Summary

We are going to implement a 2D approximation simulation of the collision of particles in hard-body physics, where each particle in the simulation has mass and a gravitational pull, with the Barnes Hut algorithm. Using the NVIDIA GPUs in the lab, we will optimize a CUDA implementation, drawing inspiration from a paper published by Burtscher and Pingali[1] and a paper published by NVIDIA[2]. We will measure the speedup caused by introducing various optimizations in the CUDA implementation, and we will investigate speedup and approximation accuracy tradeoffs.

[1]https://iss.oden.utexas.edu/Publications/Papers/burtscher11.pdf
[2]https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

Webpage

Our project details are hosted at 418.jtromero.com**.**

Background

Barnes-Hut Algorithm

We will be implementing the tree-based Barnes-Hut algorithm that was introduced in lecture to perform an N-body approximate simulation for particle collision. To simplify the simulation, we will refer to the location of the center of mass of a particle as the particle's position. The pseudocode for the algorithm is as follows:

```
For each timestep:
     Compute bounding box around all particles
     Construct the quadtree based on particle positions
     For each particle:
          Approximate the force acting on that particle
     Update new particle positions
```

Each timestep must be computed sequentially. The steps listed above within a timestep must also be computed sequentially (the force approximation computations could potentially be done in parallel), however each step itself could benefit from parallelism.

Computing the Bounding Box

The bounding box is the smallest rectangle that contains all particles in the simulation. This will make the space over which we construct the quadtree for one time step finite, but it does not limit where future particle locations may be. This bounding box will be represented as the root in our quadtree.

This part of the algorithm could benefit from parallelism. It boils down to finding the minimum and maximum x and y coordinates across the particles, with comparisons that can be performed in parallel because the particle positions are fixed at this time.

Constructing the Quadtree

We then construct the quadtree over the space contained in the bounding box. Partition the space into four quadrants, and represent each quadrant as a child of the root node in the quadtree. Continue to recursively partition any portion of the space that contains more than 1 particle in it until every partition has at most one particle within it. Internal nodes in the quadtree represent space partitions, and external nodes represent a particle. An example bounded box containing particles and its corresponding quadtree construction is depicted below:

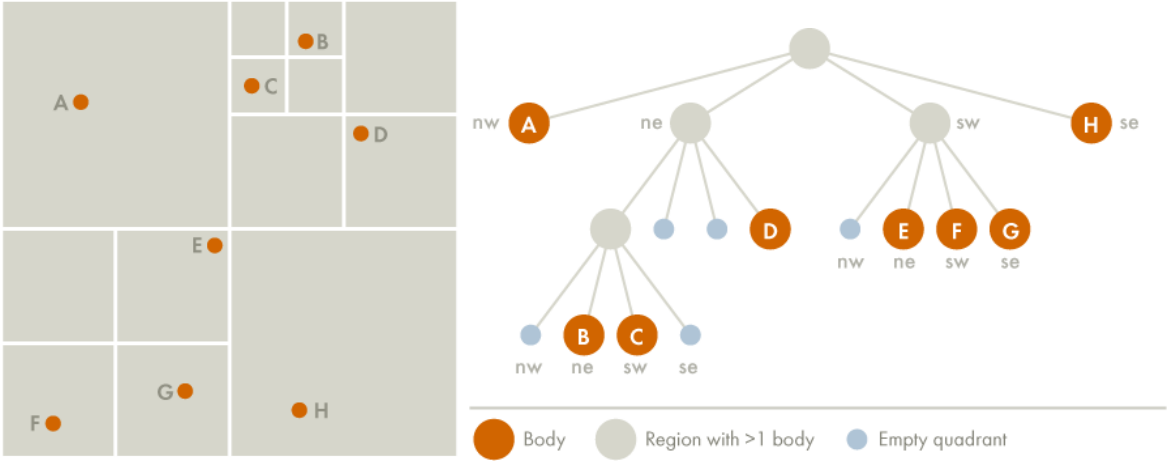

Figure 1 - Barnes-Hut quadtree example[3]

This portion of the algorithm would benefit from parallelism. Each quadrant can be partitioned in parallel because particles in different quadrants don't interfere with each other for the purposes of further partitioning. There will be one quadtree data structure shared across all threads, however due to how the quadtree is constructed, two different partitions will expand upon two different nodes.

Force Approximations

The approximation accuracy is determined by the parameter θ. The approximation comes into play as particles that are classified 'far enough' from a particular particle will be grouped together with nearby particles that are also 'far enough' from the particle for which we are computing the force acting on it. A particle/cluster of particles represented by an internal node *A* is 'far enough' from the particle *B* if the width of the space represented by *A* divided by the distance between *A*'s and *B*'s centers of mass is greater than θ. When θ = 0, this is equivalent to using no approximation as each particle's individual gravitational effect is calculated on each particle.

This portion of the algorithm would benefit from parallelism. At this time, particle locations are still fixed, so the computation of the force on each particle can be calculated in a data parallel model. In addition, there is some degree of potential parallelism when calculating the force

acting on a particle, as particles that aren't clustered together in the approximation can have its force on a particular particle calculated in parallel.

The Challenge

The tree-based Barnes-Hut algorithm is naturally and thus commonly framed in a recursive manner. Specifically, constructing and traversing the quadtree are performed recursively. However, this recursion is not ideal for CUDA programming as this limits parallelism, since each thread would be performing a recursive operation that could be further parallelized amongst more threads. Our first challenge will be to implement the Barnes-Hut in an iterative manner that allows parallelization.

From there, the main challenge comes with optimizing the CUDA implementation. The quadtrees are expected to be irregularly shaped and imbalanced since the particles are initially randomly distributed. As the simulation progresses, we expect the quadtree to remain irregularly shaped since the particles also start with a random initial velocity, which prevents all the particles from slowly gravitating towards each other. This presents two major challenges: divergent execution and many memory operations.

The amount of work required to compute the approximate force on each particle can vary greatly across each particle depending on how much quadtree traversal is required before a node is classified as 'far enough.' This makes dividing up the workload amongst threads trickier as the workload is not predictable as far as we can tell right now. For a similar reason, the memory access patterns will vary across computations and across time step iterations. Pointers will need to be reassigned as the quadtree is recomputed. There is also potential for spatial locality with the tree traversal, but this will depend on how we implement our quadtree data structure. In particular, since we are optimizing a CUDA implementation, we will need to address the challenge of taking advantage of the CUDA memory hierarchy and handling the divergent execution especially within thread blocks.

We may also face challenges with regards to synchronization as we will need to determine when it's necessary to have synchronizations between thread blocks or kernels.

Through this project, we hope to improve our skills in creating and debugging efficient parallel programs, gain a deeper understanding of CUDA programming, and expand upon what we've learned in assignment 2 to better tailor our algorithm implementation to be more CUDA friendly.

Resources

We will be using NVIDIA GPUs in the GHC labs, just as we did in assignment 2. We will be starting our code base from scratch, beginning with creating a sequential implementation of the Barnes-Hut algorithm before parallelizing and optimizing it. We found two papers that are good references for our project. The first paper by Burtscher and Pingali[1] provides guidance on high-level optimizations for a CUDA-implementation of Barnes Hut and the second paper published by NVIDIA[2] provides a more general discussion of performing N-body simulations on CUDA without a focus in particular on Barnes-Hut.

[1]https://iss.oden.utexas.edu/Publications/Papers/burtscher11.pdf
[2]https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda
[3]http://arborjs.org/docs/barnes-hut

Goals and Deliverables

Planned Goals

We plan to produce the following deliverables:

- A program that generates a set of particles with random positions, velocities, and masses given the number of particles as input. This will be used to generate data for testing.
- A sequential implementation of the Barnes-Hut algorithm that runs entirely on the CPU to use as a comparison for our parallel implementation.
- A CUDA implementation of the Barnes-Hut algorithm.
    - We are unsure about precise performance goals at this time. While we would like to achieve similar speedup observed by Burtscher and Pingali, we are using different hardware which will affect our observations. We will investigate further to derive a reasonable performance goal, but this may involve first needing to produce a Barnes-Hut implementation.
- Graphs displaying:
    - The speedup of the CUDA implementation against the sequential CPU-only implementation for varying number of particles in the simulation.
        - We will use the same parameter θ and the same number of time step iterations across all the simulations when collecting this data. We will follow Burtscher and Pingali and use problem sizes of 5000, 50,000, 500,000, 5,000,000, and 50,000,000.
    - The accuracy of the CUDA implementation simulation as a function of the parameter θ (threshold for ‘far enough’).
        - We will consider calculations with θ = 0 as the accurate values of particle positions, and use the sum of the squared distance between a particle’s approximated position and its true position as the cost metric for accuracy. So the plot will display the cost for parameter θ values of 0.05, 0.5, 1, 5, 50, and 500. We will create this plot for each of the problem sizes of 5000, 50,000, 500,000, 5,000,000, and 50,000,000 in the simulation to also observe if there are patterns with problem size and accuracy.
- A visual rendering of the particles as their location gets updated.
    - This will be similar to the circle rendering in assignment 2. However, we had issues with rendering the image on our own computers through SSH, so should we successfully implement this functionality, we may only be able to demo it directly on a GHC lab machine or have to display a screen recording of a prior run.

Extension Goals

Should we have extra time, we hope to achieve the following in order as listed:

- Optimizing the CPU-only implementation of the Barnes-Hut algorithm by introducing parallelism with pthreads.
- Speedup graphs of the parallel CPU-only implementation.
    - We would compare these graphs against the speedup graphs produced by the CUDA implementation.
- Extra parameters to tune the initial state of the particles. For example, we could ingest data for the actual positions and masses of stars.

Demo Plan

For the poster session, we will include the speedup and accuracy graphs described above in the planned goals subsection. As mentioned above, we are facing issues with SSH and XQuartz compatibility so if we are able to display a visual of the particle collision simulation on our own computers, we would include this as part of a live demo. However, if this is infeasible, we will instead include a pre-recorded visual of a simulation and include snapshots of the simulation over different timesteps.

Platform Choice

We have chosen to use CUDA over the NVIDIA GPUs in the lab. We have used the Barnes-Hut algorithm throughout this course as a case study, and implementing the CUDA version will allow us to apply what we've learned. Specifically, we can focus on handling divergent execution and taking advantage of the CUDA memory hierarchy, which is a different angle to approach the parallelism of Barnes-Hut than what we have seen in lectures which focused on workload distribution. Since we are familiar with the NVIDIA GPUs in the lab and they are our most convenient access to GPUs supporting CUDA, it makes the most sense to use them as our focus is on software implementation development and optimization.

Post-Milestone Revised Schedule

Deadline Date	Action Item Completed by Deadline	Assignee
Friday, April 19	Complete initial CUDA implementation without optimizations	Aimee + Jackson
Saturday, April 20	Produce speedup graphs for unoptimized CUDA implementation against sequential implementation	Jackson
Wednesday, April 22	Implement the following control flow optimizations discussed in the Burtscher and Pingali paper: minimize thread divergence, combine operations	Aimee
Thursday, April 25	Implement the following control flow optimizations discussed in the Burtscher	Jackson

	and Pingali paper: throttle threads, minimize control flow	
Saturday, April 27	Implement main memory optimizations discussed in the Burtscher and Pingali paper (minimize accesses, etc.)	Aimee
Friday, May 3	Implement lock optimizations discussed in the Burtscher and Pingali paper (minimize locks, etc)	Jackson
Friday, May 3	Implement visualization of bodies across time steps	Aimee
Saturday, May 4	Draft report and collect data and produce speedup and accuracy graphs	Aimee
Sunday, May 5	Complete final report; Final Report Due	Jackson
Monday, May 6	Poster Session	Aimee + Jackson