Spring 2016

The Parallelization and Optimization of the N-Body Problem using OpenMP and OpenMPI

Nicholas J. Carugati
Gettysburg College

The Parallelization and Optimization of the N-Body Problem using OpenMP and OpenMPI

Abstract
The focus of this research is exploring the efficient ways we can implement the NBody problem. The N-Body problem, in the field of physics, is a problem in which predicts or simulates the movements of planets and how they interact with each other gravitationally. For this research, we are viewing if the simulation can execute efficiently by delegating the heavy computational work through different cores of a CPU. The approach that is being used to figure this out is by integrating the parallelization API OpenMP and the message-passing library OpenMPI into the code. Rather than all the code executing on a single thread, the computational work should be individually distributed based on the current nodes within the Barnes- Hut tree data. This research is an alternative to not only simulations alone but for also bigger data (which may require more distribution on work), or for computationally expensive procedures.

Keywords
Physics, N-Body, Simulation

Disciplines
Computer Sciences | Other Astrophysics and Astronomy | Physical Processes

Comments
This paper was written based on individualized research (*CS 460*), Spring 2016.

Nicholas Carugati
CS 460 – Individualized Research

The Parallelization and Optimization of the N-Body Problem using OpenMP and OpenMPI

Abstract

The focus of this research is exploring the efficient ways we can implement the N-Body problem. The N-Body problem, in the field of physics, is a problem in which predicts or simulates the movements of planets and how they interact with each other gravitationally. For this research, we are viewing if the simulation can execute efficiently by delegating the heavy computational work through different cores of a CPU. The approach that is being used to figure this out is by integrating the parallelization API OpenMP and the message-passing library OpenMPI into the code. Rather than all the code executing on a single thread, the computational work should be individually distributed based on the current nodes within the Barnes-Hut tree data. This research is an alternative to not only simulations alone but for also bigger data (which may require more distribution on work), or for computationally expensive procedures.

Introduction

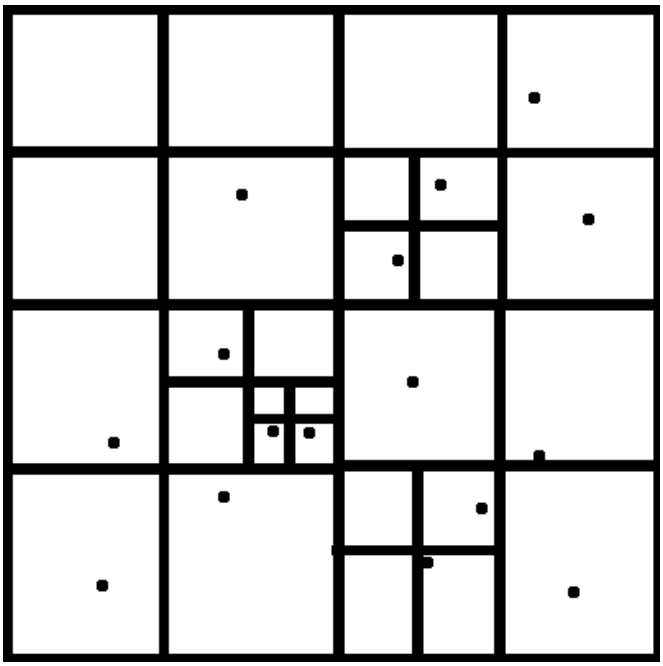
Simulations are important in how we emulate or anticipate certain activities or behaviors in science. The primary application for such simulations is in the field of Physics. When performing these simulations there is much room to efficiently execute and calculate equations in said simulations. By parallelizing these simulations with utilities such as OpenMP and OpenMPI, the work that is being done in a simulation can be evenly distributed through a n- amount of processors in which calculations can be done quickly and efficiently. The N-Body Simulation is a perfect application to how simulations can be done faster through additional utilities and libraries by evenly distributing nodes (or planets for this matter) through a finite amount of cores in a processor.

The N-Body Problem

In Physics, the N-Body Problem is a problem in which anticipates the movements of celestial objects in a bounded galaxy. A complex type of problem in the N-body simulation is globular cluster that are stars orbiting a giant star or a galactic core. A simple example of simulating the N-body problem is by anticipating all the planets in the solar system's gravitational pull to the sun. For placing the planets (nodes) into the system, they must be meticulously done through a more efficient process other than a brute forcing strategy. The approximation algorithm that will be used in this research will be the Barnes-Hut Tree Algorithm.

Barnes-Hut Tree Simulation

The Barnes-Hut Tree Simulation Algorithm will cut the amount of time it will take to insert and access nodes significantly. The Barnes-Hut tree is a quad-tree (in terms of 2D simulations) that has a fixed set of nodes with a finite amount of levels within the tree. When a Barnes-Hut tree does an insert or an access it recursively goes trough a quad-tree. When the node is at a certain point in the plane, the quad-tree creates a new level with that node inside it and assigns it to a certain region in a two dimensional space.



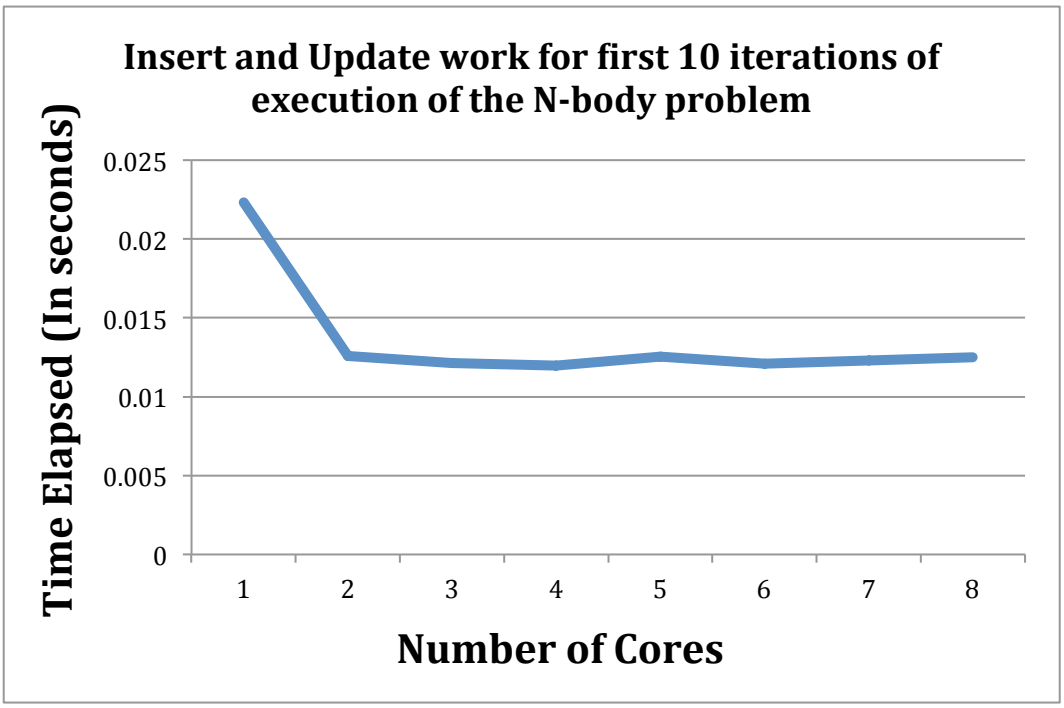
Graphical Representation of a quad-tree

Unlike a brute force algorithm such as the direct-sum algorithm (which takes the order $O(n^2)$), The Barnes-Hut Tree simulation will cut down the order time to $O(n \log(n))$. The speedup can be even more improved with the Parallelization tool OpenMP. With parallelizing the work, there will be noticeable decreases in work time as more processes are involved with the workload of the program. With OpenMPI, the work that is done for each node can be delegated to a specific processor core depending on which region the node is placed in.

Results with OpenMP

The results for OpenMP were marginal but not staggering. Times for tree insertion and update were both accounted for when monitoring the time. The time values are the time it takes to insert and update all of the nodes respectively for all iterations in the main loop that executes the program. For this research, only the first 10 iterations were recorded in the execution. The testing was done on a Linux machine,

which consisted of 4 cores with hyper threading that constitutes to 8 cores. The amount of nodes used for these trials were 1,000,000 nodes.

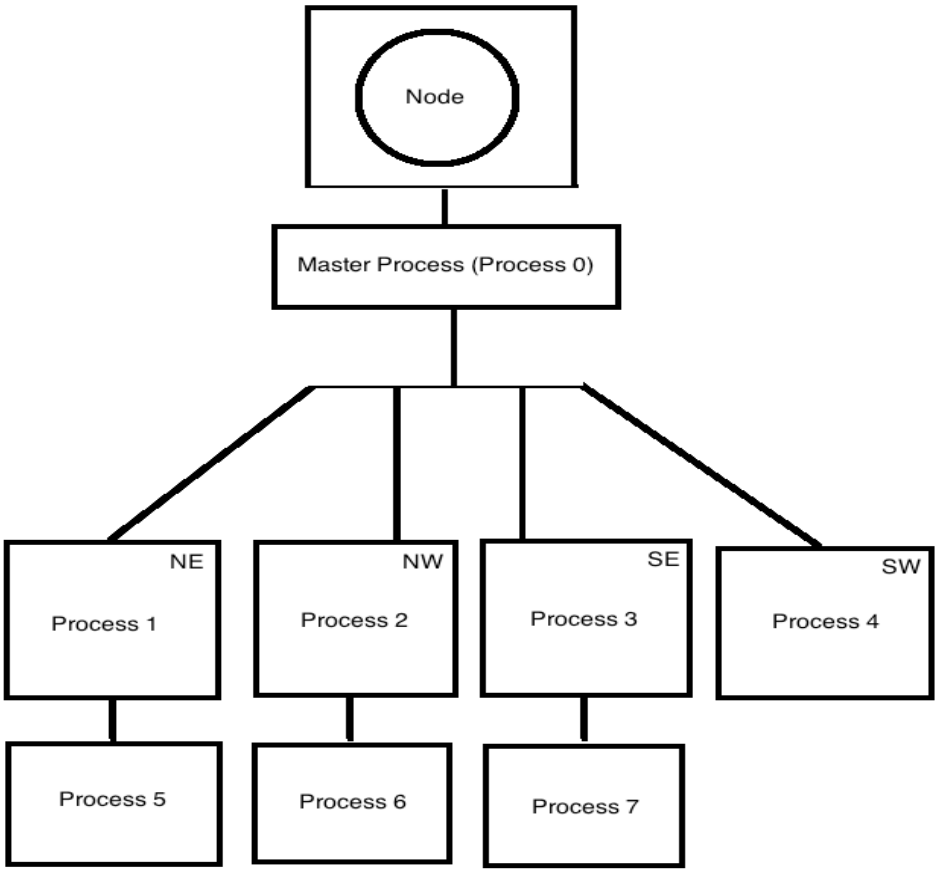


Cores	Average Time (s)
1	0.0223109
2	0.0125674
3	0.0121461
4	0.011967
5	0.0125268
6	0.0121167
7	0.012315
8	0.0125158

As shown, there is a significant speed up when 2 or more cores are incorporating in the tasking. As more cores are being applied to the execution, there is no change in speed up regardless of how many processors are added.

Approach in OpenMPI

The approach to implement the distribution of nodes through OpenMPI is by first determining where the node is inside of the quad-tree. By finding its particular position in that node, it is assigned to its proper core based on which position each core is assigned to. For example, if one node is in the NW part of the plane and Core 2 is assigned to that node in the quad-tree, then OpenMPI will use the node that is being selected in the master core (core 0) and sent out to its respective process (core 2). This type of approach is asymmetric as the main process that is being used is being served as a “boss” or a “master” process, which carries out the nodes to their respected processes to be properly calculated. With a machine that uses 8 cores in its processor, each core is assigned to an area on the quad tree in multiples of four.



The OpenMPI mapping for distributing nodes in a machine with 8 cores

The nodes are properly distributed depending on the amount of processors given. To get to the additional processes assigned to the area on the quad-tree a simple flag is checked to see if a node was placed in there recently and if the condition is fulfilled it sends the node into the secondary process (e.g.: process 5) instead of the primary (e.g.: process 1).

Conclusion

OpenMP and OpenMPI are some of the most efficient ways we can produce precise and fast output for bigger data in simulation. Since physics and astronomy usually deals with bigger data such as the amount of nodes used, and their traits such as mass and force, there is a demand for faster solutions to simulations such as parallelizing the work with OpenMP or properly delegating the work to different processes through OpenMPI. Creating faster programs has been a endless mission in computer science and the current research found here can be greatly expanded and branched out to many other solutions through other programming languages and other conceptual approaches in parallel and distributed processing.