Daniel Valdes Artiles

High Performance computing

Data 7B

Introduction:

In this exercise, we will address the problem of calculating π using numerical integration techniques. Specifically, we will focus on the method of Riemann sums, a fundamental technique in the field of integral calculus. This approach will allow us to approximate the value of π by computing the area of a quarter circle.

Our approach is based on the fact that the area of a unit radius quarter circle is precisely π/4. To achieve this, we will utilize the function (f(x) = sqrt(1 - x^2)) to describe the quarter circle over the interval (x = 0, ..., 1). We will then employ the Riemann sum method to divide the quarter circle's area into small rectangles, whose areas will be summed to approximate the total area.

Finally, we will multiply this approximation by 4 to obtain an estimation of the value of π.

We will explore various implementations of this method, starting with a sequential solution in Python and then advancing to parallel solutions using libraries such as multiprocessing and mpi4py.

Solutions:

The sequential solution to approximating π using numerical integration involves dividing the area of a quarter circle into small rectangles and summing their areas. The key steps include defining the function that describes the quarter circle, determining the number of rectangles (N) for the approximation, calculating the width of each rectangle (Δ𝑥), iterating over the rectangles to compute the area of each, and finally summing up these areas to obtain an approximation of π.

Key steps:

We define the quarter\_circle\_area function to calculate the area of the quarter circle using the Riemann sum method.

We calculate the width of each rectangle (Δ𝑥) based on the total number of rectangles (N).

We iterate over each rectangle, compute the value of 𝑓(𝑥𝑖), and sum up the areas to obtain the total area.

Finally, we multiply the total area by 4 to approximate π.

Parallel solution:

The parallel solution utilizes multiprocessing to distribute the computation of areas among multiple processes, thus improving efficiency by utilizing multiple CPU cores simultaneously. The key parts of the code involve defining a function to calculate the area of each rectangle in parallel, creating a pool of worker processes, mapping the computation function to the pool, and gathering the results for the final approximation of π.

Key Steps:

We define the quarter\_circle\_area\_parallel function to calculate the area of each rectangle using parallel processing.

We create a pool of worker processes using the Pool class from the multiprocessing module.

We generate a list of arguments for the quarter\_circle\_area\_parallel function, specifying the index of each rectangle and the total number of rectangles.

We use the map method of the process pool to distribute the computation of areas among the worker processes.

After obtaining the areas from all processes, we sum them up to calculate the total area.

Finally, we multiply the total area by 4 to approximate π.

Parallel Solution with mpi4py

In the solution using MPI for distributed computing, we follow the same general approach as the sequential solution to approximate π using numerical integration. However, instead of computing all rectangle areas in a single process, we distribute the calculation among multiple processes using the MPI (Message Passing Interface) standard.

Key steps:

Definition of the quarter\_circle\_area\_parallel function:

We define a function called quarter\_circle\_area\_parallel that calculates the area of a set of rectangles in parallel. Each process receives a range of rectangle indices and the total number of rectangles as arguments.

Calculation of area locally:

Each process calculates the area of the rectangles corresponding to its range locally, using the quarter\_circle\_area\_parallel function.

Reduction of local areas:

The locally calculated areas are reduced using MPI reduction operations to obtain the total area of the quarter circle. The root process (with rank 0) receives the final result.

Calculation of the approximation of π:

The root process multiplies the total area by 4 to obtain an approximation of π.

Conclusions:

The scalability of the program is evident as the execution time generally increases with the augmentation of the number of rectangles, denoted as N. This correlation suggests a direct relationship between the computational workload and execution time across all processes.

The parallel efficiency of the program is subject to scrutiny. While increasing the number of processes may potentially reduce execution time, the extent of this improvement is contingent on mitigating communication overhead. Despite the possibility of achieving enhanced parallel efficiency, there exists a threshold beyond which further proliferation of processes may lead to diminishing returns or even performance degradation.

Determining the optimal number of processes for a given problem size is paramount.