EECE 5640
HOMEWORK 4

Shreyash Shrikant Jadhav

Q1.

With lesser number of darts thrown, the accuracy of pi was less.

Number of darts: 1,000,000

Number of Processes	Time(ms)
1	55.38
2	52.3
4	46.29
6	44.27
8	38.37
10	33.5

Number of darts: 10,000,000

Number of Processes	Time(ms)
1	444
2	638.88
4	600.54
6	560.32
8	500.25
10	421.90

Number of darts thrown: 2,000,000,000

Number of Processes	Time(ms)
1	125812
2	123776
4	121305
6	119328
8	118974
10	117498

Q2.

a.)

N = 100,000

	Node = 1	Node = 2	Node = 4
Classes	Time	Time	Time
5	4.184	2.5	4.08
10	4.9	3	2.2
20	7.4	3.9	3.02
40	11.49	8.08	3.9
50	12.91	10.4	7.08

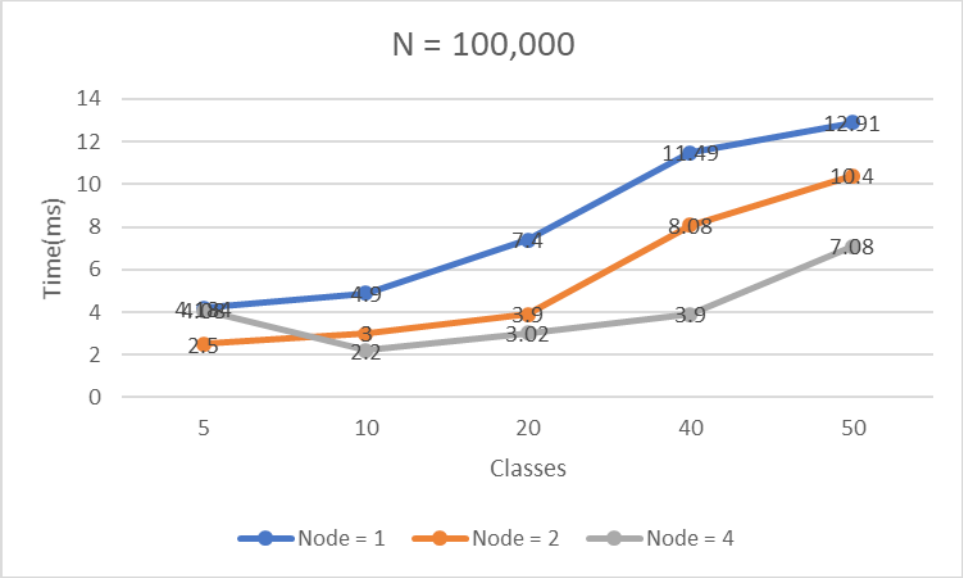

Figure 1

In **Part A**, I have divided N into number of MPI processes. So I have divided the numbers (N) by the size which we get from MPI_Comm_size() and the classes are defined by the users. For example if N = 1000, and the MPI size is 4 and number of classes are 10, each process will get 250 numbers and then each processes will count the numbers which belongs to each of the classes and then send the data to the master node(0). At end I am printing the values in the each class/bin.

In **Part B**, depending on the number of processes running there are classes created. If the number of the processes are 4 then the number of classes created is 4. If the range is 1000 then there are 4 classes with the range 0-250, 250-500, 500-750, 750-1000. For example if N = 1000 and number of classes are 4 i.e. there are 4 processes running. So each processes will get all the 1000 integers and each process will bin the numbers that belong to that range.

In Part A, figure 1 and 2 shows the performance of method followed for binning. We can see that as the number of classes increases the runtime also increases. But as we distribute the work among nodes we can significant decrease in the runtime.

N = 1,000,000

	Node = 1	Node = 2	Node = 4
Classes	Time	Time	Time
5	33.8847	29.45	34.8
10	42.898	38.54	39.8
20	60.4881	61.34	64.8336
40	94.5	91.97	102.986
50	112.43	111.45	95.114

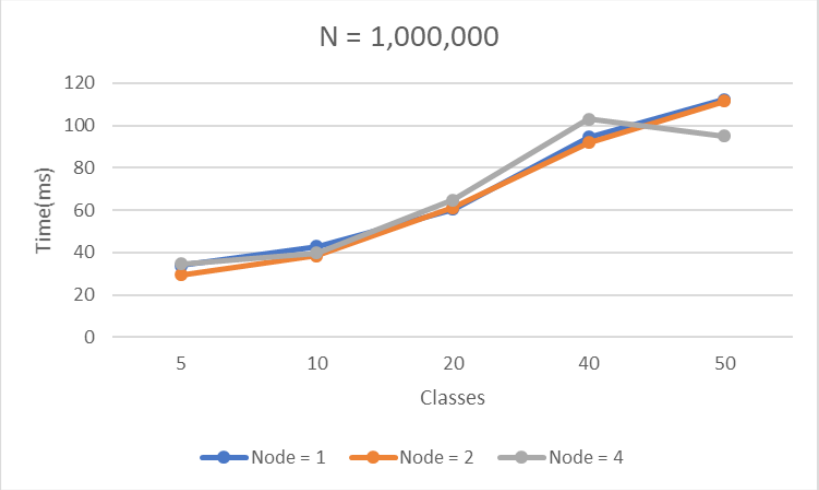

Figure 2

b.)

N = 100,000

Node Combination	Classes(Processes)	Time
1,5	5	47.2
2,5	10	65.8
4,5	20	216.99
4,10	40	298.2
2,25	50	381.6

N= 1,000,000

Node Combination	Classes(Processes)	Time(ms)
1,5	5	145.02
2,5	10	179.8
4,5	20	889.5
4,10	40	1025
2,25	50	1102.04

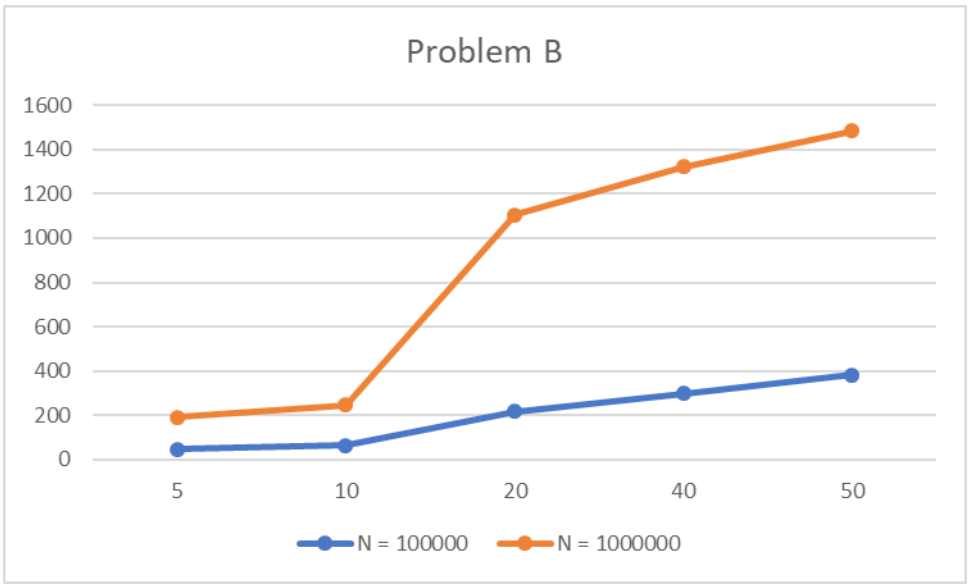

Figure 3

In Part B, from Figure 3 we can see that as the N increases the runtime of the program also increases with increase in the number of classes.

c.)

N	Classes	Problem A time(ms)	Problem B time(ms)
100,000	5	4.08	47.2
	10	2.2	65.8
	20	3.02	216.99
	40	3.9	298.2
	50	7.08	381.6
1,000,000	5	34.8	145.02
	10	39.8	179.8
	20	64.8336	889.5
	40	102.986	1025
	50	95.114	1102.04

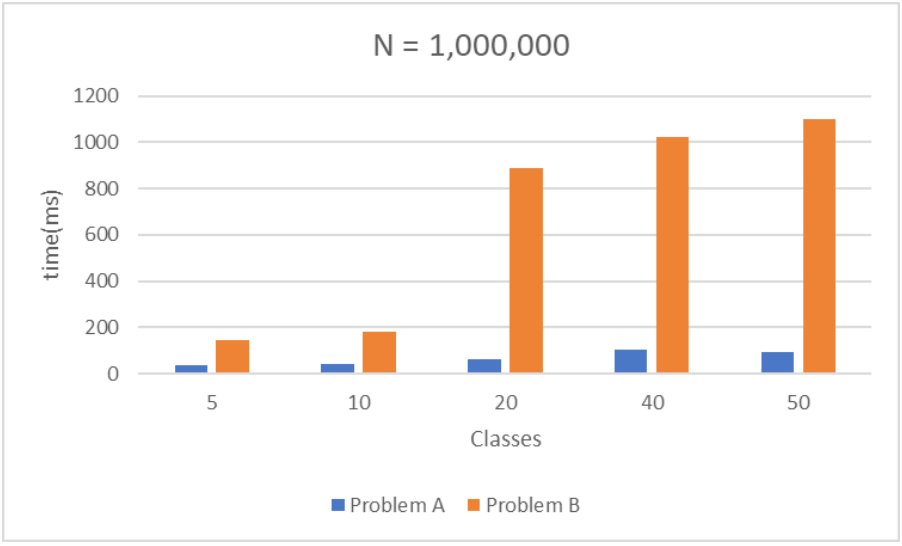

Figure 4

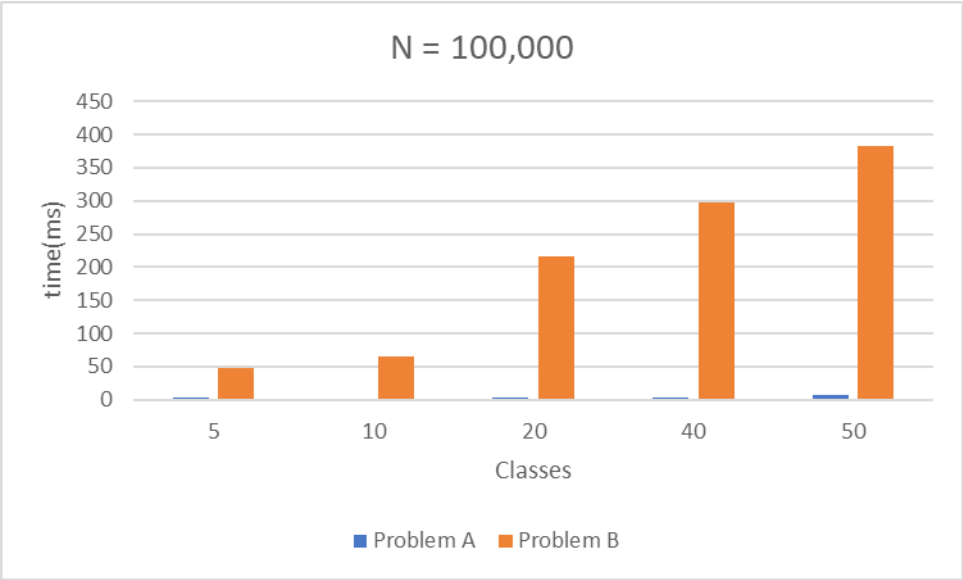

Figure 5

In the figure 4, we can see the difference in the runtime with same number of classes. We can see that in the problem B the run time is more compared to the problem A because in problem A only a fraction of integers are given to a single process in oppose to all the N integers given to single process and then the process will bin the integers according to the assigned class to the process. So in case as N can go to infinity but the classes cannot. The classes are limited and can be increased in the limited order but N can be increased more than classes. In figure 5 as the run time for the Problem A is very low it can be properly be seen on the

3.)
For performance anlaysis of a MPI program we perform profiling of the program. The profiler will show whether the code is compute bound, memory bound or communication bound.
Perfromance Analysis tools for MPI applications:

1. Intel Vtune
2. ARM MAP
3. TAU Commander

1. Intel Vtune [1]

- Intel Vtune gives the application performance snapshot for profiling the program.
- It consits of Intel Trace Analyzer and Collector and Intel MPI Tuner for anlaysis of MPI code for the cluster for analysing the communication hotspots, synchronization, bottlenecks and the load balancing.
- It also consists of Intel Vtune Amplifier for analysis of OpenMP programs
- It also has Intel Advisor for more of the core level performance i.e. single thread performance on particular hardware.
- It works best for single node[3].
- Intel Vtune supports the Intel processor but it does not support ARM processors[5].
- It provides detailed analysis for the MPI program giving program execution time for each of the MPI program.

2. ARM Map[2]

- Arm Map shows the time taken by each line of the code. For running Map compile the program with the debug flag -g.
- With running a MPI program with the Map you can specify the number of processes, number of nodes, processes per node and the implemenation.
- Implementation is the OpenMPI, MPICH or others. It is normally initialized at start of the program as it is required for the start of the program.
- It has a simple GUI used for start of the program and to start of the program. Using this you can also profile only a part of program and not the entire program.
- Vtune does not support ARM processors while Map supports Intel Xeon processor used mostly in the compute nodes in the cluster.

Q4.

Changes that occurred after the paper

- **Fault tolerance** in the MPI programs have increased since the paper was published. There are different approaches taken to decrease the faults that may occur in the processes of an MPI programs. In this paper[6] they have discussed about using Algorithm based Fault Tolerance(ABFT) to solve the faults in the processes. They had ring communication as an ABFT to reduce the faults occurring during the process creation.
- **Shared memory** was an issue when the paper was written, but now due to faster bus speeds and increased cache sizes, there is less dependency on the shared memory to achieve a speed up from an application.
- There was a **communication overhead** which is significantly reduced due to faster interconnects.
- There are some MPI functions that take longer arguments and hence take larger memory in each process, this still is same but due larger cache sizes this problem is now mitigated but not yet eliminated.

Exascale Computing

- There are many issues in the exascale computing that were prevalent in the MPI at the time that the paper was written. These some challenges are still haunting us in the development of the exascale computing.
- The main issue discussed in the [7] is the load balancing that had to be done when distributing the workload on millions of the cores on the clusters.
- While distributing the workload there is always network overhead that occurs, there is also limited off chip memory bandwidth for application requiring larger amount of data.
- Also due to the locks in some processes we are limiting the parallelism so still there is serialization in the code that prevents us to go to the exascale computing.

References:

1.https://software.intel.com/en-us/vtune-help-mpi-code-analysis
2. https://developer.arm.com/tools-and-software/server-and-hpc/debug-and-profile/arm-forge/arm-map
3. https://www.osc.edu/sites/default/files/staff_files/skhuvis/performance.pdf
4. http://taucommander.paratools.com/tau-commander-tutorial-example-1-mpi-only/
5. https://software.intel.com/en-us/comment/1601498
6.https://www.researchgate.net/publication/220953589 Building a Fault Tolerant MPI Application A Ring Communication Example
7. https://dl.acm.org/doi/abs/10.5555/3199700.3199729