Analyzing the Performance of OpenMPI's Collective Operations

By
Muhammad Mubashar Shahzad
[SM3600012]

Report submitted in partial fulfillment of
the requirements for the project of

High Performance Computing

DEPARTMENT OF MATHEMATICS AND GEOSCIENCES

UNIVERSITY OF TRIESTE

2024

TABLE OF CONTENTS

ABSTRACT

This report investigates the performance of OpenMPI collective operations, specifically focusing on broadcast and reduce functions. Utilizing the OSU Micro-Benchmark tool, the study evaluates various algorithms, including ignore, chain, pipeline, binomoal, rabenseifner, ignore and binary tree, under different parameters such as message size, number of processes, and number of cores. Experiments were conducted on the Epyc partition of the ORFEO high-performance computing cluster. Results indicate that the binary tree and binomial algorithm consistently outperforms others for large message sizes due to its efficient distribution method. Conversely, the chain algorithm shows poor scalability with increasing message size. Mapping policies generally have minimal impact except for the pipeline algorithm, which benefits from a core mapping strategy. These findings are supported by a linear model analysis, providing insights into optimizing parallel applications using MPI.

Chapter 1

INTRODUCTION

In the realm of high-performance computing (HPC), optimizing the efficiency and performance of parallel processing libraries is crucial for enhancing computational speed and resource utilization. This study focuses on evaluating the performance of collective operations in the OpenMPI library, particularly the broadcast and scatter functions. Collective operations, which involve data exchange among multiple processes, are fundamental in parallel computing environments, and their efficiency directly impacts the overall performance of parallel applications.

The primary objective of this report is to analyze the performance of different OpenMPI algorithms for broadcast and reduce operations under various conditions, including different data sizes, number of processes, and process mapping strategies. The experiments were conducted using the OSU Micro-Benchmarks on the EPYC partition of the ORFEO cluster, which features AMD EPYC 7H12 processors with a total of 256 cores. The intricate architecture of these processors, with multiple Core Complex Dies (CCDs) and Core Complexes (CCXs), is designed to deliver high computational capabilities, making it an ideal platform for performance evaluation.

This study aims to provide insights into the performance characteristics of different algorithms implemented in OpenMPI, helping to identify the most efficient strategies for various scenarios. By understanding the impact of algorithm choice, data size, and process map on collective operations, this research contributes to optimizing the use of OpenMPI in HPC environments, ultimately leading to improved application performance.

1.1 Background and Objectives

OpenMPI is a widely used library for parallel computing, providing a range of algorithms for collective operations to optimize performance based on various parameters. This study focuses on two key collective operations: broadcast, where a single process sends data to all other processes, and reduce, where data is distributed from one process to all others. These operations are critical for synchronizing and distributing data in parallel applications.

The objectives of this study are:

- To evaluate the default OpenMPI implementations of broadcast and reduce operations.
- To compare the performance of different algorithms for these operations, including binary tree, pipeline, chain, binomial, and rabenseifner.
- To analyze the impact of different message sizes and the number of processes on the performance of these operations.
- To investigate the effect of process mapping strategies on the latency of collective operations.

By conducting a comprehensive analysis of these factors, the study aims to identify optimal configurations and strategies for using OpenMPI in various HPC scenarios.

Chapter 2

Methodology

The experiments were conducted on the EPYC partition of the ORFEO cluster, utilizing two nodes, each equipped with AMD EPYC 7H12 processors. The OSU Micro-Benchmarks were employed to evaluate the performance of broadcast and reduce operations under different configurations. The key aspects of the experimental setup include:

2.0.1 OSU Micro-Benchmark

OMB provides benchmarks for a range of MPI blocking collective operations, including Allgather, Barrier, Bcast, Gather, Reduce, Reduce-Scatter, Scatter, and vector collectives. These benchmarks function as follows: when users execute the osu Bcast benchmark with N processes, the benchmark evaluates the minimum, maximum, and average latency of the MPI Bcast collective operation across these N processes for various message sizes, repeated over a substantial number of iterations. For our experiments, we will focus on the average latency for each message size. In this report, we will specifically examine two operations: broadcast and reduce.

2.0.2 Algorithms:

The research evaluates multiple OpenMPI algorithms for broadcast and reduce operations, incorporating different strategies to enhance performance. These algorithms include:

Binary Tree:

Uses a hierarchical structure where each node communicates with its child nodes, creating an efficient distribution pattern for scalability.

Ignore:

A baseline algorithm that skips certain optimizations, often utilized for comparison purposes.

Pipeline:

Divides data into segments that are passed through a sequence of processes, which can improve performance for larger messages.

Chain:

Connects processes in a sequential manner, where each process sends data to the next one, suitable for specific network configurations and message sizes.

Binomial:

Follows a binomial tree approach, balancing communication and computation steps, which is typically effective for medium-sized clusters.

Linear:

Involves straightforward, direct communication between neighboring processes, ensuring simplicity and predictability.

Rabenseifner:

Integrates recursive doubling with reduce-scatter techniques, minimizing the number of messages and synchronization steps, particularly advantageous in large-scale systems.

This analysis aims to determine which algorithms deliver the best performance across various conditions and message sizes, helping to choose the most effective method for broadcast and reduce operations in OpenMPI setups.

2.0.3 Mapping

The impact of process mapping strategies, such as mapping by core, was analyzed to identify the optimal configuration for minimizing latency. By examining how different mapping approaches affect communication performance, the study aims to determine the most effective strategy for reducing latency and enhancing overall efficiency in MPI operations.

2.0.4 Test Parameters

The experiments varied the number of cores per node from 1 to 128, totaling 256 cores across the system. Additionally, message sizes ranged from 1 byte to 2^{18} bytes. This comprehensive approach allowed for a detailed analysis of performance across different core counts and message sizes, providing insights into the scalability and efficiency of the various MPI algorithms and process mapping strategies.

2.1 Performance Model

The performance model in this case involves predicting latency based on various features of the data. We use linear regression, a standard method in statistical learning, to model the relationship between the input variables and the response variable. To create a general equation that represents the latency for all operations based on the plots, we need to consider the linear trend in each model and combine these observations. To generalize, we can consider an average of the coefficients from these equations shown in plots. This will give us a representative linear model that approximates the behavior of all these models. The general form of the linear regression equation is given by:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p + \epsilon \tag{2.1.1}$$

where:

- y is the dependent variable (latency).
- $x_1, x_2, \ldots, x_p$ are the independent variables $\beta_0, \beta_1, \beta_2, \ldots, \beta_p$ are the coefficients.
- ϵ is the error term.

2.2 Model Implementation

2.2.1 General Model for Latency

Based on the analysis of the plots for different models, the general equation for latency (L) as a function of the number of processes (P) is given by:

$$L = 1.2175 + 0.0094 \times P \tag{2.2.2}$$

This equation represents an average trend of latency across all operations considered. The naive model for latency (L) using rabenseifner algorithm as a function of the number of processes (P) is given by:

$$L = 1.58 + 0.0085 \times P \tag{2.2.3}$$

The naive model for latency (L) using ignore algorithm as a function of the number of processes (P) is given by:

$$L = 0.18 + 0.0118 \times P \tag{2.2.4}$$

The naive model for latency (L) using the linear algorithm as a function of the number of processes (P) is given by:

$$L = 2.98 + 0.0059 \times P \tag{2.2.5}$$

The naive model for latency (L) using the binomial algorithm as a function of the number of processes (P) is given by:

$$L = 0.13 + 0.0114 \times P \tag{2.2.6}$$

Chapter 3

RESULTS AND DISCUSSION

The analysis of MPI collective operations, specifically focusing on broadcast and reduce operations, provided key insights into performance characteristics influenced by various algorithms, message sizes, number of processes, and mapping policies.

3.1 Broadcast Operation

The pipeline algorithm demonstrates the best performance in the graph due to its consistent, low-latency profile and stable behavior across a wide range of core counts. The ignore and binary tree algorithms show similar trends with a steady increase in latency as the number of cores increases, indicating stable and predictable performance.The chain algorithm generally follows the same trend but has a significant spike in latency around 190-200 cores, which could indicate a temporary inefficiency or an outlier.

3.2 Reduce Operation

For the reduce operation, The linear algorithm has a notable peak but performs well with a higher number of cores, showing efficient reduction in latency after the peak. The ignore algorithm provides consistent and relatively low latency across the range of cores.The binomial algorithm maintains stable and low latency without significant peaks.The rabenseifner algorithm has the highest latency peak and generally higher latency compared to the others, particularly with an intermediate number of cores.

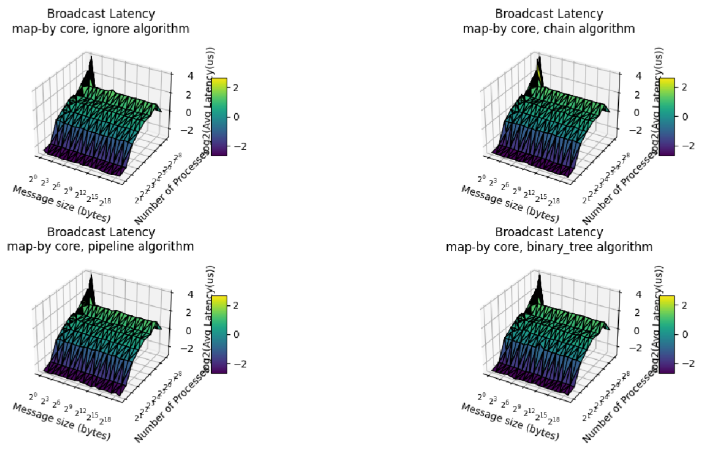

Figure 3-1: 3D Heatmap of broadcast operation for 4 different algorithms

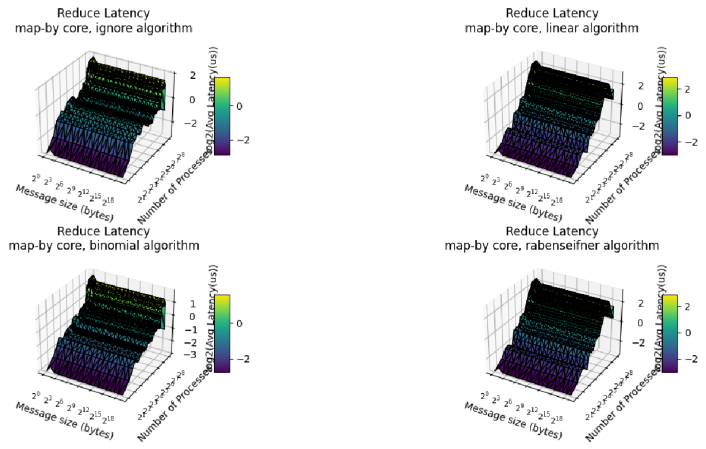

Figure 3-2: 3D Heatmap of reduced operation for 4 different algorithms

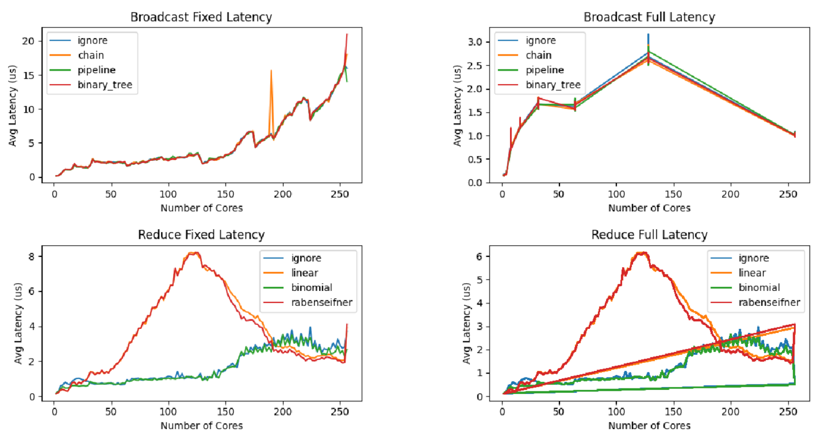

Figure 3-3: Plot of latency of Broadcast and reduce operations for different algorithms

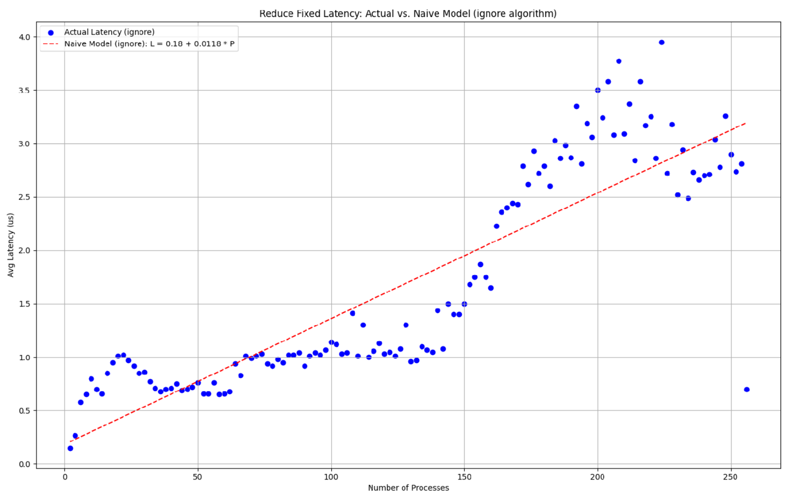

Figure 3-4: Line map of the ignore algorithm for Nave Model performance with a fixed size

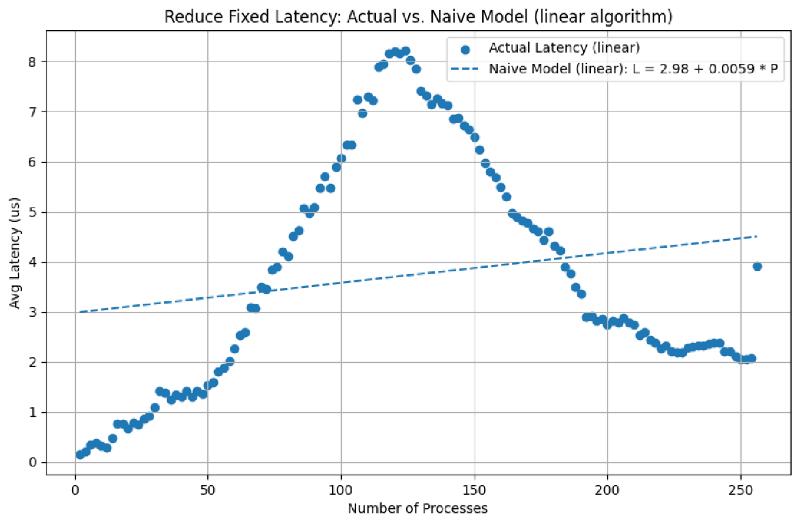

Figure 3-5: Line map of the linear algorithm for Nave Model performance with a fixed size

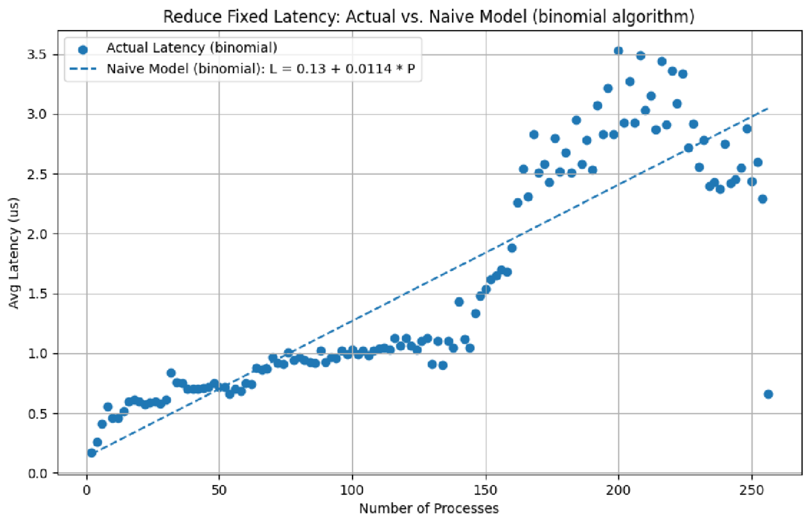

Figure 3-6: Line map of the binomial algorithm for Nave Model performance with a fixed size