CS508 - Assignment 2 Report

Aaditya Arora B17071

Note:

1. Instruction of executing a program is written inside each program as comment.
2. File structure P1/p1.c P2/p2.c P3/p3.c
3. All observations are taken with time shell command.

P1. SELECTION SORT O(n^2)

The performance gain or loss completely depends on the size of the input array that is given to the program. The main reason of using MPI is we can parallelise heavy independent computational work and thus can increase performance but If passing of data (message passing) from one process to another becomes the bottleneck then the performance of parallel program in MPI would be less than that of its sequential implementation.

For Array Size : 1e5

Parallel Program: 2.75 s (using 4 processor)
Sequential Program: TLE (not able to compute)

We can see that from above comparison using parallelisation we can do computation which is not possible to do in sequential form.

For Array Size : 1e4

Parallel Program: 0.11s (using 4 processor)
Sequential Program: 0.11s

The computation time is almost the same as time saved in parallelisation is used while message passing.

For Array Size : 1e3

Parallel Program: 0.06s (using 4 processor)
Sequential Program: 0.00s

The computation time is more in the case of parallel programs as compared to sequential.

Let's say **p** be number of processors and **n** be the size of array then the time complexity of the parallel program is given by :
$O((n/p)^2 + nlog(p))$

Where $(n/p)^2$ is time taken in each process for doing selection sort and $nlog(p)$ is for merging arrays from different processes into one. So, based on which factor dominates decides the time complexity of the program.

P2.

In this program, We are doing parallelisation in 2 segments :

1. We Scatter (MPI_Scatter) the image data into all the processes from the root process and then we have computed a 4 x 8 vector (Histogram) in each process.
2. After histograms are computed in each process we are merging (element wise sum of each 4 x 8 vector) them parallely.

Let

W : width of Image
H : height of Image
P : number of processes.

then

Time Complexity in parallel program : O(W*H + 32log(p))
Time Complexity in sequential program : O(W*H)

Although, the time complexity looks the same but a huge difference lies in the constant terms associated with each variable.

Let lambda be latency and beta be bandwidth then the time complexity of scatter (MPI_Scatter) fun is defined as :

$$\sum_{i=1}^{\log p} (\lambda + 4n/(2^i\beta)) = \lambda \log p + 4n(p-1)/(\beta p)$$

As, we can see that these constant factors also decide whether to split a process and also how much splitting is good.

For Image Size : 540 x 360

Parallel Program: 0.08s (using 4 processor)
Sequential Program: 0.00s

For Image Size : 1024 x 1024

Parallel Program: 0.14s (using 4 processor)
Sequential Program: 0.05s

As we can see from above observation -- performance of sequential performance is better than parallel program as message passing is the bottleneck here.

Also for a given image if we increase the number of processors the performance of the program will decrease because of the above mentioned result.

P3.

This program is somewhat similar to the above program -- The only difference lies is that of reading two images as compared to one. The parallelization in this programs are :

1. Scattering (MPI_Scatter) the data of two images into all the processes and computing two 4 x 8 vectors in each process.
2. After histograms are computed in each process we are merging (element wise sum of each 4 x 8 vector) them parallely.

Then, computing the hellinger distance (H.D.) in the root process.
The complexity of the program would be the same as of **P2**.
For Two Images of Size : 540 x 360

Parallel Program: 0.09s (using 4 processor)
Sequential Program: 0.02s

For Image Size : 1024 x 1024

Parallel Program: 0.22s (using 4 processor)
Sequential Program: 0.10s

As we can see from above observation -- performance of sequential performance is better than parallel program as message passing is the bottleneck here.

Conclusion :

1. Selecting the number of processors for a given job highly depends on the size of input.
2. M.P. can be a bottleneck in deciding the performance of the program.
3. Currently, We are C file i/o fopen -- instead if we use parallel file i/o for distributed file system. For Example : If we Big Data problem then loading whole data into the root process and then scattering would be a problem.

[REFERENCE]

[1] https://stackoverflow.com/questions/10625643/mpi-communication-complexity