Practical Parallel Programming Intensive

SciNet HPC Consortium

9–13 May 2011

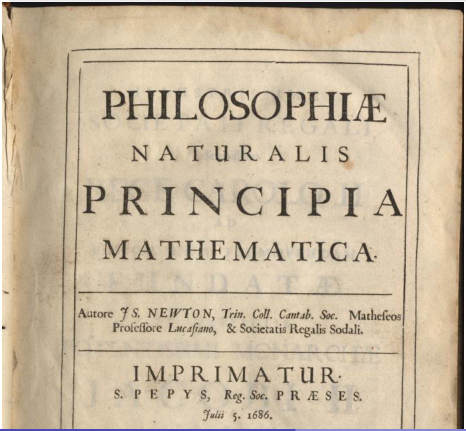
Part VIII

N-body simulations

What's an n-body simulation?

Originally (Newton)

The computation of the motion of **n** bodies (e.g. planets or stars) each of which is pulling and being pulled by every other body through the force of gravity. The force between bodies is inversely proportional to square of the distance between them.



PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore *J S. NEWTON*, *Trin. Coll. Cantab. Soc.* Matheſeos Profeſſore *Lucaſiano*, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, *Reg. Soc.* PRÆSES.
Julii 5. 1686.

More generally

Given the positions and velocities of a set of **n** bodies and given the force between them as a function of their distance, predict the positions and velocities in the future.

Gravity

- Important in most astrophysical situations: Large scale cosmology, galaxy clusters, star formation, solar system.

Molecular dynamics

Predicting the trajectories of atoms.

- Fluid dynamics where particular character is important.
- Biochemistry.
- Does not really have to be “molecular”: toy problems.

Quantum   Huh?

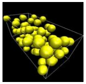
- Even in some representation of quantum systems.

Any classical system where particles are involved and the continuity approximation cannot be made (so it’s not hydrodynamics).

Different types of classical dynamics

Forces can be categorized as either short range or long range:

Short range forces



- Beyond a certain distance **r**, two bodies no longer exert a force.
- Or, this is true to good approximation.
- Example: Van der Waals forces in molecular dynamics, hard core interactions, Yukawa potentials.

Long range forces

- There is no distance beyond which two bodies do not interact.
- Interaction typically still becomes weaker with larger distance, but not enough to cut them off as an approximation.
- Example: Gravity, electric and magnetic forces.

Different types of systems

Bounded systems

The system either in bound by itself or has walls.
Isolated molecules.

Simulated infinite system

The system is finite, but mimics an infinite systems by periodic boundary condition.
Useful for solvated system!

Infinite systems

The system floats in infinite space.
Gravity!

Different computational approaches

Operator splitting

- Not the only way, but very popular because it is very stable.
- Roughly, approximate the true dynamics $\mathbf{D}$ by easier, exactly solvable factorization $\mathbf{D_1 D_2} \ldots$
- Saw this in the hydro code already: dimensional splitting.
- For n-body dynamics, these are "symplectic".
- Energy is conserved up to the order of the algorithm (i.e., no drift).

Common n-body scheme

- In exact dynamics, velocities and positions change simultaneously.
- This gets split up as:
  1. Update the positions $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{v}\delta\mathbf{t}$.
  2. Compute the forces $\mathbf{f}$.
  3. Update the velocities $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{f}\delta\mathbf{t}/\mathbf{m}$.
  4. Repeat.
- Second order, if we view velocities as half a step ahead.
- Force calculation often the bottleneck.

Force computations

The combination of type of interactions and type of boundaries determines special techniques to get optimal performance.
Simplest case is to loop over all pairs of particles and compute the force.
This takes a time $\mathbf{T} \sim \mathbf{n^2}$.

However. . .

- For short range interactions: computation only with 'neighbours'. In principle, computation only needs to take $\mathbf{T} \sim \mathbf{n}$.
  Requires binning and/or neighbour lists.
- For long range electrostatic interactions in pbc: Particle Mesh Ewald $\mathbf{T} \sim \mathbf{n \log n}$.
- For long range interactions in infinite space: tree codes, adaptive meshes, etc. $\mathbf{T} \sim \mathbf{n \log n}$.

We'll do none of these here, but focus on the loop over pairs, as this is still typically present for the short range part of the interaction.

I.e.: Gravity . . .

Difficulties

- Somehow information about all particles has to make it to all particles. Lots of communication.
- Universe is clumpy - clumps have more particles, higher acceleration.
- Good codes go like **n log n** (or even better?)

Nothing Compared to. . .

Holmberg in 1941 did analog n-body simulation.
Took light bulbs - light falls like $\mathbf{r}^2$, just like gravity.
Started off 74 light bulbs, used photovoltaic cell to measure light intensity at each light bulb, which is gravity. Calculated motion, physically moved lightbulbs, repeat. Found colliding galaxies merged, took 30 years to verify.
Claimed to get spiral structure.

Step 1: Calculate Forces

- Look at nbody.c.
- In general, particles should not be thought of as point masses. Instead, treat as diffuse blobs of matter.
- Force law

$$\mathbf{F}_{x,i} = \sum_j (\mathbf{x}_i - \mathbf{x}_j)/\mathbf{r}_{ij}^{3/2}$$

$$\mathbf{r}_{ij}^2 = \varepsilon^2 + |\mathbf{x}_i - \mathbf{x}_j|^2$$

- Extra ε^2 softens force between very close particles. So, $\mathbf{F} \to \mathbf{0}$ as $\mathbf{x}_i \to \mathbf{x}_j$. Kicks in when

$$|\mathbf{x}_i - \mathbf{x}_j| \sim \varepsilon$$

Why Soften?

- Well, lets see. Grab the new nbody. Edit it and set EPS=0.1 at the top.
- Compile and run. Look at the final output column. That is total system energy. How does it behave?
- For ref: columns are iter, step dt, simulation time elapsed, wall-clock time for step, total system energy.
- Now re-do with EPS=0.0. How did the total energy do this time?

What's Going on?

- If there is no softening, particles that interact closely have arbitrarily large accelerations.
- Must track acceleration accurately for accurate solution.
- $\boldsymbol{\delta v} = \mathbf{a}\boldsymbol{\delta t}$. If $\boldsymbol{\delta v}$ in a time step $\ll$ v $\mathbf{v_{typical}}$, system behaves. Max force at $\mathbf{r} \sim \varepsilon$, $\mathbf{a} \sim \mathbf{Gm}/\varepsilon^2$. So, want $\boldsymbol{\delta t} \ll \varepsilon^2 \mathbf{v_{typical}}/\mathbf{Gm}$. Cannot do this if $\varepsilon = \mathbf{0}$.

Oh, by the way: step 0, initial conditions

Taking random initial positions and velocities.
Can set options on command line (see nbody -h).

Let's look at the main force function...

```
void calculate_forces_fastest(NBody *data, int n) {
  for (int i=0;i<n;i++) {
    for (int j=0;j<NDIM;j++)
      data[i].f[j]=0;
    data[i].PE = 0.;
  }
  for (int i=0;i<n;i++)
    for (int j=0;j<i;j++) {
      double rsq=EPS*EPS, dx[NDIM],forcex;
      for (int k=0;k<NDIM;k++) {
        dx[k]=data[j].x[k]-data[i].x[k];
        rsq+=dx[k]*dx[k];
      }
      double ir =1./sqrt(rsq);
      rsq=ir/rsq;
      for (int k=0;k<NDIM;k++) {
        forcex=rsq*dx[k] * data[i].mass * data[j].mass * GRAVCONST;
        data[i].f[k] += forcex;
        data[j].f[k] -= forcex;
      }
      data[i].PE -= GRAVCONST * data[i].mass * data[j].mass * ir;
      data[j].PE -= GRAVCONST * data[i].mass * data[j].mass * ir;
    }
}
```

Hands-on part 1

- Copy nbody.c to nbody_omp.c
- OpenMP the force routine, without using temporary buffers
- Are there any potential race conditions. Remember atomic!
- In simplest incarnation, to avoid data race, may have to find force of jth particle on ith, but not add force to jth.
- Test with diffent sizes (default n = 1500, try e.g. n = 10000 as well).
- Scaling analysis?

Hands-on part 2

- Make a copy of nbody.c (or nbody_omp.c) called nbody_buf.c. OpenMP the force routine allowing yourself as much temporary space as you like. Should no longer need to do double work.
- How well are you scaling? If you aren't getting reasonable speed up, can you think of why? Might OpenMP be able to help you?
- Do a top while running serial and openmp. What is ratio of memory usage?
- What is your estimate of computing to reduction work, and how does it scale with n?
- Test with diffent sizes (default n = 1500, try e.g. n = 10000 as well).
- Scaling analysis?

Hands-on part 3 (tricky)

- Ideal code would scale well (no factor of 2) and have no large buffers. Code also needs to produce correct answers.
- Can you make a code, nbody_nobuf.c that does this?
- Be creative! Use whatever OpenMP arsenal you like/need: schedules, locks. . . Concepts from the matrix block-multiply may be useful.