CS426 - Parallel Computing - Project № 2

Erdem Ege Maraşlı - 21602156, Bilkent University 01/04/2020

1. Implementation and Design Choices

This program uses the data parallelism strategy since the program applies the same functions across the elements of a dataset. This is similar to Single instruction, multiple data(SIMD) execution.

In this project I use Scatter, Gather, Send and Recv MPI communication features. I used 2 different types of approach, in the first approach I use Scatter and Gather to divide and combine data. I used this approach if the number of the experiment is perfectly divisible by the number of processes. Using Scatter and Gather is way more efficient than using Send and Recv, however, if the number of the experiment is not perfectly divisible by the number of processes I use the second approach which is using Send and Recv communication features.

On MPI platforms every process has its own memory and their address spaces are not shared with other processes. They are like different OS-level processes. Processes use the message-passing paradigm to communicate with each other. They need to connect in the same network to use the message-passing paradigm. So we can use MPI applications in distributed systems, after all, MPI applications already used in distributed systems. Passing pointers are not possible in the MPI paradigm, however, in this project passed functions are initialized for each process separately so it would work in a truly distributed platform.

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 execution.

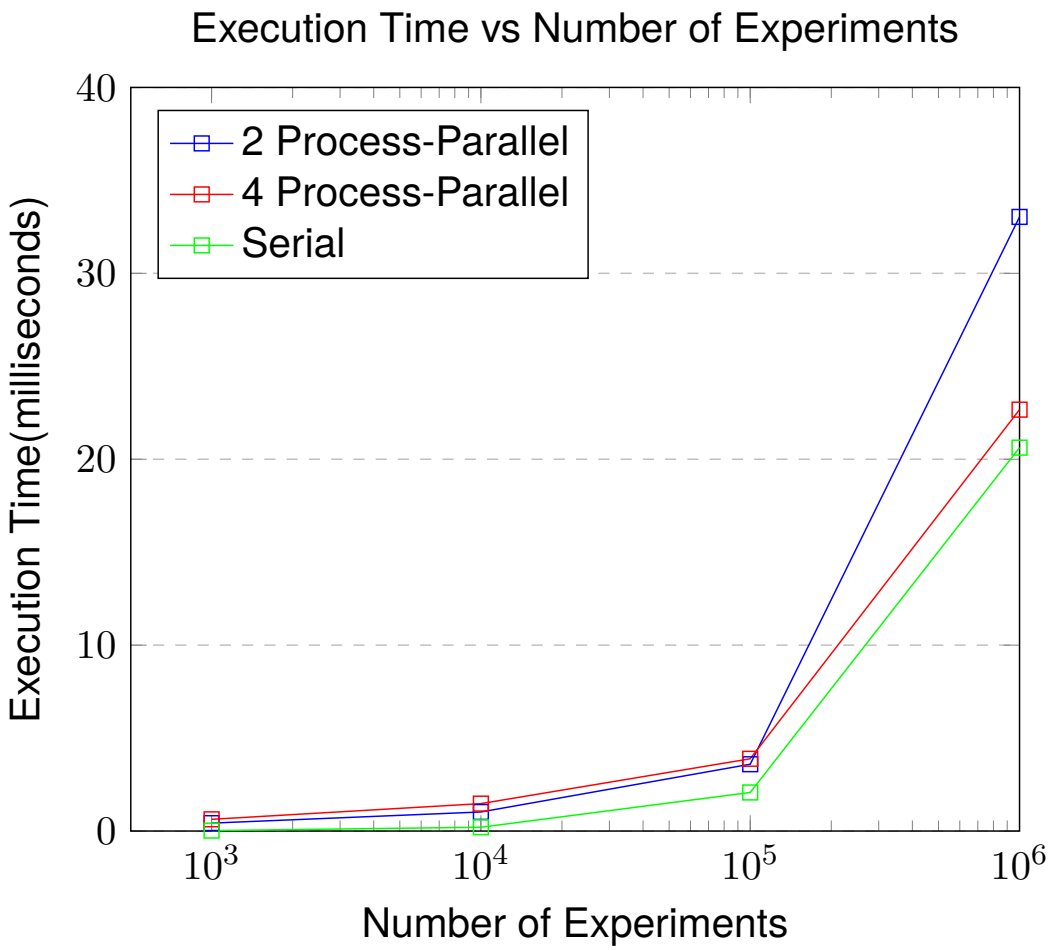

Figure 1: Execution Time vs Number of Experiments.

3. Discussion of Experiments

Serial executions generally 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. Only when the number of experiments is 1000000, serial execution and 4 process parallel execution time is really close since data size is really big overhead for dividing and combining the data gets really small and available 4 processor can be used to perform computations efficiently.

When the number of the experiment is different than 1000000, we can see that the execution time of 2 processes and 4 processes parallel executions are nearly the same. 2 process parallel execution use less time to divide and combine data since chunks are bigger, however, computation takes longer since only 2 available processors are used for big chunks of data. 4 process parallel execution use more time to divide and combine data, however, computation takes less since 4 available processors are used for smaller chunks of data. When the number of the experiment is 1000000, we can see that 4 process parallel execution out-performs 2 process parallel execution since computation time gets much more important for this case because the needed computation number increased excessively.