

Assignment 4: MPI Collectives and MPI-IO

Isaías A. Comprés Ureña
compresu@in.tum.de

Prof. Michael Gerndt
gerndt@in.tum.de

06-01-2016

1 Introduction

This final assignment will focus on collective and parallel IO operations available in MPI libraries. There is a large set of collective operations defined in the MPI standard and their use can greatly simplify the development process of parallel applications, while providing better performance when compared to equivalent point-to-point communication patterns. The set of parallel IO operations allows MPI libraries to abstract the underlying parallel file system implementation in a generic and standard interface.

The use of MPI collectives provides several benefits to MPI developers. It is recommended that developers use collectives in their applications as much as possible. A programmer is free to emulate any collective operation through the use of point-to-point communication, but this would result on more code to verify for correctness. Additionally, MPI collectives use well tuned distributed algorithms that reduce the amount of communication required to complete the operations. These algorithms are designed with scalability in mind; therefore, applications that rely on MPI collectives are generally easier to scale to large number of processes. Last but not least, MPI implementations will use hardware acceleration for these operations when available.

The MPI-IO interface allows MPI applications to access parallel file systems through a standard interface. There are several parallel file systems available and while they provide POSIX interfaces, they usually require the use of proprietary interfaces in order to access their more attractive features and performance. Having a higher level and standard interface simplifies the development process and improves portability. Additionally, MPI can reduce and streamline accesses to the file system through optimizations such as data sieving and 2-phase IO.

In this assignment, students will continue to tweak the provided Gaussian Elimination code. The students will need to transform it to use MPI collective operations on identified communication patterns. Additionally, the initialization code will be updated to use MPI-IO to process the input matrix and vector, and output the solution vector in a distributed manner. In both of these main tasks, the aim is to improve the quality and portability of the code. These changes may also improve the performance and scalability of the application.

1.1 Submission Instructions

Your final assignment submission for Programming of Supercomputers will consist of 2 parts:

- A 5 to 10 minute video with the required comments described in each task.
- A compressed tar archive with the required files described in each task.

2 MPI Collectives

A common mistake when developing MPI applications is to rely on point-to-point communication when there are clear patterns that can be replaced with an available collective operation provided by MPI. A valuable

skill to develop as a developer is to be aware of these patterns.

For this task, take some time to revisit the documentation of the following MPI collectives and their non-blocking versions:

- `MPI_Allgather/v`
- `MPI_Allreduce`
- `MPI_Alltoall/v/w`
- `MPI_Bcast`
- `MPI_Gather/v`
- `MPI_Reduce`
- `MPI_Reduce_scatter`
- `MPI_Scatter/v`

With a refreshed understanding of what each of these operations do, perform the following tasks:

1. Take a careful look at the provided Gaussian Elimination code.
2. Identify each communication pattern in the code that can be replaced with any of the collectives above.
3. Evaluate whether the application can achieve overlap in any of the identified locations.
4. Rewrite each of the identified communication patterns with the appropriate collective (blocking or non-blocking version).

2.1 Required Submission Files

Submit the updated Gaussian Elimination code.

2.2 Required Video Commentary

Answer the following questions:

1. Which patterns were identified and replaced with collective communication in the code? Explain.
2. Were you able to identify any potential for overlap and used any non-blocking collectives?
3. Was there any measurable performance or scalability improvement as a result of these changes?
4. Is the resulting code easier to understand and maintain after the changes? Why?

3 MPI Parallel IO

In the provided Gaussian Elimination code the initial read of input files is done at rank 0. After loaded, rank 0 then proceeds to distribute parts of the input to each of the participating ranks. In addition, after each rank has computed their partial solution, these are collected at rank 0 again. In the end, rank 0 holds all input and output matrices.

Before you start, make sure to understand MPI-IO operations to perform the following tasks:

- Opening and closing files.

- Reading and writing from files.
- Setting file views.
- Collective IO operations.

Use the MPI-IO operations to change the Gaussian Elimination code such that:

1. Each rank reads its own initial blocks of data from the input files.
2. Each rank writes its own partial solution of the solution vector to a common output file.

3.1 Required Submission Files

Submit the updated Gaussian Elimination code.

3.2 Required Video Commentary

Answer the following questions:

1. Which MPI-IO operations were applied to transform the code? Explain your choices.
2. What is "Data Sieving" and "2-Phase IO"? How do they help improve IO performance?
3. Was the original implementation scalable in terms of IO performance?
4. Was the original implementation scalable in terms of RAM storage?
5. How much of the communication in the application was replaced or eliminated with MPI-IO?
6. Were there any performance improvements due to the change to MPI-IO?