CS 434: Parallel and Distributed Computing

Percy Brown: 40382022

March 30, 2021

Lab Assignment 3

1 Matrix Multiplication using MPI

In this algorithm, I design use the very simplistic approach of row-block partitioning and column-block partitioning onto P processors. Two matrices A and B are assumed to be square matrices using P number of processors. A is partitioned into sqrt(P) rows and matrix B is partitioned into sqrt(P) columns.

1.1 Pseudocode overview

All matrices are allocated memory. Matrices A and B are initlaised with integers, depending on the matrix size specified. The size represents the number of rows and columns, denoting a square matrix. I used the MPI struct as derived data type to help the communication between processes and perform the matrix multiplication. Because of the difficulty in knowing what Process 0 carries at a particular point in time, I made Process 0 idle, purposely for performing the matrix computation and sending to matrix C, while the remaining processors perform the row and column block partitioning processes. Therefore the number of processors required when running the program is P + 1. The additional one serves as the main processor to perform the computation and carry the values into matrix C.

When P processors are passed during the running of the program, the number of worker processors become P - 1. The program checks if P - 1 processors is a square and evenly divisible by the matrix size before proceeding to other sections of the code. Te matrix data is sent to the worker tasks. Rows of matrix A and columns of matrix B as sent as needed. This is done by slicing the rows and columns based on the block size, which is dependent on the number of worker processors available. The worker processes calculate the starting and ending indices of the final result which will be later sent to the master processor. The worker processors receive the sliced rows and columns in a vector form, and then converts them into arrays to enable easy multiplication. After the multiplication, the arrays are linearized again and sent to the master processor, along with the starting and ending indices of the values to be positioned in matrix C, with the aid of our struct data type.

The master processor receives results from the worker processors alongside the start and end indices of the results. The master processor now gathers all the values, and their positions, and fills the matrix C as instructed. Therefore, if N is 16, and we want to use 4 processors, we need to add an additional one as the master while the 4 worker processes perform theur tasks of carrying sliced rows and columns and performing the multiplications. Therefore, anytime the program is being run and for instance N is 16 and we want P as 4 (which is a square and it's square root is evenly divisible by N), we use 5 as the number of processors on the command line during running. The program takes the spare processor and assigns it as the master processor, and works with the others as

the worker processor. This was mainly done to easily know what each processor is tasked to do. Hence the spare processor, serving as the master processor, only receives all computations and their positions, and fills the values in matrix C.

1.2 Comparative Table

N	P	Sequential	MPI
4	4	0.0000002s	0.000139s
8	4	0.0000006s	0.000169s
16	4	0.000049s	0.000333s