UNIVERSITY OF MANITOBA

ECE 4530: Parallel Processing **Fall 2018**

Assignment 1: Divide-and-Conquer: The Barnes-Hut Algorithm

October 24, 2018

The purpose of this assignment is to program a parallel implementation of the Barnes-Hut-accelerated N-body problem for computing the force interactions between N bodies.

Assignments, like labs, can be performed in groups of two. **Code should be well documented**.

You are responsible for turning in a report that includes the answers requested at the end of each problem and a title page.

The assignment is due November 10th, 2018 at 11:59 pm (electronic submission).

Introduction

As we have discussed in class, the N-body problem requires solving for the force interactions between a system of N point-like masses. The solution can be accelerated using the Barnes-Hut algorithm which replaces clusters of interactions with a single center-of-mass (CoM) interaction whenever possible. Accepting the CoM is controlled by the parameter θ. The Barnes-Hut algorithm is easily implemented using a Tree structure. You have been provided with a basic structure and some of the functionality required. You will need to complete the missing functions to get even a serial implementation to work.

In order to parallelize the algorithm you will need to use an ORB decomposition. You should use the function you developed for Lab 3. If you were unable to complete Lab 3 please see me for a working version of ORB including a parallel bucket sort after the Lab deadline.

1.1 Instructions

Given the attached code in `Utilities.h/cpp` and `TreeNode.h/cpp` your job is to:

- Implement the `computeCoM()` and `computeForceOnBody()` functions in `TreeNode.cpp`
- Implement the `parallelBucketSort()` and `ORB()` functions in `Utilities.h/cpp` (you should have these from Lab 3 - if you are concerned with their performance I can provide you with working copies after everyone has submitted their Lab 3).
- Write a main program that:
  - Reads the number of points per processor and the value of θ from the command line.
  - Generates that many random points in three-dimensions on each processor over the unit cube with random masses between 1kg and 1000kg.
  - Partitions the points using ORB and distributes them.
  - Builds a local tree from the ORB'd points.
  - Obtains the locally essential bodies required to complete the tree and adds them to the tree - this will require implementing the function `LETBodies` in `TreeNode.cpp`. Note that this function should examine the local tree on a given processor and determine which of its points are required for the given domain.
  - Computes the force on the bodies.
  - Determines the average, minimum and maximum absolute force (each component) over all processors and outputs it to the screen (part of verification).
  - Times the important modules of the computations including the overall time and displays the timing results.
  - Displays some simple diagnostics (Hint: use the diagnostics function).

1.2 Hints - Read all of these carefully!!

- The all-to-all vector of vector function is still available. Use it!
- We will be using the `domain_t` struct to identify a domain associated with a given processor for building the LET trees. It is defined in `Utilities.h` - if you introduced another "domain" structure for your ORB routine make sure these don't conflict.
- The `TreeNode` class comes equipped with `diagnostics()` and `prune()`. You don't need to prune the tree. In fact, be very careful about pruning the tree before the LET is completely built as this will cause problems.

- The `LETBodies` function in `TreeNode.cpp` is already equipped with a method for determining the minimum distance between two boxes.
- You have been provided with a function `getPointExtent` in `Utilities.h/cpp` that will compute, given a list of bodies, the domain that encloses them. It has a flag that will make the extent local to a processor or global across all processors.
- You will need to convince me that your code works. There are a number of ways to suggest that this is the case. For example, the CoM of the local trees before adding the LET bodies should be localized to obvious regions in the unit cube. Also, the CoM of the LET on each processor should be the same. Add some output to your code to support the fact that it is performing correctly. You could add a direct calculation.
- The provided tree code **will crash** if two points are co-located in space. This shouldn't happen provided N is not too large and your random number generator is properly seeded (differently) over all processors.
- Remember that the command line value is the number of points per processor. If you want to fix N (the number of points in the global system) you will need to adjust the command line parameter appropriately).

1.3 Questions and Discussion

Q1. Run your program using one processor while changing N **in a meaningful way** for $\theta = 1$. Provide output (console dump but not all your debugging stuff please!!) for the first and last N you choose. Plot the overall execution time as a function of N (for more than just the first and last run). Include a curve that represents the theoretical performance as a function of N appropriately scaled to fit the data. Comment on the results.

Q2. Run your code using one processor for a fixed N while changing θ from 1 down to 0. Provide output for the first and last θ you choose. Plot the overall execution time as a function of θ. Comment on the results.

Q3. Run your program for a fixed N and θ for $P = 1, 2, \ldots, 5$. Provide the output for $P = 1, 2$ and 3. Plot the overall execution time as a function of P and include a curve that represents the theoretical performance as a function of P appropriately scaled to fit the data. Comment on the results.

1.4 Hand-In

Your code and the answers to the above questions including any figures in a report.