PVL part II - MPI basic communication

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 task involves implementing a basic MPI (Message Passing Interface) program to measure the time taken for message exchanges between two processes. It is part of a lab assignment for the course Introduction to High Performance Computing and Optimization. The goal is to use MPI's communication primitives to send and receive a message between two processes (rank 0 and rank 1) in a loop, and compute the average time per message exchange.

The program demonstrates core concepts of parallel computing, including message passing, process synchronization, and performance measurement. By using MPI, the exercise familiarizes students with distributed memory programming models, which are foundational in high-performance computing.

Contents

Introduction

In the context of high-performance computing, efficient communication between processes is critical. This document analyzes a task requiring the implementation of a basic MPI program to measure the message passing time between two processes. The program exchanges a single double value between rank 0 and rank 1, repeating the operation 50 times in a loop. The task also involves measuring the total communication time and calculating the average time per message exchange. This analysis includes implementations in both C++ and Python, highlighting the differences in performance and ease of use.

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. *`intel-oneapi-mpi/2021.13.0`*: Intel MPI library for distributed memory programming.
2. *`python/gcc/11.4.0/3.11.7`*: Python 3.11.7 compiled with GCC 11.4.0 for scripting.
3. *`openmpi/gcc/11.4.0/5.0.3`*: OpenMPI library for parallel computing.

Program Description

The program implements a basic MPI-based message passing system where a single double value is exchanged between two processes (rank 0 and rank 1) in a loop repeated 50 times. Rank 0 sends the message to rank 1, which sends it back, completing one exchange. The program measures the total time taken for all exchanges using *`MPI_Wtime()`* and calculates the average time per message in microseconds. Implementations in C++ and Python adhere to the same logic, with the Python version leveraging the mpi4py library and a numpy array for MPI compatibility, while the C++ version uses native MPI functions for optimized performance.

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.
c. Output would be

```
Total time: 5.4765e-05 seconds

Average time per message: 0.54765 microseconds
```

Output Analysis

The C++ program outputs a total time of 5.4765e-05 seconds for 50 message exchanges, with an average time per message of 0.54765 microseconds. This indicates the efficiency of the implementation, as C++ is a compiled language that minimizes overhead in MPI communication. [2] The low average message time suggests that the message passing between the two ranks is highly optimized, with minimal latency.[1]

This is typical of C++ in high-performance computing, where the language's low-level operations and minimal runtime overhead enable fast communication even in simple message passing tasks.

Conclusion

The C++ implementation of the MPI program successfully demonstrates efficient message passing between two ranks, achieving an average message time of 0.54765 microseconds. The results highlight the superior performance of C++ in high-performance computing tasks [1][2], especially when low-latency communication is crucial. This experiment reinforces the value of using compiled languages like C++ for MPI-based applications [3], where performance is a primary concern, and offers insight into how to optimize parallel programs for faster execution in distributed computing environments.[1]

References

[1]. MPI Forum: MPI Standard Documentation - https://www.mpi-forum.org/

[2]. OpenMPI: https://www.open-mpi.org/

[3]. Intel MPI Library:
https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html