15-418/618, Spring 2024 Final Project Milestone Report

Name: Nicole Feng (nvfeng), Marian Qian (marianq)

1 Title

Parallel Fast Multipole Method (FMM) on Distributed Memory

2 URL

https://github.com/marianqian/parallel-fmm

3 Work Completed & Progress on Goals

So far, we have a sequential and OpenMP parallelized versions of the FMM. This FMM can be used in multiple configurations—for gravitational and Coulombic interactions, in 2D and 3D, and for uniform and non-uniform particle distributions. Additionally, we have begun parallelizing the FMM using CUDA instead of OpenMP, in hopes of choosing the better parallelization strategy to use in conjunction with MPI in the future.

Originally, we had planned to implement a sequential version of the FMM ourselves from scratch by rewriting existing python code for the algorithm in C++. However, it took us a while to understand exactly how FMM works in terms of the algorithmic and mathematical details. We had a hard time digesting the primary research paper we were going off of (A Massively Parallel Adaptive Fast Multipole Method on Heterogeneous Architectures), and had to read through many supplemental materials and watch videos in order to get a better understanding of how exactly FMMs work and how the formulas, algorithms and data structures are built to accomplish this.

When we finally started to write our sequential code, we soon realized that it would take a lot of time and effort to write modular and clean C++ code for implementing the FMM, when this was not even the main portion or intent of our project. We wanted to be able to focus on the real challenge, which was focusing on how to parallelize the FMM for distributed memory using MPI. Of course, parallelizing the FMM using OpenMP and CUDA would also be a significant portion of the project, but this parallelization is more straightforward than distributed memory parallelization due to the fairly obvious parallelizable portions of the algorithm. As such, we decided to pivot and use an existing C++ codebase that already implemented the FMM with OpenMP optimizations. We stripped it down to exclude any OpenMP directives and used that as our sequential code, and then put the OpenMP optimizations back in and used that as our OpenMP code.

Now, our efforts will mainly be focused on optimizing the FMM using CUDA instead of OpenMP, and then incorporating MPI to parallelize it for distributed memory. Then, we plan to perform a thorough analysis on the performance benefits of our various parallelization strategies.

Ideally, we would have completed our CUDA implementation by now, but there were a few things that held back progress. The first was the unexpected amount of time it took us to really understand how FMMs work. The second was the time we wasted in attempting to implement our own sequential FMM, only to later pivot to using an existing codebase so that we could focus our time on parallelization implementations and performance analysis. One thing that also held us back in making more progress for the milestone report is that the codebase we used relied on GSL, which is not installed on the GHC machines. It took us a while to figure out how to install GSL locally and get the codebase to compile on the GHC machines. Additionally, both of us were pretty involved in Carnival this year, and the second round of midterms for our classes was the week right before carnival, so it was difficult to find time to meet and make progress between the time of the initial proposal and the milestone report.

4 Adjusted Goals and Deliverables

Plan to Achieve:

- Correct implementation of FMM in C++, parallelized using OpenMP (already achieved via existing codebase).
- Correct implementation of FMM in C++, parallelized using CUDA to exploit data parallelism within a shared memory.
- Correct implementation of FMM in C++, parallelized using MPI to make the implementation adaptable and scalable across distributed machines/memory.
- We aim to have our parallel implementations at least exhibit some sort of speedup compared to the sequential implementation, perhaps almost linear speedup for up to 8 cores (7x speedup for 8 cores) — as this is what they achieved in the paper.
- We aim to have our MPI version perform better (speedup) on non-uniformly distributed particle data compared to the implementation that just has parallelism in the shared memory space (CUDA, OpenMP).

Hope to Achieve:

- We hope to achieve an extremely scalable solution that has good speedup on larger core counts (up to 128), maybe around 64x speedup (just extrapolating from the speedup numbers in the paper).

Poster Session Demo:

- At the poster session, we plan to present graphs of the speedup achieved by parallelizing using OpenMP vs CUDA, showing why we chose to use one or the other as our acceleration method for the FMM computations.
- Then, we plan to present graphs of speedup achieved by parallelizing using MPI and OpenMP/CUDA, showing how parallelizing across distributed memory can impact the performance of the FMM.
- We will also show speedup graphs for our 2 implementations on different problem sizes and particle distributions, so we can demonstrate the benefits of a heterogeneous architecture (distributed memory) in FMMs with uneven particle distributions.

5 Preliminary Results

We have no preliminary results as of yet.

6 Issues and Concerns

Our main concern is the difficulty of the implementation of MPI into the FMM. It involves partitioning the tree leaves across processes and then constructing a locally essential tree (LET) for each process in which it can calculate the N-body sum in parallel. The LET construction seems like it will be a slightly complex operation to implement correctly. Additionally, the reduce and scatter method detailed in the paper as well as the re-partitioning for load balancing are aspects that will require a significant amount of effort to get right.

7 Schedule

Due Date	**Task(s)**
Thur, 4/20	● Finish setting up CUDA/test run code for CUDA. Marian Qian
Fri, 4/19	● Finish implementing CUDA version of code. Marian Qian Nicole Feng
Sat, 4/20	● Record performance metrics on OpenMP and make speedup graphs (include problem size and particle distribution sensitivity studies). Nicole Feng ● Record performance metrics on CUDA and make speedup graphs (include problem size and particle distribution sensitivity studies). Nicole Feng ● Create a detailed outline for MPI integration, following the paper. Marian Qian
Mon, 4/22	● Finish one iteration of MPI implementation on GHC machines. Marian Qian Nicole Feng
Wed, 4/24	● Finish incorporating MPI parallelization into implementation. Marian Qian Nicole Feng ● Record performance metrics on MPI and (include problem size and particle distribution sensitivity studies). Marian Qian
Sat, 4/27	● If speedup is not ideal on GHC, do performance debugging to identify bottlenecks. Nicole Feng ● Collect and record performance metrics on PSC machines to see if the solution is scalable. If not, do performance debugging to identify bottlenecks. Marian Qian
Wed, 5/1	● Finish iterating on algorithm design. Marian Qian Nicole Feng
Sat, 5/4	● Finish collecting all finalized performance metrics for the report. Marian Qian Nicole Feng

Sun, 5/5	Final Report DUE
Mon, 5/6	Poster Session DUE