Assignment Question 2

Project Report

Name: VIpul Krishnan M.D. (ID: 28104641)
5/14/2018

Introduction

Our naval fleet patrols an area with 1000 location. Considering the ships as processes, we need to write an MPI program to simulate the patrolling to maximize the number of launched strikes. For a vessel to launch strike at a location, it should satisfy the condition that the location should be occupied by at least one odd numbered vessel and 2 even numbered vessels. We can choose the number of ships.

We consider MPI processes as vessels and a 1000 locations as an array with 1000 elements. The basic idea is to generate a random number in each process and consider it as the position of that vessel, i.e. that process. After fixing the positions, we need to check for the possible locations for launching a strike, i.e. the positions satisfying the above condition. We count the number of those positions and that is our output.

We need to finding the rate of launching strikes by repeatedly performing the above steps for 1 minute. As a general rule, more the number of processes, more will be the probability of strikes, and more will be the total number of strikes. However, due to communication delay between processes, we expect the performance to be reduced beyond a certain number of vessels. (processes)

Program structure and Provisions for Efficiency

As mentioned, there are n number of processes where 'n' is the required number of vessels. In the first step each process calls a random number generation function for generating a number between 0 and 999. The time multiplied by the rank of the process is given as the feed for the random number generator to ensure that the generated number is completely random.

Once they obtain the number, makes an array with size = number of processes, and fills the element with index = rank of the process, with their obtained random number, and all the other elements are set to 0.

The next step is the communication step which is accomplished with the help for MPI_Reduce function. Each process calls MPI_Reduce with MPI_SUM as the reduction function and the generated array as the input. The length to be mentioned in the MPI_Reduce function should be the number of processes.

The process 0 obtains the reduces array, which contains the vector sum of the arrays from each processes, which in fact is a superposition of arrays from all processes, ie the one filled in each index with the random number generated in corresponding processor.

In the last step, the process 0 executes an algorithm to check the number of positions with at least one odd process and 2 even processes. This is the final result.

The above mentioned steps are one execution of the task. We need to do continuously for 1 minute to get the rate of launching strike. This is done by putting all the above steps in a time controlled while loop which stops running after 1 minute.

The program should be highly efficient as we have used only the optimized in-built methods for communication. The MPI_Reduce method is highly optimized. Also we have used only the basic MPI_SUM reduction function and not used any complex user defined functions. The program is overall a very simple, small program with very less number of lines of code.

The Inter-process Communication Scheme

As we know, the communication time is the bottleneck for the performance of most of the distributed systems. To make the process efficient, we try to reduce the number of communications as much possible. In our scheme, everything is completed by a single MPI_Reduce operation. All the processes calls this function and the reduced result is obtained at the root processes which is the 0^{th} process. As we don't use any methods like MPI_Send, MPI_Receive, this should be increasing our performance.

Performance Metrics

The most important performance metric is the number of strikes launched per minute itself.

The second possible metric is the number of the outer loops executed. This gives an idea about the effect of the process communication delays. When the communication delay increases, the while loop execution comes slower

Observations

Number of processes	Strike rate (avg)	Number of loop executions
60	297	31765
70	763	32018
80	1918	29708
90	1967	32889
100	3452	26803
110	15603	20533
120	10458	14251
130	9588	12884
140	13929	10340
160	19999	7000
180	15717	3920

Please note that some small sleep delays are introduced in the program to keep the computer not crashing.

Conclusion

The strike rate increases with the the number of processors. We couldn't reach at the max point because of the hardware restrictions. However we can say that, the strike rate is enhanced by increase in number of processes. If we take a probabilistic approach we can say that strike rate will increase with cube of number of processors. So it matches with our result.

Also we can note the effect of the message passing delays of the MPI, from the number of execution loops. It decreases, but in a smaller rate compared to the rate of increase of strikes. So we can say that our program is efficient enough such that the message passing delay is not affecting the strike rate.