Assignment 1 FHPC 2021

Zainab Nazari, MHPC

December 2021

1 Section 1: MPI programming

1.1 Ring

This code is done in c. I singled out master rank=0 among all other processors to compute the time elapsed to go through the loop. I could use all the time of all the processors and then take the average, however to measure the average could also use a processor timing to avoid that I used a single processor, in this case processor with rank zero for all numbers of processors, starting from 2-36. I used the Cineca-marconi login node for this problem as the orfeo was busy that time. I use a bash to mutilple run the code for better result and take the average. I execute the code with the bash for 10 times, and took the average for timing, and plot the result with python.

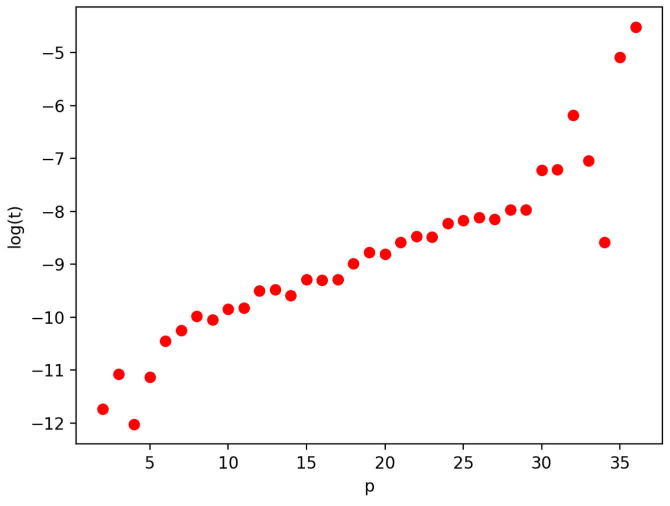

Figure 1: Ring

1.2 Matrix addition in parallel using different topology

This code is done in C++. It does not involve a scatter operation, because the partial matrices can be initialized on the individual processes themselves. The matrices are represented as three-dimensional arrays, which are pointers. The user enters how big the total matrices should be. Each process divides the third dimension by the number of processes (i.e. the *size*), so it knows how big the matrices should be that it needs to deal with. It then instantiates two matrices of that size and fills them with random double numbers between 0 and 1. Then, it adds the two matrices, resulting in a new matrix of the same size. These matrices are gathered into one big matrix (which is the desired result) by process 0.

I avoided the scatter operation, because I thought that each process has enough information to start computing, so there is no need that one process prepares data for all the others (such as initializing the big matrix with random numbers). However, the gather operation needs a matrix to gather into, and while I could have filled that matrix with any numbers, it was convenient to just fill it with random numbers the same way that the other processes do. This means that the advantage of pre-filling the matrix and not scattering cannot be played out here.

Another issue is that I got pointer errors when trying to delete the matrices after use (causing a memory leak), but I think that is not a big issue, because they are up for re-allocation on the next run of the program.

The code can be run with different sizes of the matrix. When I tried different topologies of the nodes, I got errors that I was unable to resolve. In general, the data is spatially distributed, but there is no interaction between neighbouring nodes, so I don't expect any differences in performance with different topologies.

2 Section 2: Measure MPI point to point performance

In this section, some of the topology of the network was considered both in gpu and thin node and the name of the files indicated which one. we plotted time vs. the message bytes and bandwidth vs. the message bytes and studied the behaviour of the curves. It is evident the the elapsed time for small messages are about the same but for larger messages the elapsed time start to increase but bandwidth start to be constant. This is the ideal behaviour we expect but for some of the data files we have somewhat different. I used python for plotting and also for fitting the curve of time vs. message size as a linear function with two parameter: latency and inverse of bandwidth. For some of the datasets I got negative estimated latency which doesn't make sense. Because of that reason, I used the minimum of time, which is supposed to be the first number if the series, as the latency; then, I used these two parameters to estimated the "t[usec] computed".

3 Section 3: Compare performance observed against performance model for Jacobi solver

In this section, as it was asked in the question I have compiled and run the Jacobi solver and in the file section3 there are two `pbs` job scripts named `sec3-gpu.sh` for the gpu and `sec3-thin.sh` for the cpu thin node. We submit these jobs in orfeo with `qsub sec3-gpu.sh` and `qsub sec3-thin.sh` these files will create two folders and two files. The two folders `results-gpu` and `results-thin` and two csv files.`results-gpu.csv` and `results-thin.csv` which contain the average time, performance, and the size of array in column 1, 2, and 3 of each files.