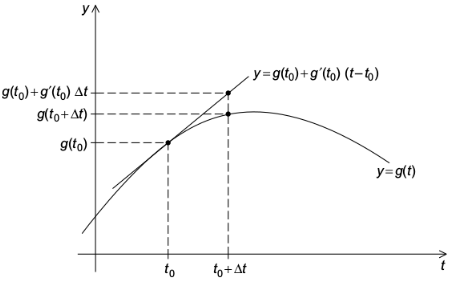
Newton's Third law

$$\begin{bmatrix} 0 & \mathbf{f}_{01} & \mathbf{f}_{02} & \cdots & \mathbf{f}_{0,n-1} \\ -\mathbf{f}_{01} & 0 & \mathbf{f}_{12} & \cdots & \mathbf{f}_{1,n-1} \\ -\mathbf{f}_{02} & -\mathbf{f}_{12} & 0 & \cdots & \mathbf{f}_{2,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -\mathbf{f}_{0,n-1} & -\mathbf{f}_{1,n-1} & -\mathbf{f}_{2,n-1} & \cdots & 0 \end{bmatrix}.$$

Reduced algorithm

```
for each particle q
   forces[q] = 0;
for each particle q {
   for each particle k > q {
      x_diff = pos[q][X] − pos[k][X];
      y_diff = pos[q][Y] − pos[k][Y];
      dist = sqrt(x_diff*x_diff + y_diff*y_diff);
      dist_cubed = dist*dist*dist;
      force_qk[X] = G*masses[q]*masses[k]/dist_cubed * x_diff;
      force_qk[Y] = G*masses[q]*masses[k]/dist_cubed * y_diff

      forces[q][X] += force_qk[X];
      forces[q][Y] += force_qk[Y];
      forces[k][X] −= force_qk[X];
      forces[k][Y] −= force_qk[Y];
   }
}
```



$$y = g(t_0) + g'(t_0)(t - t_0).$$

Since we're interested in the time $t = t_0 + \Delta t$, we get

$$g(t + \Delta t) \approx g(t_0) + g'(t_0)(t + \Delta t - t) = g(t_0) + \Delta t g'(t_0).$$

$$\mathbf{s}_q(\Delta t) \approx \mathbf{s}_q(0) + \Delta t\, \mathbf{s}'_q(0) = \mathbf{s}_q(0) + \Delta t\, \mathbf{v}_q(0),$$

$$\mathbf{v}_q(\Delta t) \approx \mathbf{v}_q(0) + \Delta t\, \mathbf{v}'_q(0) = \mathbf{v}_q(0) + \Delta t\, \mathbf{a}_q(0) = \mathbf{v}_q(0) + \Delta t \frac{1}{m_q} \mathbf{F}_q(0).$$

Pseudocode to compute the position and velocity

```
pos[q][X] += delta_t*vel[q][X];
pos[q][Y] += delta_t*vel[q][Y];
vel[q][X] += delta_t/masses[q]*forces[q][X];
vel[q][Y] += delta_t/masses[q]*forces[q][Y];
```

Parallelizing the basic solver using OpenMP

```
   for each timestep {
      if (timestep output) Print positions and velocities of
          particles;
#     pragma omp parallel for
      for each particle q
         Compute total force on q;
#     pragma omp parallel for
      for each particle q
         Compute position and velocity of q;
   }
```

```
#  pragma omp parallel for
   for each particle q {
      forces[q][X] = forces[q][Y] = 0;
      for each particle k != q {
         x_diff = pos[q][X] - pos[k][X];
         y_diff = pos[q][Y] - pos[k][Y];

 dist = sqrt(x_diff*x_diff + y_diff*y_diff);
 dist_cubed = dist*dist*dist;
 forces[q][X] -= G*masses[q]*masses[k]/dist_cubed * x_diff;
 forces[q][Y] -= G*masses[q]*masses[k]/dist_cubed * y_diff;
   }
}
```

Parallelizing the second loop

```
#   pragma omp parallel for
    for each particle q {
       pos[q][X] += delta_t*vel[q][X];
       pos[q][Y] += delta_t*vel[q][Y];
       vel[q][X] += delta_t/masses[q]*forces[q][X];
       vel[q][Y] += delta_t/masses[q]*forces[q][Y];
    }
```

parallel directive for the outermost loop

```
#   pragma omp parallel
    for each timestep {
       if (timestep output) Print positions and velocities of
            particles;
#      pragma omp for
       for each particle q



            Compute total force on q;
#      pragma omp for
       for each particle q
            Compute position and velocity of q;
    }
```

Adding the single directive

```
#  pragma omp parallel
   for each timestep {
      if (timestep output) {
#        pragma omp single
         Print positions and velocities of particles;
      }
#     pragma omp for
      for each particle q
         Compute total force on q;
#     pragma omp for
      for each particle q
         Compute position and velocity of q;
   }
```

Parallelizing the reduced solver using OpenMP

```
#   pragma omp parallel
    for each timestep {
       if (timestep output) {
#         pragma omp single
          Print positions and velocities of particles;
       }
#      pragma omp for
       for each particle q
          forces[q] = 0.0;
#      pragma omp for
       for each particle q
          Compute total force on q;
#      pragma omp for
       for each particle q
          Compute position and velocity of q;
    }
```

Critical Directive

```
#   pragma omp critical
    {
        forces[q][X] += force_qk[X];
        forces[q][Y] += force_qk[Y];
        forces[k][X] -= force_qk[X];
        forces[k][Y] -= force_qk[Y];
    }
```

Locks

```
omp_set_lock(&locks[q]);
forces[q][X] += force_qk[X];
forces[q][Y] += force_qk[Y];
omp_unset_lock(&locks[q]);

omp_set_lock(&locks[k]);
forces[k][X] -= force_qk[X];
forces[k][Y] -= force_qk[Y];
omp_unset_lock(&locks[k]);
```