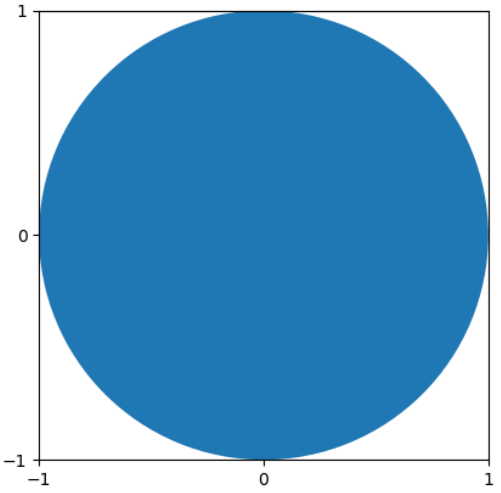
Parallel Computing

2022/2023 1st semester

Exercise Sheet #3 2022-10-26

Systematic circle area scan to obtain the numerical value of π

Create a python program that creates a grid of $n = m \times m$ equidistant points with $x, y \in [-1, 1]$, determines the fraction of points falling inside the inlying circle with radius 1, and calculates an approximated value for π. Do not use any trigonometric functions, as this would presuppose the knowledge of π's value. Use only basic arithmetic operations.
Compare the precision and efficiency (runtime needed) of this approximation with the one obtained through the Monte Carlo algorithm.
The actual value of n should be taken as a parameter from the command line.

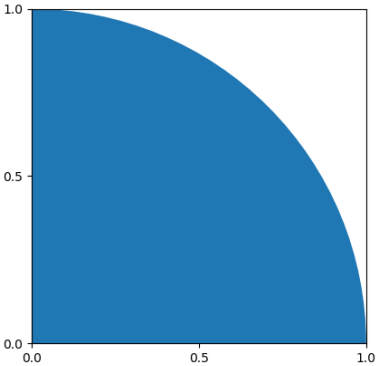


1. Use a Python program without any parallelization ("single core") to simulate $n = 10^8$ points, and compare the results with the ones obtained by Monte Carlo simulation for the same n. Measure the real time the program took to complete.

2. Create a parallel code in order to run the same exercise in p parallel processes, $n = 10^8$ points as well. To divide the workload over the p threads, subdivide the y-axis into p equally sized slices.

   Run it on your PC with the number of cores (i.e. threads) present, at least $p = 4$. The program should determine the actual number of processes involved through the `MPI.COMM_WORLD.Get_size()` method.

3. Due to the symmetry of the system, it is sufficient to consider only the right upper quarter of the circle respectively the square, i.e. the area with $x, y \in [0, 1]$. Optimize the program in this way, and use it to simulate $n = 10^8$ points. What can be observed in terms of time to run the code and precision of the result?



4. Whenever we scan the line for a given value x_i on the x-axis, beginning at $y = 0$ and increasing y_j in steps, as soon as we get a point (x_i, y_j) outside the circle we know that all following points with $y > y_j$ for the same x_i will fall outside the circle, too. So we can break the loop at this point and save time.

   Optimize the program in this way, and use it to simulate $n = 10^8$ points. What can be observed in terms of time to run the code and precision of the result? Verify the time used for each of the threads.

5. Discuss the consequences of this optimization with respect to parallel processing. How can we adapt the distribution of the whole task over the parallel threads in order to minimize the total time of computation?

   Develop an optimized version of the code in order to have each thread using approximately the same time of computation.

Submit the codes of your programs (1 até 5) at `https://trixi.coimbra.lip.pt/cap`
until the beginning of the next lecture.