CS 420, Parallel Programming, Spring 2023
Project

March 26, 2023

The goal of this project is to implement an algorithm using MPI for distributed memory parallelism, and OpenMP for multithreading; and optimize the implementation.

The project will require considering different implementation options, performing an approximate analysis of the pros and cons of the different options, implementing them and measuring the achieved performance and speedup with strong/weak scaling. Few examples (of options to consider):

- Number of MPI processes per node and threads per process.
- Use of different work sharing options
- Use of different MPI communication functions.

Useful resource: https://courses.engr.illinois.edu/cs554/fa2015/notes/

1 Logistics

1.1 Timeline

- Project proposal (not more than 1-page) report due: 04/12/2023 (on Gradescope)
- Final project report submission due: 05/12/2023 (on Gradescope)

1.2 Team size

Team size of 1 or 2.

2 Report

The final report should include details on the implementation options that were considered and the performance improvements achieved. You can choose to have a section on future work. Your report can be 4 to 5 pages (based on the format/font used). Attached is a report outline for reference. Note that we do not expect that you should structure your report along these lines, it is **only** for reference.

3 Example projects

We list below several examples of algorithms one might choose; feel free to choose any another algorithm that interests you.

3.1 LU factorization

Given a system of linear equations $Ax = b$, we want to solve for x. An LU factorization finds $A = LU$ where L is unit lower triangular and U is upper triangular. The system then becomes $LUx = b$, where you can solve the lower triangular system $Ly = b$ with forward-substitution, then $Ux = y$ with back-substitution to obtain the solution x to the original system. Partial pivoting may be necessary to ensure numerical stability. Reference: https://en.wikipedia.org/wiki/LU_decomposition and those within

3.2 Cannon's Algorithm

Cannonś algorithm is a distributed algorithm for matrix multiplication for two-dimensional meshes. It is especially suitable for computers laid out in an N $\times$ N mesh. The main advantage of the algorithm is that its storage requirements remain constant and are independent of the number of processors. The Scalable Universal Matrix Multiplication Algorithm (SUMMA) is a more practical algorithm that requires less workspace and overcomes the need for a square 2D grid. It is used by the ScaLAPACK, PLAPACK, and Elemental libraries. https://en.wikipedia.org/wiki/Cannon%27s_algorithm

3.3 BFS

Implement a parallel Breadth-First Search to compute the distances to all the nodes in a graph starting from node zero. Use a level-synchronous algorithm that synchronises after computing distances to all nodes at depth d before visiting nodes at depth d+1. The graph can be generated using a random graph generator.

3.4 Cholesky factorization

A symmetric positive definite matrix A has a Cholesky factorization $A = LL^T$ where L is lower-triangular with positive diagonal. The linear system $Ax = b$ can then be solved using forward-substitution for $Ly = b$ followed by back-substitution for $L^T x = y$. Reference: https://en.wikipedia.org/wiki/Cholesky_decomposition and those within

3.5 QR factorization

A mxn matrix A has a QR factorization where Q is mxm orthogonal and R is nxn upper triangular. This can be used to solve linear and least squares problems (among others). Three common techniques for computing QR factorizations are Householder transformations, Givens transformations, and Gram-Schmidt orthogonalization. Reference: https://en.wikipedia.org/wiki/QR_decomposition and those within

3.6 Tridiagonal linear systems

You want to solve a linear system Ax = b where A is tridiagonal (i.e. it has nonzeros only on the diagonal, superdiagonal, and subdiagonal).These systems can be solved using LU or Cholesky, but there is little parallelism. This can be solved more efficiently using cyclic reduction. Reference: https://en.wikipedia.org/wiki/Cyclic_reduction

3.7 Finding Eigenvalues and eigenvectors

Given a square matrix A, find its eigenvalues and eigenvectors. There are many algorithms for doing this, as well as preprocessing such as similarity transformations, that can simplify the problem. Reference: https://en.wikipedia.org/wiki/Eigenvalue_algorithm provides an overview of different algorithms

3.8 FFT

The discrete Fourier transform has applications in many different fields. The Fast Fourier Transform is a particular way of computing the DFT efficiently using a divide-and-conquer algorithm. Two common approaches to implementing this algorithm in parallel are the binary exchange algorithm and the transpose algorithm. References: https://en.wikipedia.org/wiki/Cooley\OT1\textendashTukey_FFT_algorithm and https://en.wikipedia.org/wiki/Fast_Fourier_transform

---

4 Reference report

We list here only the section headings mentioned in the report.

Report by Hao Chen and Jared Espino

4.1 Abstract

This report optimizes and analyzes the performance of a parallel algorithm using MPI and OpenMP for shortest path problem. The parallel algorithm is developed based on Floyd–Warshall algorithm to solve an all-pairs shortest path problem in a weighted 2D graph without negative cycles. This report starts from a sequential Floyd–Warshall algorithm and parallelizes it through OpenMP and MPI, respectively. Then, we further optimize the communication structure in the MPI-based algorithm and try MPI+OpenMP strategy. The performances of different version of codes are compared with each other. Results show that our optimization about communication structure for MPI provides a dramatic improvement.

4.2 Introduction

4.3 Sequential Floyd algorithm

4.4 Parallel Floyd algorithm

4.4.1 OpenMP

4.4.2 MPI

4.4.3 Performance comparison

4.4.4 Further improvement