

Project 2 Report for Problem 2.1

Zhu Liang

October 3, 2023

1 Project Description

In this project, we undertake two primary tasks. The first task is to provide a detailed description of the algorithms behind five collective communication operations provided by MPI. The descriptions are provided in Section 2.1. The second task is an empirical comparison between the built-in `MPI_Bcast` and a custom broadcast implementation named `MY_Bcast()`.

2 Algorithm Description

2.1 MPI Build-in Operations

`MPI_Bcast()`: This function broadcasts a message from the process with the designated root rank to all other processes in the communicator. All processes must call this function, with matching arguments.

`MPI_Scatter()`: This function distributes distinct blocks of data from the root process to each process in the communicator. The root sends data to itself as well as to the other processes.

`MPI_Allgather()`: Each process sends its own data to all other processes and gathers data from all processes. At the end, every process has the data from all the other processes.

`MPI_Alltoall()`: This function allows each process to send distinct data to every other process. It generalizes the functionality of both scatter and gather, with different data being sent to each process.

`MPI_Reduce()`: All processes in the communicator contribute their own data, which is combined (reduced) into a single result using a specified operation, like sum, max, etc. The result is stored on the root process.

2.2 Custom Broadcast Function: `MY_Bcast()`

In large-scale parallel computations, efficient data transfer across processes is paramount. A simple broadcast approach, which involves a root process sending messages to every other process sequentially, can be inefficient, especially when the number of processes grows, leading to a linear time complexity of $O(n)$.

To enhance this, we leverage it with a binary tree structure. Here, each process communicates only with its parent and potential left and right children (when children exist). Thus, it works even when the number of processes is a full binary tree. The root initiates the broadcast, and the message cascades down the tree, reaching all processes. Please refer to Figure 1 for an

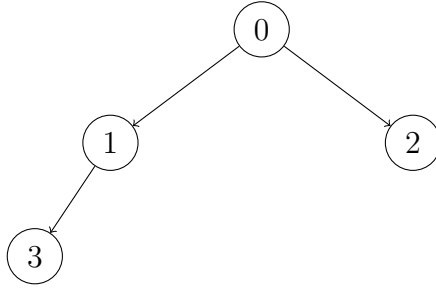


Figure 1: Binary Tree Structure when $P = 4$

illustration of the binary tree structure when $P = 4$, which is not a full binary tree. This approach reduces the time complexity from $O(n)$ to $O(\log(n))$.

For more details on the project structure and the implementation, please refer to the `README.md` file in the `project2` folder.

The pseudocode is shown below.

Algorithm 1 Custom Broadcast Function Using Binary Tree

```

1: procedure MY_BCAST
2:   Get rank and size of processes
3:   if current process rank is not root then
4:     Receive content of buffer from parent
5:   end if
6:   if left child exists then
7:     Send content of buffer to left child
8:   end if
9:   if right child exists then
10:    Send content of buffer to right child
11:  end if
12:  return MPI_SUCCESS
13: end procedure

```

3 Results

In the following tables, we present the execution times for two different broadcast implementations: `MPI_Bcast` and `MY_Bcast`. It is essential to note that while individual run times might vary depending on the specific runtime environment, the general pattern observed should remain similar across different runs.

	$P = 4$	$P = 7$	$P = 28$	$P = 37$
$N = 2^{10}$	0.000010s	0.000059s	0.000045s	0.000069s
$N = 2^{12}$	0.000023s	0.000054s	0.000061s	0.000063s
$N = 2^{14}$	0.000036s	0.000057s	0.000151s	0.000181s
$N = 2^{16}$	0.000120s	0.000125s	0.000454s	0.000526s

Table 1: Execution time using `MPI_Bcast`

	P = 4	P = 7	P = 28	P = 37
$N = 2^{10}$	0.000012s	0.000020s	0.000057s	0.000086s
$N = 2^{12}$	0.000010s	0.000024s	0.000072s	0.000091s
$N = 2^{14}$	0.000021s	0.000058s	0.000123s	0.000145s
$N = 2^{16}$	0.000057s	0.000165s	0.000385s	0.000416s

Table 2: Execution time using `MY_Bcast`

4 Analysis

Based on the empirical results obtained, both `MPI_Bcast()` and our custom implementation `MY_Bcast()` demonstrate closely matched performance. This is a significant observation given that the `MY_Bcast()` function was optimized to operate in $O(\log(n))$ time complexity, using a binary tree approach.

File Notes

The source code of the program is in the `project2` folder. For more details, please refer to the `README.md` file in the folder.