Lecture 18: N-Body Problems

CS178: Programming Parallel and Distributed Systems

April 9, 2001
Steven P. Reiss

I. Overview

A. The last few times we have been talking about relatively simple applications of MPI

1. First we looked at a diffeq problem

a) Heat transfer

b) Done by defining constraints on a linear matrix

c) Solving iteratively for those constraints

2. Then we discussed sorting algorithms

3. Then we discussed array algorithms

a) In particular multiplication and gaussian elimination

B. This time I want to look at nbody simulation

1. This gets us into additional techniques for parallel systems

2. This gets us into another interesting domain for parallel systems

3. This lets us look at another application

II. N-Body problems

A. Bodies that interact via inverse-squared forces

1. Force can be gravity

a) Solar system simulation (orrery)

b) Simulating galaxy formation, collision

c) Simulating solar system formation, collision, ...

2. Force can be electrical/magnetic

a) Particles in solution or a gas

b) Proteans that interact with one another

B. Why is this problem difficult

1. For 2 objects, we can get closed form solutions

a) Given initial positions, velocities, everything is known

2. For 3 or more objects, the situation is chaotic

a) Slight variation in initial values, slight deviation in computations (i.e. using finite arithmetic) causes very different solutions

b) Simulation must be used to solve the problem

3. Simulation is N**2 per step

a) Each object interacts with each other object

b) N(N-1)/2 different pairs of interaction (computed nicely)

c) Each step can only compute a little

d) Lots of steps required to cover long range of time

4. Other factors

a) 3D is somewhat more complex than 2D

b) What to do about collisions (detection and processing)

c) Should be an adaptive time step

C. Since the problem is hard and important

1. We'll attempt to parallelize it

III. Breaking the problem down

A. The naive algorithm

1. Basic algorithm

a) For each particle, sum forces for all other particles

b) Use the combined force to compute new velocity

c) Use new velocity to compute new position

2. Parallelizing this

a) Each processor handles N/p particles

b) Issues

(1) Avoiding pairs of computations

(2) Updating the values after computation

c) This is okay if N is large, but there are better approaches

B. Barnes-Hut approach

1. Think of the solar system

a) If we want to consider the effect of Jupiter on the earth

b) Do we need to consider each moon separately, or can we just consider the whole system of Jupiter and its moons

c) This can be generalized

2. We can glob together objects and use their center of mass

a) This is the mass-weighted average of their position

b) However this introduces error into the computation

3. We minimize error by looking at the angle of the ball containing all the objects

a) Let the ball be radius d, suppose we want the ball to lie within and angle T

b) Then we just need to have r >= d/T

4. Assuming a reasonable distribution of objects

a) This reduces the computation to nlogn

b) But you have to construct the tree

C. Tree-based approach

1. For each time step

a) Build the tree

b) Compute the total mass and center of gravity for the tree

c) Sum the forces using the tree

d) Update the positions of the nodes

2. How can be parallelize this

a) Each processor looks at N/p objects

b) Does all the computations for those objects

3. Problems with this approach

a) Look at the algorithm in terms of communication & processing

b) Each processor needs to have the tree handy

c) The tree needs to be recomputed after each step

IV. Tree-based approach

A. Node-balanced X-Y tree

1. Construction

a) Sort the nodes by their X coordinate

b) Split the nodes in half this way (find median)

c) Tree first branches on X at the median

d) Then recursively do the two halves on Y

e) Then repeat their halves on X

2. Diagram of how this works

3. Problem: the information is needed by all nodes

4. Can/should this be parallelized

a) Run on host and then distribute the tree

b) Run on all machines independently

c) Run in parallel

B. Force computation

1. For each node (of this processor)

a) Start at the root and check if it satisfies angle check

b) If so, use its mass/center of gravity

c) If not, repeat on the two subtrees

2. Can speed this up by determining what remote nodes to use for single tree nodes

a) But this speed up is not that large (log n) and requires lots of storage/lookup

C. Updating positions

1. This is relatively simple, applies force to each node independently

2. Computes new node velocity and position

D. The result has to be conveyed to all nodes

1. But when -- only in computing the tree the next time

a) Is the tree likely to change significantly

b) Would changes in the tree affect anything

2. **Some speedup then by only updating the tree periodically**
    a) Don't need to compute that
    b) Don't need to update the point positions from other processors

E. Effectiveness

1. **Lots of communication vs processing**
2. **Processors only busy about 20% of the time**
    a) Not particularlly optimized
    b) Could do a bit better

V. Alternative Tree-based Approach

A. Rather than breaking the nodes in half, we can break the area in half

1. **Actually break the area in quadrants (octants in 3D)**
    a) Single tree node divides both by X and Y (and Z)
    b) Quad-tree or oct-tree based approach
2. **Tree construction**
    a) Start with root with empty quadrants
    b) Add each node in turn
        (1) Check it against root, see which quadrant it falls into
        (2) If no tree node exists there, create one
        (3) If this is the first object for node, add it
        (4) Otherwise, split the node and add old and new to their proper quadrants (recursively)
3. **Pros and cons**
    a) Disadvantages -- tree can be very unbalanced (and arbitrarily deep)
    b) Advantages
        (1) Tree is geometrical in nature, d known a priori
        (2) Tree update can be done incrementally
        (3) Might not need all information all the time
4. **Implementing advantages can be tricky**

VI. Implementation

A. Passing around user data types

1. MPI lets you specify your own data types to ship around
2. These are built up from the standard types
3. Code for doing this in the example

B. Broadcasting the tree

1. Going locally to root
2. Then having root broadcast to all nodes
3. Could do all to all and then all compute