Erdem Ege Maraşlı
21602156

CS426 - Project 1 - Report

1.Implementation and Design Choices

1.1.Minimum Calculation Program

Serial implementation is very easy and straight forward. Min variable compared with the current number on the file at each iteration. If the min variable is bigger than the current number, the min variable updated as the current number. At the end of the file, the min variable holds the result which is the minimum number.

Implementation of min-mpi-v1.c is really easy and straight forward too. First of all, master processes iterate the whole file to find out how many numbers exist. Numbers divided nearly equally(some processes can get one more number since the total number can not get divided perfectly to the number of processes) to the number of processes and sent them to responsible processes by MPI_Send call. All processes other than master receive numbers by MPI_Recv call. All processes calculate their local minimum value serially. All processes other than master send local minimum values to the master and master calculates minimum value between local minimums serially and prints the result.

Implementation of min-mpi-v2.c is again really easy and straight forward. Master processes again iterate the whole file to find out how many numbers exist. Then MPI_Bcast call called as the master is the root to broadcast the data to all processes. All processes calculates the minimum value serially and prints them.

1.2.Grayscale Program

Serial implementation is pretty straight forward. Line by line dot products of the RGB values with the [0.21, 0.71, 0.07] calculated and written in the output file.

For the implementation of grayscale-mpi-v1.c first of all, the master process sends the nearly equally distributed pixel data to the other processes with the MPI_Send call. Processes receive pixel data with the MPI_Recv call and calculate the dot products of RGB values with the [0.21, 0.71, 0.07] and send the calculated dot products to the master process with the MPI_Send call. The master process receives the calculated dot products with the MPI_Recv call and writes the dot products to the output file.

For the implementation of grayscale-mpi-v2.c first of all, the master process organizes the data with the calculated grid sizes and sends the equally distributed grids to the other processes with the MPI_Send call. Processes receive pixel data with the MPI_Recv call and calculate the dot products of RGB values with the [0.21, 0.71, 0.07] and send the calculated dot products to the master process with the MPI_Send call. The master process receives the calculated dot products with the MPI_Recv call and reorganizes them to write the dot products with row-major order again to the output file.

For the implementation of grayscale-mpi-v3.c first of all, the master process organizes the data with the calculated grid sizes and use MPI_Scatter call to distribute the data. I chose to use MPI_Scatter operation to improve the performance since data size is equal for all processes and MPI_Scatter uses the most efficient distribution algorithm with respect to the current hardware which will become more efficient than using MPI_Send call for each process. Processes calculate the dot products of RGB values with the [0.21, 0.71, 0.07] and use MPI_Gather call to collect calculated dot products in the master process. Again MPI_Gather is much more efficient then to use MPI_Send and MPI_Recv for each process since it uses the most efficient algorithm according to current hardware. The master process reorganizes the data to write the dot products with row-major order to the output file.

2. Benchmark Results

CPU: Intel Core i5-8259U @ 2.30GHz
RAM: 16 GB 2133 MHz

All of the data points are taken as the minimum execution time of the 20 executions.

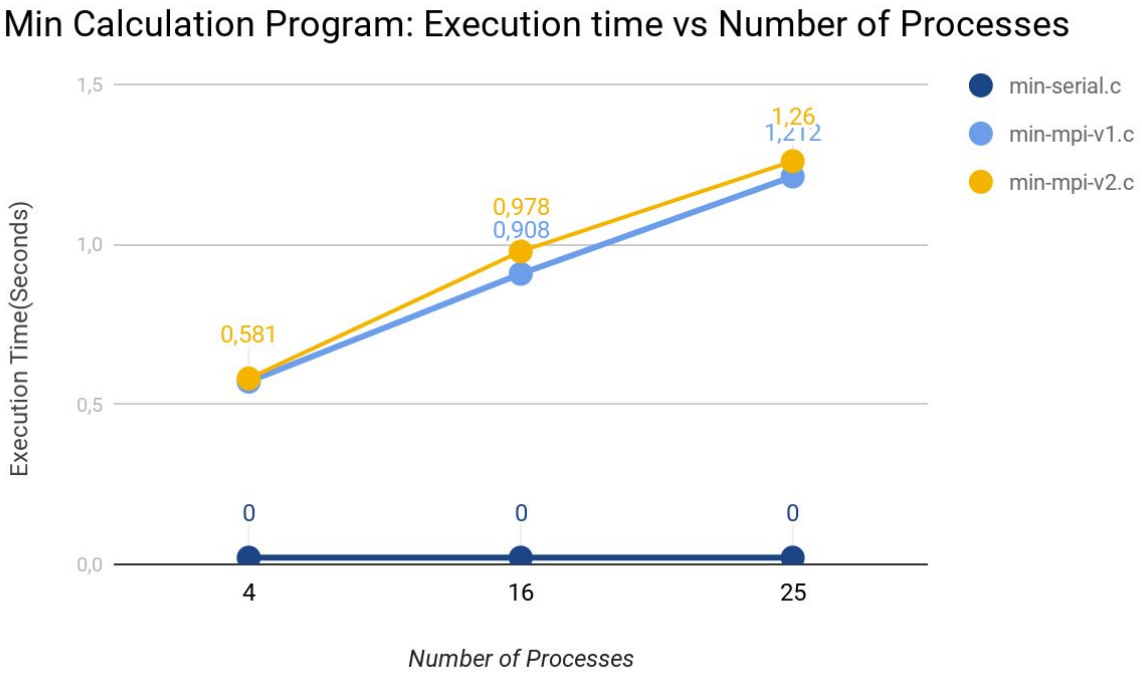

Figure 1: Min Calculation Program: Execution time vs Number of Processes.

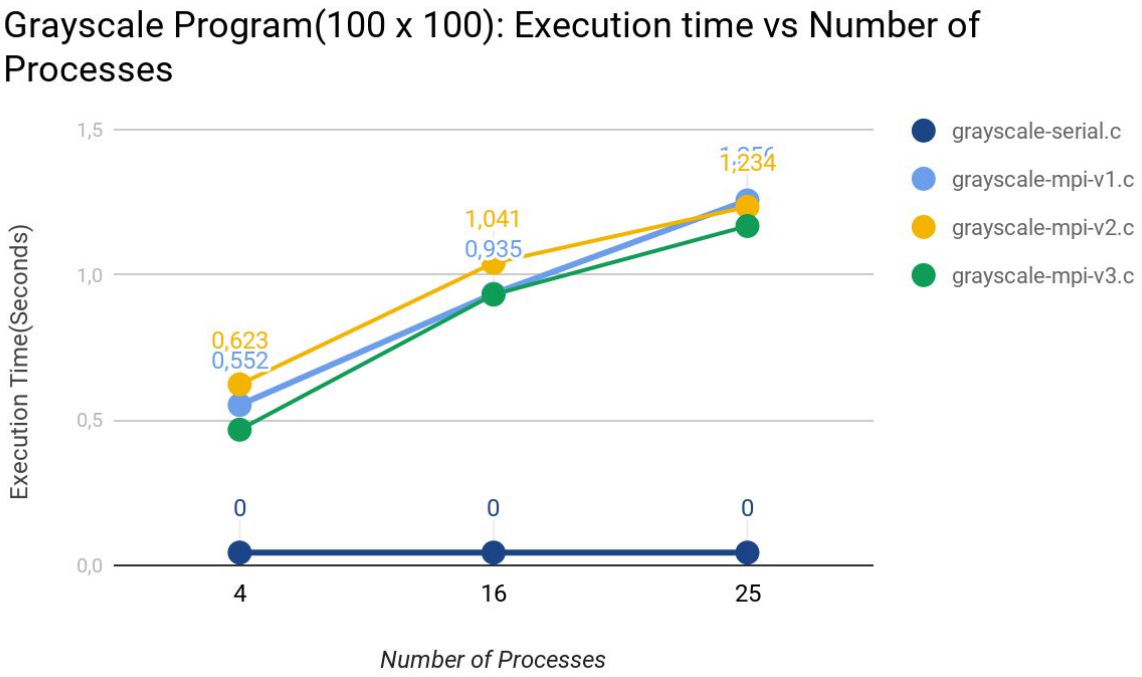

Figure 2: Grayscale Program(100 x 100): Execution time vs Number of Processes.

3. Comments and Observations

Serial executions out-performs the mpi executions since all tests are conducted in one single machine which already uses a share memory system so serial execution does not waste time to send and receive messages.

For the minimum calculation program, we can see that performance results are nearly the same. Even they use different communication implementations, v1 uses most of its time to divide, distribute and collect the data while v2 uses most of its time at calculations since calculations duplicated for each process. v1 and v2 eventually use similar times for different events.

For the grayscale program, we can see that v1 performs slightly better than v2 they both make the nearly same amount of send and receive calls, however, v1 directly sends the data in row-major order while v2 use time to reorganize the data to send as grids. V3 out-performs both v1 and v2 because it uses Scatter and Gather implementations to send and receive data. Scatter and Gather operations are implemented efficiently for the given hardware in the MPI library so that it will perform better than Send and Receive calls for each process.