Assignment 4:
MPI Collectives and MPI-IO

Madhura Kumaraswamy
kumarasw@in.tum.de

Prof. Michael Gerndt
gerndt@in.tum.de

21-12-2018

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 though optimizations such as data sieving and 2-phase IO.

In this assignment, students will continue to tweak the provided Gaussian Elimination code by transforming 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.

1.1 Submission Instructions

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

1. A 5 to 15 page report with the required answers.
   - Submit the report in PDF format.
   - Plots and figures should be used to enhance any explanations.
   - Provide PRECISE answers for each task in the same order as the questions.
   - The report must contain the contribution of each group member to the assignment (max. 2 sentences/member).
2. A compressed tar archive with the required files described in each task.

2 Gaussian Elimination

2.1 Provided MPI Implementation

You will use the same implementation and the modules provided for the previous assignment. This implementation relies on MPI blocking point-to-point communication. The Load-Leveler batch script was also provided together with the implementation. The instructions for generating traces for Vampir are the same as in the previous assignment.

Like the previous assignment, the tests will be performed with a maximum of 4 nodes and 64 MPI processes with the input files for sizes: 64x64, 512x512, 1024x1024, 2048x2048, 4096x4096 and 8192x8192. Make sure to verify this in the job script.

3 MPI Collectives (47 points total)

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).

3.1 Required Submission Files

1. The updated `gauss.c`. (16 points)
2. The performance plots and description in the report. (7 points)

3.2 Questions (25 points)

1. Which patterns were identified and replaced with collective communication in the code? Explain. (8 points)
2. Were you able to identify any potential for overlap and used any non-blocking collectives? (Use Vampir) (6 points)
3. Was there any measurable performance or scalability improvement as a result of these changes? (6 points)
4. Is the resulting code easier to understand and maintain after the changes? Why? (4 points)

4 MPI Parallel IO (53 points total)

In the provided Gaussian Elimination code, the initial read of the input files is done at rank 0. Then, rank 0 distributes parts of the input to each of the participating ranks. After each rank has computed its partial solution, the individual solutions are collected at rank 0 again. Ultimately, rank 0 holds all the 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.

4.1 Required Submission Files

1. The updated `gauss.c`. (16 points)
2. The performance plots and description in the report. (7 points)

4.2 Questions (31 points)

1. Which MPI-IO operations were applied to transform the code? Explain your choices. (8 points)
2. What is "Data Sieving" (2 points) and "2-Phase IO" (2 points)? How do they help improve IO performance? (2 points)
3. Was the original implementation scalable in terms of IO performance? (2 points)
4. Was the original implementation scalable in terms of RAM storage? (2 points)
5. How much of the communication in the application was replaced or eliminated with MPI-IO? (Use Vampir) (6 points)
6. Were there any performance improvements due to the change to MPI-IO? (6 points)