Project 2 Report for Probelm 2.1

Zhu Liang

September 26, 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 description 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()`

The `MY_Bcast()` function is designed to mimic the MPI broadcast functionality using non-blocking send and receive calls. It works by designating one process as the root, which sends the content of its buffer to all other processes. Each of the other processes waits to receive this content and store it in its own buffer.

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

The source code of the `MY_Bcast()` function is included in `functions.c` file. The pseudocode is shown below.

Algorithm 1 Custom Broadcast Function

```
1: procedure MY_ BCAST
2:     Get rank and size of processes
3:     if current process rank is root then
4:         for all the other processes i do
5:             Send content of buffer to process i
6:         end for
7:     else
8:         Receive content from root process into buffer
9:     end if
10:    return MPI_SUCCESS
11: 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
$N = 2^{10}$	0.000016s	0.000015s	0.000057s	0.000092s
$N = 2^{12}$	0.000013s	0.000023s	0.000076s	0.000089s
$N = 2^{14}$	0.000029s	0.000067s	0.000128s	0.000241s
$N = 2^{16}$	0.000095s	0.000164s	0.000356s	0.000655s

Table 1: Execution time using `MPI_Bcast`

	P = 4	P = 7	P = 28	P = 37
$N = 2^{10}$	0.000014s	0.000026s	0.000158s	0.000218s
$N = 2^{12}$	0.000015s	0.000032s	0.000194s	0.000293s
$N = 2^{14}$	0.000031s	0.000060s	0.000357s	0.000582s
$N = 2^{16}$	0.000086s	0.000175s	0.001076s	0.001689s

Table 2: Execution time using `MY_Bcast`

4 Analysis

In this section, we delve into the observed performance variations between the two broadcasting functions, `MPI_Bcast` and `MY_Bcast`. Our primary objective is to decipher the underlying reasons behind their distinct behaviors under varying conditions.

4.1 Summary of Results

From the results presented in the previous section, it is evident that `MY_Bcast` tends to perform better with a smaller number of cores, while `MPI_Bcast` shows superior performance as the number of cores increases.

4.2 Possible Explanations

Several hypotheses might explain the observed performance trends:

1. **Simplicity of Implementation:** The `MY_Bcast` is simpler, without involving hierarchical broadcasting, considerations for network topology, or other advanced features. In situations with fewer cores, this simplicity might lead to higher efficiency.
2. **Startup Overhead:** The `MPI_Bcast` operation might have certain startup overheads, especially when setting up the broadcast operation or initializing network communications. As the number of cores increases, these overheads get amortized over more parallel work, becoming a smaller concern. However, with fewer cores, these overheads might be more pronounced.
3. **Advanced Optimizations and Features:** `MPI_Bcast` might employ various advanced optimizations and features. These optimizations might become evident in large-scale parallel environments but might not be as prominent with fewer core counts.

4.3 Conclusion

The comparison offers valuable insights into parallel computing. The key takeaway is the importance of understanding the relationship between software design and hardware. It's evident that the right choice depends on the specific scenario and that no single solution is universally optimal.

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.