N moving body simulation
15618 Project Proposal

Team members: Akhil Kashyap and Ashwin Nayak Ullal

Website URL:

https://quantum109678.github.io/15618-course-project/

SUMMARY:

We are going to implement the N body simulation, where we simulate the movement of N bodies/particles in a 2 dimensional space under the influence of gravitational force using Pthreads, OpenMP and MPI. We will compare the Speedup achieved using these 3 approaches.

BACKGROUND:

The N-body simulation is a classic computational physics problem in which the objective is to predict the individual motions of a group of celestial objects interacting with each other gravitationally. Each body exerts a force on all others, influencing their acceleration, velocity, and ultimately, their position over time. This problem scales quadratically with the number of bodies (`N^2`), as every body interacts with every other body, making it computationally intensive, especially for large values of `N`.

The simulation involves calculating the gravitational forces between all pairs of bodies, updating their velocities and positions, and iterating this process over discrete time steps. The computationally demanding nature of this task makes it an ideal candidate for parallelization. Parallelism can be exploited in several ways: by distributing the computation of forces among processors, by parallelizing the update of body attributes, or by doing both concurrently.

The goal of parallelizing the N-body simulation is to reduce the overall computation time by dividing the work among multiple processing elements. In shared-memory systems, threads can be created via Pthreads or OpenMP to perform calculations concurrently, with OpenMP providing a higher-level, often simpler, abstraction for parallelism. Message Passing Interface (MPI) is used for distributed-memory systems where data is passed between processes running on different nodes of a cluster. Hybrid approaches can combine MPI with OpenMP to leverage both distributed and shared memory models, and for highly parallel architectures like GPUs, CUDA can be utilized to perform massive amounts of calculations in parallel.

The N-body simulation's inherent lack of data dependency during the force calculation phase makes it particularly amenable to parallel processing. Each body's new state can be computed

independently based on a common, unchanging snapshot of the system. This characteristic allows for the distribution of the simulation across multiple computational units without the need for synchronization during the calculation phase, thus providing a significant opportunity for performance gains through parallelism. The speedup and efficiency of parallel implementations can be measured by comparing the execution time to that of a sequential algorithm, particularly as the number of bodies and the computational resources vary.

THE CHALLENGE:

Challenge 1: Work Distribution Dynamics

The non-static nature of particle positions in the N-body simulation necessitates adaptive work distribution among processors to handle the constantly changing computational load. The challenge is to develop a dynamic load balancing strategy that can efficiently redistribute work in response to particle movement without incurring high overhead from synchronization and load rebalancing.

Challenge 2: Gravitational Calculations

Accurately calculating the gravitational influences between all particles is computationally demanding due to the quadratic increase in interactions with the number of particles. The challenge is to optimize the parallel computation of these forces while managing the high communication-to-computation ratio and ensuring accuracy and efficiency.

Workload and System Constraints

- **Data Locality**: The simulation lacks data locality as each particle needs information about all others, potentially leading to inefficient memory usage and cache misses.
- **Communication Overhead**: In message passing systems without a shared memory (MPI) systems, the frequent exchange of particle states can dominate the computation time, especially as particle count increases.
- **Divergent Execution**: Variations in workload distribution can lead to divergent execution paths, affecting the uniformity and efficiency of parallel processing.
- **Parallel Programming Models**: Different programming models (shared memory via Pthreads/OpenMP, distributed memory via MPI, and hybrid approaches) present unique synchronization and data-sharing challenges.

RESOURCES:

We will be starting this code from scratch.
We will be using GHC machines for our initial development purposes, and finally we will use the PSC cluster to get metrics for our implementation.

There are certain papers we would like to take inspiration from:

https://ieeexplore.ieee.org/abstract/document/6468522?casa_token=h8a9hX-Nh0AAAAAA:iNyufWmU1-z252KyVuDQUC1QqYPfujWgKYV6cLrPOn5HgUl4xVnLOIc3mdNSweRhqN1E4rWx
A massively parallel N body solver, this paper talks about different optimizations used and finally they ran a simulation of 4 million particles over 250000 cores.
https://www.sciencedirect.com/science/article/pii/S0010465511004012?casa_token=a44ibz44ylAAAAAA:sGMj-0Zb_Q-4mbIPeBemqTZWTtBtZoPICJjSseuFl_CJY70jOWqE5t9ocqOamJGmsXqDlJIaew
A similar paper, but a simulation of 2 billion particles run on 300000 cores.
https://link.springer.com/chapter/10.1007/10703040_47

GOALS AND DELIVERABLES:

[75%] represents deliverables if things go slowly
[100%] represents deliverables if things are on track
[125%] represent deliverables that we hope to achieve

[75%] An OpenMP based implementation of the N-body simulation
[75%] An MPI based implementation of the N-body simulation
[75%] Detailed analysis of the speedup achieved with different problem sizes on GHC machines and PSC nodes for both the implementations (Demo)

[100%] A pthread based implementation of the N-body simulation
[100%] Detailed analysis of the speedup achieved with different problem sizes on GHC machines and PSC nodes for all three implementations (Demo)

[125%] A CUDA implementation of the N body simulation
[125%] Detailed analysis of the speedup achieved on GPU with different problem sizes on GHC

PLATFORM CHOICE:

As we want the fastest language we will be using c++ for our computing needs. We will be running our simulations on the PSC cluster. For the massive number of particles in the system, we would be needing as many cores there are available, so that we could spread across this massive workload into multiple parts. Hence we will be using PSC, with upto 128core per node and upto 27400 cores in total, this would totally suit our needs.

SCHEDULE:

03/27 - 03/30: Exploring the references and getting the starter code ready as we are starting from scratch
04/01 - 04/07: Implement the sequential version of the algorithm and the OpenMP version

04/08 - 04/15 (includes Milestone report): Implement the parallel version using pthreads
04/16 - 04/23: Implement the MPI version of the parallel algorithm
04/24 - 04/31: CUDA implementation of the algorithm
05/01 - 05/05: Final report with detailed analysis of speedup with different problem sizes