MPI File Transfer Program Report

Hoang Minh Quan
ID 23BI14371

December 5, 2025

Contents

1 Introduction

This report presents an MPI-based file transfer program implemented in C++. The goal of the program is to allow one MPI process (the `ROOT` process) to read a file and distribute its contents to all other MPI processes using the `MPI_Bcast` communication primitive.

Each non-root process receives the file and writes it to a new local file named according to its MPI rank.

2 Program Objectives

The application fulfills the following objectives:

- The ROOT process reads the name and contents of a file.
- The file name and its length are broadcast to all MPI processes.
- The file data buffer is broadcast to all MPI processes.
- Each receiver process writes the received data into a new file named:

  `received_rankX_filename`

The program requires at least 2 MPI processes:

- One sender (rank 0)
- One or more receivers (ranks 1, 2, . . .)

3 System Illustration

Figure 1 shows a high-level view of the MPI file transfer system. Rank 0 acts as the sender: it reads the input file and broadcasts its contents to all other ranks, which act as receivers.

4 Source Code

The full source code of the MPI program is shown below.

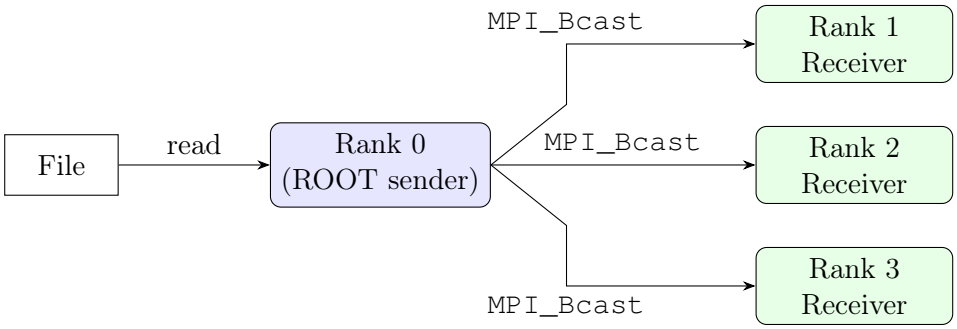

Figure 1: High-level illustration of the MPI file transfer system.

```cpp
#include <mpi.h>
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

int main(int argc, char** argv) {
    MPI_Init(&argc, &argv);

    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    const int ROOT = 0;

    if (size < 2) {
        if (rank == ROOT) {
            std::cerr << "Need at least 2 MPI processes (1
               sender + 1 receiver)." << std::endl;
        }
        MPI_Finalize();
        return 1;
    }

    std::string filename;
    std::vector<char> nameBuf;
    int nameLen = 0;

    if (rank == ROOT) {
        if (argc < 2) {
```

```
30              std::cerr << "Usage: mpirun -np <n> ./
                    mpi_file_transfer <file_to_send>\n";
31              MPI_Abort(MPI_COMM_WORLD, 1);
32          }
33          filename = argv[1];
34
35          nameBuf.assign(filename.begin(), filename.end());
36          nameBuf.push_back('\0');
37          nameLen = static_cast<int>(nameBuf.size());
38      }
39
40      MPI_Bcast(&nameLen, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
41
42      if (rank != ROOT) {
43          nameBuf.resize(nameLen);
44      }
45
46      MPI_Bcast(nameBuf.data(), nameLen, MPI_CHAR, ROOT,
            MPI_COMM_WORLD);
47
48      if (rank != ROOT) {
49          filename = std::string(nameBuf.data());
50      }
51
52      int fileSize = 0;
53      std::vector<char> buffer;
54
55      if (rank == ROOT) {
56          std::ifstream in(filename, std::ios::binary);
57          if (!in.is_open()) {
58              std::cerr << "Rank " << rank << ": cannot open
                        file '" << filename << "' for reading.\n";
59              MPI_Abort(MPI_COMM_WORLD, 1);
60          }
61
62          in.seekg(0, std::ios::end);
63          fileSize = static_cast<int>(in.tellg());
64          in.seekg(0, std::ios::beg);
65
66          buffer.resize(fileSize);
67          if (fileSize > 0) {
68              in.read(buffer.data(), fileSize);
69          }
```

```
70          in.close();
71
72          std::cout << "Rank " << rank << ": read " <<
               fileSize
73                    << " bytes from '" << filename << "'.\n";
74      }
75
76      MPI_Bcast(&fileSize, 1, MPI_INT, ROOT, MPI_COMM_WORLD);
77
78      if (fileSize == 0) {
79          if (rank != ROOT) {
80              std::cerr << "Rank " << rank
81                        << ": received file size 0, nothing
                             to write.\n";
82          }
83          MPI_Finalize();
84          return 0;
85      }
86
87      if (rank != ROOT) {
88          buffer.resize(fileSize);
89      }
90
91      MPI_Bcast(buffer.data(), fileSize, MPI_CHAR, ROOT,
           MPI_COMM_WORLD);
92
93      if (rank != ROOT) {
94          std::string outName = "received_rank"
95                                + std::to_string(rank)
96                                + "_" + filename;
97
98          std::ofstream out(outName, std::ios::binary);
99          if (!out.is_open()) {
100             std::cerr << "Rank " << rank
101                       << ": cannot open '" << outName << "'
                             for writing.\n";
102             MPI_Abort(MPI_COMM_WORLD, 1);
103         }
104
105         out.write(buffer.data(), fileSize);
106         out.close();
107
108         std::cout << "Rank " << rank << ": wrote " <<
```

```
            fileSize
109                 << " bytes to '" << outName << "'.\n";
110     }
111
112     MPI_Finalize();
113     return 0;
114 }
```

Listing 1: MPI File Transfer Source Code

5 How to Compile

To compile the program on a system with MPI installed:

```
mpic++ mpi_file_transfer.cpp -o mpi_file_transfer
```

6 How to Run the Program

Run the program with at least 2 processes:

```
mpirun -np 4 ./mpi_file_transfer example.bin
```

Expected behavior:

- Rank 0 reads `example.bin`.
- All ranks receive the file through MPI broadcast.
- Ranks 1, 2, 3 write:

```
received_rank1_example.bin
received_rank2_example.bin
received_rank3_example.bin
```

7 Conclusion

This MPI application demonstrates how to efficiently distribute file data from one process to all others using collective communication.

The program:

- Uses `MPI_Bcast` for distributing metadata and file contents.
- Avoids sending the file multiple times by using a single collective operation.
- Ensures each process receives an identical copy of the input file.

This approach is well-suited for small to medium file sizes and showcases a clean example of using MPI for data distribution.