Project 2: N-Body Simulation

Project Outline
- Model
- Implementation
- Parallel Algorithm

Reading
- MPI information on-line
  - don’t overlook tutorials
- CACM article
  - http://portal.acm.org: search for “seitz cosmic”
- Wilkinson & Allen, pp. 126-131

Project Outline
- The goals of the N-Body project are:
  - learn more about an important class of computational science problems
  - gain experience with MPI
  - examine trade-offs involved in parallel programming

  Outline:
  - implement very simple sequential method
  - verify using solar system data
  - parallelize using MPI
  - experiment with larger data sets
  - maybe consider other methods

Review: Force Calculations
- N-Body simulations are based on models of energy and force
  - a “force field” defines the sum of forces acting on a body
  - sum over different types of forces (bond, angle, electrostatic, ...)
  - sum pairwise interactions with other bodies

  Use a time-stepping method to carry out the simulation
  - define location of all bodies at \( t = 0 \)
  - for each time step:
    - compute forces acting on each body
    - compute positions of bodies at \( t_{i+1} \) as a function of positions and forces at \( t_i \)
  - we’ll use a fixed size time step, \( \Delta t \)
Mathematical Model

- For this project, the only force is gravity
  - no springs, no collisions...
- Newton's law of gravitational attraction:
  \[
  \vec{F}_{ab} = -Gm_am_b \frac{\vec{r}_a - \vec{r}_b}{|\vec{r}_a - \vec{r}_b|^3}
  \]
- Note the force pushing \(A\) toward \(B\) is balanced by an equal force pushing \(B\) toward \(A\)

Mathematical Model (cont’d)

- Notes:
  - masses \((m)\) and the gravitational constant \((G)\) are scalars
  - positions \((r)\) and forces \((F)\) are vectors
  - the norm of the difference in positions is also a scalar
  - the force is inversely proportional to the square of the distance
  - the minus sign points the force in the right direction (see next slide)

\[
\vec{F}_{ab} = -Gm_am_b \frac{\vec{r}_a - \vec{r}_b}{|\vec{r}_a - \vec{r}_b|^3}
\]

Vector Operations

- The vector difference \(A-B\) is a vector that points toward \(A\)
- Multiplying the difference by a negative constant "turns it around"
- Note the units in the force equation:
  - the RHS is a vector (the difference in positions times a bunch of scalars)
  - the LHS is a vector pointing in the direction of the force

\[
\vec{F}_{ab} = -Gm_am_b \frac{\vec{r}_a - \vec{r}_b}{|\vec{r}_a - \vec{r}_b|^3}
\]

Summing Forces

- When there are three or more bodies, the force on one body is the sum of the pairwise forces with respect to all the other bodies

\[
\vec{F}_{ab} = -Gm_am_b \frac{\vec{r}_a - \vec{r}_b}{|\vec{r}_a - \vec{r}_b|^3}
\]
Equations for Motion

- Since the bodies we’re simulating are moving, we need to include velocity and acceleration in the model.
- Both are vectors.
- From Newton’s equation derive a formula for acceleration:
  \[ \vec{F} = m \vec{a} \]
  \[ \vec{a} = \frac{\vec{F}}{m} \]
- Over a small time step, the change in velocity and position are:
  \[ \Delta \vec{v} = \vec{a} \times \Delta t \]
  \[ \Delta \vec{r} = \vec{v} \times \Delta t \]

Simulator Outline

- Assign a mass and initial position and velocity vectors for each body.
- At each time step:
  - for each body \( i \)
    - \( a = 0 \)
    - for each body \( j \neq i \)
      - \( a[i] += \text{accel}(i,j) \) // see next slide
  - for each body \( i \)
    - \( v += a[i] * dt \)
    - \( r += v * dt \) // use new \( v \)

Computing Acceleration

- The formula for the acceleration of body \( i \), as a function of the sum of the forces from each other body \( j \):
  \[ \vec{F}_{ab} = -G m_a m_b \frac{\vec{r}_a - \vec{r}_b}{||\vec{r}_a - \vec{r}_b||^3} \]
  \[ \vec{a}_i = -G \sum_{j \neq i} m_j \frac{\vec{r}_i - \vec{r}_j}{||\vec{r}_i - \vec{r}_j||^3} \]

Implementation

- The project tar file has C++ code you can use.
- \texttt{nbody.C}
  - program outline (\texttt{main()}, ...)
  - constants \( G \) and \( dt \)
    (number of seconds in a day)
  - mass and initial positions and velocities of sun and nine planets
Implementation (cont’d)

- `vector.h`
- `vector.C`
- `vdemo.C`
  - A vector class, for vectors in 3-space
  - operations on vectors (+, -, )
  - norm and other methods
- `Makefile`

Suggestion

- Compile and run `vdemo` -- make sure you understand how the vector class works
- Define a Body class with the necessary state variables (mass, position, velocity) and methods (force calculation, movement)
- Develop a sequential program:
  - Have your program print positions of each body at each time step
  - The output should be in the form of a text file that can be loaded into a program that can draw orbits (R, Matlab, etc)
    - see “visintro” slides for R commands
  - Verify the program works by testing it on the solar system data

Parallel Implementation

- A simple SPMD parallel algorithm for the N-Body project uses a “token ring”
- Use one process per body
  - note: “process”, not “processor”
  - later this term we’ll talk about mapping processes to processors
- Process i will be the home of body i
- Tokens carrying descriptions of bodies are passed between processes

Parallel Implementation (cont’d)

- Operations in process i at each time step:
  
  initialize A to 0
  create token for body i
  repeat N-1 times:
    pass token to next
    read token j from previous
    A += accel(i,j)
  move body i
Parallel Implementation (cont’d)

- To print results, have each process mail current position to process 0
- Time complexity
  - sequential: $O(n^2)$
  - parallel with $p$ processors: $O(n^2/p)$
  - parallel with one processor per process: $O(n)$

Chordal Ring

- An improvement: include acceleration $A$ as part of the token
- Have process $i$ compute $x = \text{force}(i,j)$
  - add $x/mb$ to local $A$
  - add $-x/ma$ to token $A$
- When token is half way around the ring, send it back to its home node
- Half as many force calculations
- Half as many messages

Warning: note assymetry when the number of processes is even...

Scaling Up

- The message and process overhead in this simple method will be very high when there are thousands of bodies
- A more general solution:
  - store $n/p$ bodies in each home node
  - put the same number of body descriptions in each token
- When a process receives a token, it does $(n/p)^2$ force calculations
  - don’t forget the home node interactions....

Latency Hiding

- Another important technique for minimizing communication costs is to use overlap communication and computation
  - known as latency hiding
  - uses asynchronous sends
- Every process should have one or more tokens in its input queue while it is working on its current token
Latency Hiding (cont’d)

- If a process never has an empty queue it won’t have to wait for messages.
- Communication overhead is limited to the amount of time it takes to deliver a message from the queue to the process.
  - Historical note: Intel Paragon and other successors to the original hypercube had two CPUs per node; one was dedicated to message passing.

Further Efficiency

- Topics for future lectures:
  - Restructuring the representation of bodies to use BLAS linear algebra routines.
  - $O(n \log n)$ sequential algorithms and parallelization of those methods.
  - Relation to hydrodynamics.

Movie

- Get out the popcorn...
- An n-body simulation involving 100 million bodies:
  - Milky Way vs Andromeda [http://www.news.utoronto.ca/bin/000414b.asp]
  - The two galaxies are moving toward each other at 500,000 km/hr.
    - Don’t worry: “collision” is still 3 billion years away.
  - 4 days of computing on 1152 processors at SDCC.