Blake Cruz

Code Report for Project 2

For this project I was required to create two different implementations of a parallel for a given sequential solution. This sequential solution was responsible for calculating the digits of pi up to a certain point. My two parallel programs followed two different methods, one employed the point-to-point communication method using MPI SEND and MPI RECV to compile an answer. The other method I used was collective communication which involved using MPI REDUCE. My code also has detailed comments explaining more line by line.

**Code explanation:**

My explanation for my point-to-point code (proj2b.c) starts with setting up variables and starting the processing timer. Then initializing MPI and getting the rank and total number of the current processes. We then calculate the step size and use the equation from the sequential solution. The start of the point-to-point communication is when we check if the rank is equal to 0 then we are in the root process. In the root we will iterate and MPI RECV the total sum of the other processes. Else the processes will send their local sum. We then check again to see if we are in the root to finally calculate the digits of pi. After this we stop the timer and print the results. This works by having the MPI communicate between two processes.

My explanation for my collective communication code (proj2a.c) is very similar to the point-to-point version except instead of doing the receive and sending using the MPI we simply replace that with an MPI reduce command that takes in the total sum and will be responsible for collecting the local sums. This demonstrates communication between number of processes running at the same time.

**Analysis:**

The differences between the different implementations was seen in both the speed and accuracy of each of the programs. When running with 1000000000 NUMSTEPS and run with 4 processes, point to point finished in 678844547 nanoseconds, collective communication finished in 682493767 nanoseconds, and the sequential solution finished in 2758977877 nanoseconds. Based on these results the parallel programs were roughly 4 times as fast as the sequential solution when dealing with a lot of NUMSTEPS. Surprisingly the accuracy at this high of numsteps was better in the parallel code instead of the sequential solution. On average, point to point had a very slight edge over collective but fluctuations still could push collective to be faster.

When running with only 100000 NUMSTEPS, the sequential version was faster than the parallel version. The acuaracy of both were worse than the accuracy of the higher amount of numsteps. This leads me to believe that the more iterations using numsteps there are the closer we get to getting the correct digits of pi. The bigger the chunks we take out of a curve the more inaccurate it will be to measure thus more numsteps help find the value of pi with more precision.

**OUTPUT:**

**KEY FOR FILE NAMES:**

proj2s.c = sequential solution

proj2a.c = collective communication

proj2b.c = point-to-point

With a high value of numsteps (1000000000):

A computer screen shot of a program

Description automatically generated

With a smaller value of numsteps (100000):

A computer screen with white text

Description automatically generated