PVL part III - MPI parallelization

By

Parsa Besharat

A handout submitted as part of the requirements

for the lecture, Introduction of High-Performance Computing, of MSc Mathematics of Data and Resources Sciences

at the Technische Universität Bergakademie Freiberg

December, 2024

Supervisor: Prof. Oliver Rheinbach

Abstract

This study investigates the numerical computation of π by approximating the integral

$$\int_0^1 \frac{1}{1+x^2}\,dx = \tan^{-1}(1) - \tan^{-1}(0) = \frac{\pi}{4}$$

The trapezoidal rule is utilized for numerical integration, dividing the interval into small subintervals to approximate the area under the curve. To enhance computational efficiency and reduce execution time, the problem is parallelized using the Message Passing Interface (MPI). Each process computes a portion of the integral, and the results are aggregated to obtain the final value of π.

A strong scaling test evaluates the performance improvement as the number of MPI processes increases. The study highlights the trade-offs between precision, performance, and scalability in high-performance computing applications. The results demonstrate that accurate computation of π\piπ to 7 decimal places is achievable with substantial reductions in computation time through parallel processing.

Contents

Introduction

The task is to compute an approximation of π\piπ using numerical integration of the integral

$$\int_0^1 \frac{1}{1+\ x^2}\ dx = \ \tan^{-1}(1) - \ \tan^{-1}(0) = \frac{\pi}{4}$$

The trapezoidal rule is employed to approximate the integral, and parallel computation with MPI is used to improve efficiency. Additionally, the solution includes a strong scaling test to evaluate the performance with increasing numbers of processes.

The trapezoidal rule is a numerical integration method that approximates the area under a curve by dividing it into trapezoids. For the integral formula we have, the trapezoidal rule approximates the value as:

$$\int_a^b f(x)dx \ \approx \ \Delta x \left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right]$$

Where $\Delta x = \frac{b-a}{n}$

For our integral, the trapezoidal rule becomes:

$$\int_0^1 \frac{1}{1+\ x^2}\ dx \ \approx \ \Delta x \left[\frac{1}{1+\ 0^2} + \frac{1}{1+\ 1^2} + 2\sum_{i=1}^{n-1} f(\frac{1}{1+\ x_i^2})\right]$$

The solution involves parallelizing this computation using MPI, where each process computes a portion of the integral over a subset of the range. The results are combined to achieve the final approximation of π. This approach ensures both high accuracy (7 decimal places) and reduced execution time through parallel processing.

Cluster Setup

Based on the TUBAF modules, these modules were required. These modules were loaded using the *module add* command to ensure compatibility and optimal performance during the execution of the codes:

1. `gcc/11.4.0`
2. `openmpi/gcc/11.4.0/5.0.3`
3. *`gdb/python/gcc/11.4.0/3.11.7/0.14.1`*

So for adding them:

1. `module add gcc/11.4.0`
2. `module add openmpi/gcc/11.4.0/5.0.3`
3. `module add gdb/python/gcc/11.4.0/3.11.7/0.14.1`

Program Description

This program computes an approximation of π\piπ using numerical integration with the trapezoidal rule for out Integral. It leverages MPI for parallelization, dividing the computation across multiple processes to achieve faster execution times. Each process computes a portion of the integral, and the results are combined using *MPI_Reduce* function. The program is designed for high precision (7 decimal places) and evaluates the performance across various numbers of MPI ranks to demonstrate strong scaling.

Running the code

In the C++ code, we have to implement in two ways:

a. *`mpicxx -o runner main.cpp.`* this will compile the C++ code and save the compiled file into a executable file called *runner*.

b. *`mpirun -np 2 ./runner.`* This will run the executable file.

Output

Number of Ranks	Computed π	Execution Time (seconds)
1	3.141593	0.262564
2	3.141593	0.132734
4	3.141593	0.114006
8	3.141593	0.068759
16	3.141593	0.036361

Output Analysis

The program computes π to a precision of 7 decimal places (3.141593) across all runs, confirming its accuracy. Execution times decrease significantly as the number of MPI processes increases, showcasing the benefits of parallelization. For instance, with 1 process, the execution time is 0.262 seconds, reducing to 0.133 seconds with 2 processes, and further down to 0.036 seconds with 16 processes. The results highlight efficient scaling, though diminishing returns are evident as the process count increases, due to communication overhead between MPI ranks.

Conclusion

This program demonstrates the effective use of parallel computing with MPI for numerical integration. It accurately approximates π while achieving substantial reductions in computation time through parallelization. The strong scaling analysis underscores the efficiency of MPI in distributing workloads but also highlights the trade-offs as communication overhead grows with additional ranks. This implementation serves as a practical example of combining mathematical methods with high-performance computing.