Q1:

In question 1, I calculated Pi on both 2 and 4 nodes. With a range of darts from 10,000 to 20,000,000. Instead of drawing a real circle and square then throw darts at it, I was using probability in unit areas. Say we have a square with Area=4, and a circle with Area=3.14. If the darts are perfectly evenly distributed, the probability of darts laying inside the circle is 3.14/4. From here, I will generate random number N in range 0-400, the probability of N<314 is also 3.14/4. Which serves for the same purpose. Each of the node will generate totalNumOfDarts/NumOfNode darts. And they will send the result to the node with rank 0. Rank 0 will compute the result. Below is the output I produced. We can conclude that more darts means more accurate results on average, as well as less execution time on average. However, I believe the best implementation is 2 Nodes and 20,000,000 darts. This combination gives an accurate result with less hardware utilization. The best result I got was Pi=3.140245. This should be a good result by any standards. The execution of using 2 nodes only takes 1.5x time than using 4 nodes when dart=20,000,000, not 2x. Which is also good.

文本

描述已自动生成文本

描述已自动生成

文本

描述已自动生成文本

描述已自动生成

文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成文本

描述已自动生成

Q2:

c) In this problem, the results are listed below. I used 7 different combinations for both part A and B:

|  |  |
| --- | --- |
| Number of Nodes | Number of Class |
| 1 | 2 |
| 1 | 4 |
| 2 | 2 |
| 2 | 4 |
| 4 | 2 |
| 4 | 4 |
| 4 | 8 |

My test cases include the cases where Node> class, Node=class, and Node < class. Each node will handle more than one classes in the case which Node<Class.

It is obvious that A has better performance than B because it only produces 1/#$class input for each node. Therefore, no input value in being wasted.

Furthermore, because A is not wasting any input value, the numbers in the histogram should add up to N, which is the input value. (Except when we have odd number of classes, then we might lose a few due to rounding errors). Numbers in the histogram in B doesn’t necessarily adds up to N, we have better randomness in B.

a)

图形用户界面, 文本

描述已自动生成图形用户界面, 文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

b)

图形用户界面, 文本, 网站

描述已自动生成

图形用户界面, 文本, 网站

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

Q3:

The performance evaluation tools I selected are **EZTrace** and **PERDAC**. In this section, I will compare and contrast the capabilities of these two frameworks. Both tools are trace generation frameworks that is used to analyze the performance of application to optimize the overall performance.

EZTrace is a more powerful tool comparing to PREDAC. It has better compatibility and more functions. Not only it can analyze MPI programs, but also on pthread and Open MP. EZTrace will automatically create tracefiles during execution using FxT library. Then it will compute statistics to analyze the overall characteristics of the application. The advantage of doing the analyzing after execution is to offer a global view of the application when analyzing events and avoids being restricted by reactivity issues. Finally, the biggest advantage of EZTrace is that it offers a script language that allows the user to define functions without having to modify their source code.

On the other hand, PERDAC is a little bit weak on capability, but it has its advantages. Just like EZTrace, it will create trace files during execution and do analyzation after. But in PERDAC, user must generate trace files with all variables by themselves. But once this operation is done, then PERDAC can trace more detailed information than EZTrace, including total execution time of an MPI function on a process, the number of occurrences of each MPI function and optionally some hardware performance counters such as cache hits or cache misses. It can also choose the best MPI library for the user if it will speed up the execution.

Based on the statistics in these two papers, both frameworks are good enough. But I will say that EZTrace is better from my standpoint, simply because of the user friendliness and compactibility.

**Works Cited**

**PERDAC:**

Gabriel, Edgar & Sheng, Feng & Keller, Rainer & Resch, Michael. (2006). A Framework for Comparative Performance Analysis of MPI Applications.. 478-484.

**EZTrace:**

F. Trahay, F. Rue, M. Faverge, Y. Ishikawa, R. Namyst and J. Dongarra, "EZTrace: A Generic Framework for Performance Analysis," 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2011, pp. 618-619, doi: 10.1109/CCGrid.2011.83.

**Extra:**

In the “Millions-of-Cores” paper, the author first talked about time and memory consumption. He stated that if the memory consumption per process grows linearly, then the system is not scalable. However, I believe the fast-growing SSD storage technology should have positive impact. Back in 2010, SSD storage devices rarely go over 128GB in size, and HDD was the dominate storage device. But now, SSDs comes with huge. storage spaces.

Another positive impact on the industry is the communication technology. The interconnect is an essential part in high performance computing. With the help of fiber optic or even silicon photonics, the transfer time between nodes will be decreased.

Finally, the author talked about the scalability of specific MPI functions a lot. Unfortunately, I’m not an expert in MPI, so I cannot comment on this part of the paper.