CS 5334 Spring 2021

Lab 4 Assignment – 50 points

Parallel Matrix Operations using MPI

Assigned Sunday, March 21, 2021

Due by 11:59pm on Sunday, April 4, 2021

For this lab assignment, you will use what you have learned about MPI programming to implement and test parallel MPI versions of functions for performing matrix operations on a distributed memory computer.

1. Please develop a function that implements matrix-vector multiplication using a 2D block decomposition of the matrix and vector. A driver program for the 2D case is provided for you. For your 2D function, assume that the data have already been distributed with the vector distributed among the processes in the rightmost column of the process grid. The result vector should be distributed in the same manner. The function is also passed a 2D cartesian communicator for the processes. You may assume a square matrix and that the dimension size is a multiple of the square root of the number of processes. (Hint: You will find it easier to program your function if you define sub-communicators for the rows and columns of the process grid).

Code:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Output:

Graphical user interface, text, application, chat or text message

Description automatically generated

1. A straightforward implementation of matrix-matrix multiplication using a 2D block decomposition of the matrices A, B, and C (assuming we are multiplying A times B to give C) requires storing all the blocks of a row of A and a column of B on all processors. Cannon’s matrix multiplication algorithm avoids this excessive storage overhead by shifting the blocks of A and B onto the appropriate processor exactly when they are needed to perform the block multiplication. We will discuss the details of Cannon’s algorithm in class. Please develop a function that implements Cannon’s algorithm and write a driver program to test it. Use a 2D Cartesian topology for the process grid. Your driver program should create the communicator for the 2D Cartesian topology and pass it to your function as an argument. You may assume square matrices and that the dimension size is a multiple of the square root of the number of processes.

Driver:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Skeleton/Code:

Text

Description automatically generated

Output:

Text

Description automatically generated

1. Makefile

Text

Description automatically generated

1. We will construct analytical models for the computation and communication times and for the space requirements for the 1D matrix-vector algorithm, the 2D matrix-vector algorithm, and the straightforward matrix-matrix multiplication algorithm in class. Please do a similar analysis for your Cannon’s matrix-matrix multiplication algorithm.

The time complexity of Cannon’s algorithm is subdivided in three parts. The first term of the time complexity is determined by the initial shift of data to the left on A, assuming the operation to calculate is C = A\*B. For this term we can observe two things a priori, first any communication has to be prepared hence that incurs on a cost of time which it is defined as ts, also there is a cost for transmitting an element (cost per word associated to communication) this cost is defined as tw. In relation to tw it is important to note that the number of “pieces” element that are moved is in the magnitude of n2 since we assume we shift/move around a square matrix of dimensions n\*n, having said that we only move the equivalent to n2/p elements because we subdivide the square matrix among “p” processors. Lastly, it can be observed that every element should has an upperbound of movement which is exactly – 1 since otherwise the shift can be neglected because the element would be transmitted to the same processor.

Hence, the first term of the time complexity of Cannon’s algorithm is equal to:

The second term follows a similar logic but the shifts do not occur on the columns of a row but instead the shift occurs on the rows of a column to shift the elements of B to their respective places. So, in general both the first and second terms of the time complexity are close to :

In relation to the last term of the time complexity it is possible to observe the time complexity is driven by the fact that we move data around and the fact we perform a computation. For the communication we move exactly all of the elements in the processors or n2/p elements at every step and the total number of steps is equal to n. A little difference is that here every element moves exactly 1 processor away instead of as many as processors away. So, the time complexity of communication is equal to O(n3/p) or more specifically:

The time complexity of the computation Apiece \* Bpiece is on the order of O(1). Hence the time complexity of the entire algorithm could be described as:

With the time complexity being on the order of

In terms of space if we assume we always have p = n that is there exists a processor for every element that will be created in the multiplication operation then the space complexity will be constant. If this is not the case then the processor will have to acquire a linear amount of memory in relation to the number of parts of the resulting array that every processor will be required to hold so the space complexity will be on the order of O(n).