Barnes Hut
- A Broad Review

Abhinav S Bhatle
The 27th day of April, 2006
Overview

- Introduction
- Sequential Code
- OpenMP, MPI, Charm Implementations
- Performance Results
- Match with other models
Introduction

- Barnes Hut is a solution to the N-body problem which takes $O(n \log n)$ time
- The basic steps are:
  - Initialise the tree (masses, positions)
  - Compute the forces on particles
  - Update the positions and migrate the particles
- We use optimizations for force calculations on particles to reduce $O(n^2)$ to $O(n \log n)$
Data Structures

- To reduce the no. of particles in the force calculation, we use quad trees and oct trees
Sequential Code

for (t=0; t<N; t++) {
    initialise the tree
    calculate forces on each particle due to every other particle
    update the positions and velocities
    migrate the particles if required
}
Parallelization

- Common steps across all programming paradigms:
  - Divide the initialised quad tree among the processors
  - Each processor is responsible for the calculation of forces for the particles in its subtree
  - Update the positions and velocities
  - Finally migrate the particles and hence update the entire tree across all the processors
Serial Implementation

```c
void doTimeStep(TreeNode* subTree, TreeNode* root) {
    int i, j;
    if(subTree) {
        if(!subTree->isLeaf) {
            for(i=0; i<2; i++)
                for(j=0; j<2; j++)
                    if(subTree->quadrant[i][j])
                        doTimeStep(subTree->quadrant[i][j], root);
        }
        else // subtree is a leaf cell.
            calcForces(subTree, root);
    }
    return;
}
```
Writing the OpenMP code was relatively easier as compared to ??? (we’ll see soon!)

Since the data structure is not an array, so I couldn’t use simple pragmas like “parallel for”

Created threads and explicitly assigned different parts of the tree to them

The threads were supposed to do the calculations for the respective subtrees: the rest was managed by OpenMP!! 😊
Message Passing Models

- Two ways of solving the problem in this case:
  - Request other threads for the part of the tree you want to work on and then calculate the forces.
  - Send your particles’ information to the other threads and let them calculate the forces for you.

- The second approach is easier but doesn’t scale very well because you end up sending a huge no. of messages.

- The first approach is better if we use a local cache to bring in some part of the remote subtree and then use it for all the particles.
MPI Implementation

- Tough going till now! 😞
- Initial Synchronization: all the threads should know about the root and no of particles with each thread
- For all the particles in your tree, send a message to the other trees with your particles’ info and receive the answer (force) in reply
- Optimization: send info about multiple particles (at a leaf) at a time and receive the same
- Still having issues with the implementation
Charm++ Implementation

- Not different from the MPI implementation
- To have a better degree of virtualization, each chare will get a small part of the tree
- Use entry methods for asynchronous calls
  - Sending particle info and asking other threads to do the calculations
  - Sending calculated answers back to the requesting thread
Performance

- The tests for the serial and OpenMP code were performed on the SGI Altix machine at NCSA called Cobalt.

- On OpenMP, there is a speedup of 2 for two processors but then the performance gets worse (for 2000 particles).

- For 50000 particles, we get speedups on 2 and 8 processors and the time on 4 processors is almost the same as on a single processor.
Performance of OpenMP
Other Models: Good ones!

- We can look at how suitable other models are from two viewpoints
  - Ease of coding
  - Performance

- OpenMP is by far the best model in terms of coding but not the best when it comes to performance

- Shared memory models like UPC, CID which are designed for irregular data structures might be a good match

- Multithreaded programming models like Cilk would be a good match wherein you can spawn off functions
The Evil Devils!

- Models which just support shared or distributed arrays are not suitable - HPF, CAF, GA
- Cumbersome in messaging passing/ explicit synchronization models like MPI, BSP, Charm++
- Don’t know if you can even write it in these without killing yourself a multiple times: Linda and a few others ...
Thank You!!!

Questions???