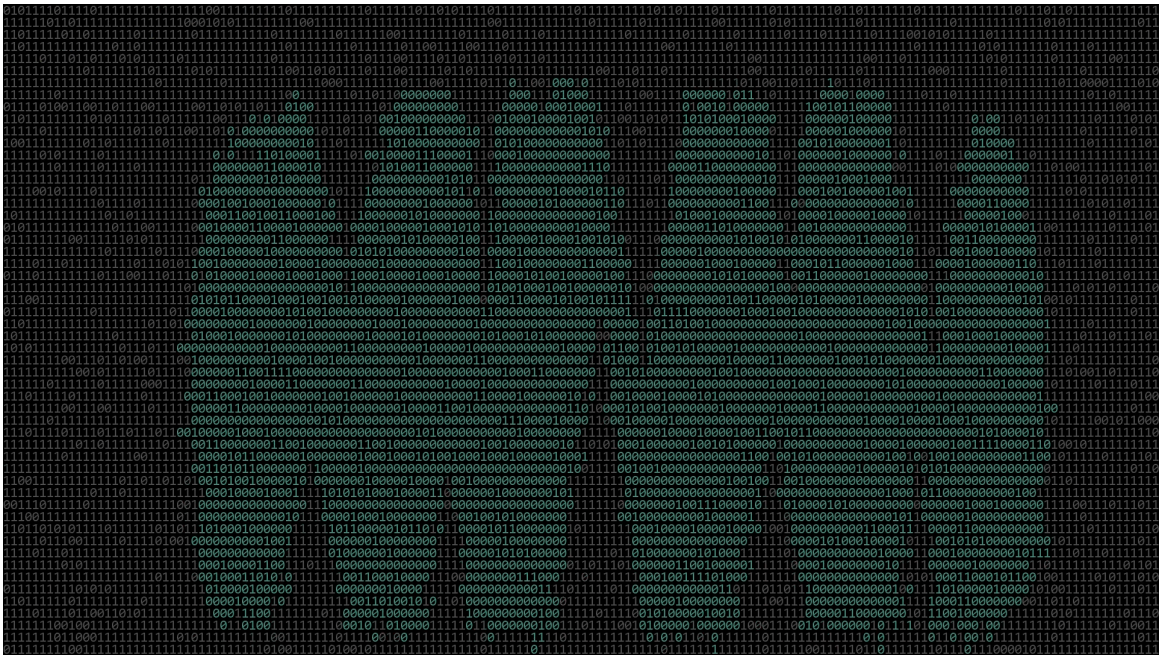
Particle Simulation and Rendering on Edge Devices

Max Locke
Introduction to Parallel Programming and Algorithms Term Project

1. Motivation / Overview
2. Technical Specifications
3. Conclusion

Motivation / Overview

Recently, I've been working with some artistic rendering for a theatrical performance I've been developing. This involves creating a video to go alongside it, a binary display with dynamically adjusting colors and values as I wish. The dynamic behavior of the visualization made it good for parallelization, as it featured thousands frames being generated, each featuring hundreds of bits that needed to be generated based on relative bit values, random chance, and given parameters. However, parallelization of this project proved very simple and not particularly effective due to the tools used and lack of optimization.



Theatrical Inspiration

Development of this project inspired me, however. Parallelizing the process of rendering the position and status of hundreds of individuals in a space stuck in the back of my mind, and it was amplified by seeing some interesting crowd dynamics at a theatre event.

As such, I decided to create a particle simulation of sorts. I've always found visual analysis far more helpful than simple numbers, so creating a particle simulation whose performance and capabilities specifically change based on optimization quality seemed perfect. The project would involve a simulation of particles responding to mouse-based gravitational forces and interacting with each other. Left-clicking attracts particles toward the cursor with distance-based force strength, while right-clicking repels them. Each particle's position needs to be calculated and rendered each frame. The end user would be able to dynamically adjust the level of parallelization, visually seeing the improvement of parallelization. The sequential approach may struggle with more than a few hundred particles at 60fps, while the parallel approach could tackle thousands.

Throughout this process, it would be tested on a variety of mediums, most notably the NVIDIA Jetson Developer Kit. While complex particle simulations may not be what this device was designed for, the project should serve to push the device to its limits and grant me hands-on experience with parallel development on edge devices like the Jetson.

Technical Specifications

This project will be developed and tested on my personal computer, the NVIDIA Jetson, and the Mill cluster if possible. Not only is my computer a smoother production environment than the Jetson, it will be interesting to see the performance comparison between the machines.

Development will be done in C++. This language is the best compromise of efficiency and simplicity, performing as well as C but requiring far less complex memory management. This is the recommended language to work on the Jetson. Development will be assisted by Claude Code, a command line interface tool that allows for the fluid integration of AI into a development environment. Tools like this are growing in popularity, so gaining hands-on experience with it helps to develop my skills as a software engineer. Additionally, it will speed up the development of complex sections of development where I lack experience, such as the development of the user interface.

There will be five levels of parallelization available to the end user. These are as follows:

- **Mode 1 - Sequential (Baseline):** Executes all physics calculations on a single CPU core with no parallelization, serving as the performance baseline for comparison. Uses standard C++ loops processing one particle at a time.
- **Mode 2 - Multithreaded (OpenMP):** Distributes particle computations across all available CPU cores using OpenMP. Achieves speedup by processing multiple particles simultaneously on different CPU cores with dynamic work scheduling.
- **Mode 3 - MPI (Message Passing Interface):** Splits the particle workload across multiple independent processes using MPI for distributed-memory parallelization with explicit inter-process communication. Each rank handles a partition of particles, achieving speedup despite communication overhead.
- **Mode 4 - GPU Simple (CUDA):** Offloads particle position updates and collision detection to the GPU using straightforward CUDA kernels with global memory access. Leverages GPU threads for parallelism, allowing for massive speedup.
- **Mode 5 - GPU Complex (CUDA Optimized):** Implements advanced GPU optimizations (TBD).

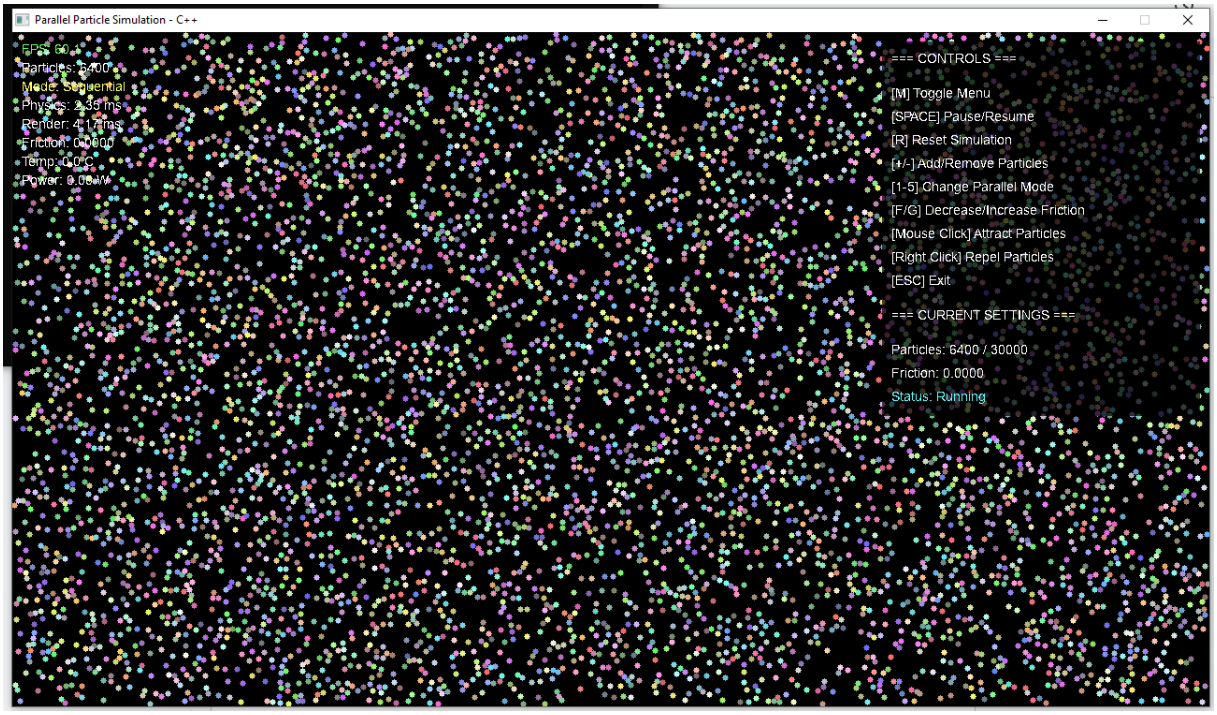
The end user will also be able to dynamically adjust the particle count, movement speed, energy loss, friction, and gravitational force strength.

In addition to parallelization, other optimization techniques will be used. This project will likely be memory bound due to the large amount of particles undergoing fairly simple computations. Since each particle doesn't interact with all others, we will likely be able to use a spatial grid to group particles with those nearby, vastly reducing the computational complexity. Further research will be done into specific memory optimizations for this domain area.

Performance metrics tracked for this project will include frame rate, power use, internal temperature, GFLOP utilization, and likely more metrics found across the way. The primary goal of this project is to make the end user feel the performance more than reporting metrics, tracking and reporting metrics will help to provide objective comparisons.

Conclusion

Currently, I have a rudimentary version of this system running on my personal computer. A screenshot of this is attached below.



Current Implementation

This implementation currently runs on my desktop computer. The particles respond to gravitational forces from mouse clicks and move fairly smoothly. With a sequential approach, my computer can currently handle 6400 particles at 60 fps. When multithreaded, this increases to 7450. When using MPI, this goes back down to 6500, which implies an error with my current implementation. The GPU modes have not yet been implemented.

Moving forward, there are still several steps needed to complete this project. I'll need to implement this on the Jetson, implement the GPU modes, optimize the other modes more effectively, refine the user interface, add more features, and polish this into a final deliverable.

There are several areas to continue exploring with both this and my honors project. There are always more optimizations to be found and features to be added. For instance, I could add wind, more dynamic colors to turn this project into a sort of heatmap, and even more complex movement that more simulates the original crowd simulations that inspired the project.

Additionally, I can continue to explore more edge computing heavy projects with the Jetson for my honors project. This project is not necessarily edge computing specific, so I could tackle an edge computing project using my experience gained throughout this. This could include a project utilizing machine vision like we may see in an industrial setting.

The expected outcome of this project is a program with which an end user can feel and see how varying levels of parallel optimization can improve performance of complex mathematical problems.