Measure CMSSW Performance running as a distributed application

Supervised by :
Dr. Fawaz Alazemi
Dr. Anderea Bocci

Date of submission
7th September 2023

Abeer Alsafran

Abstract

CMS is a Compact Muon Solenoid. It is one of the main projects at CERN. It is a general-purpose detector at the Large Hadron Collider (LHC), the CMS detector uses a huge solenoid magnet to bend the paths of particles from collisions in the LHC. It has a broad physics programme ranging from studying the Standard Model (including the Higgs boson) to searching for extra dimensions and particles that could make up dark matter [1].

One of the main cores of CMS is CMSSW which is a short for Compact Muon Solenoid Software. It is a main software in CMS. It has a lot of components used in it, one of the component is MPI, which is Message Passing Interface. It is a standardized and portable message-passing standard designed to function on parallel computing architectures. It is used to transfer data from one process to another using a unique interface. What is done by MPI is transferring the data using multiple MPI techniques. Each technique is an approach that may be useful to reduce latency and increase performance. There are 7 approaches in total, each one of them uses a different function in MPI library. Four of them were already implemented, and three were implemented by me.

Table of Contents

List of Figures

Chapter 1
Introduction

1.1 Problem statement

The main problem is that, transferring the data always takes a significant amount of time due to copying from one process to another one. Since time is very important to physicists to do their calculations. We need to improve the time taken to send and receive data between processes. One thing that we computer scientist can do to help is, optimizing the data transition with respect to time and performance.

1.2 Aim and Objectives

We need to use different approaches to improve time taken to transfer data, and to use more than one communication method to send and receive data. Also to measure the time on each method, and try to compare them to get the best outcome. So far there are 4 methods to send and receive the data, Blocking Scatter, Non-blocking Scatter, Blocking Send, and Non-blocking Send. And the new methods are, Blocking SendRecv and One Sided Communication Master and One Sided Communication Worker.

Chapter 2

Methodology

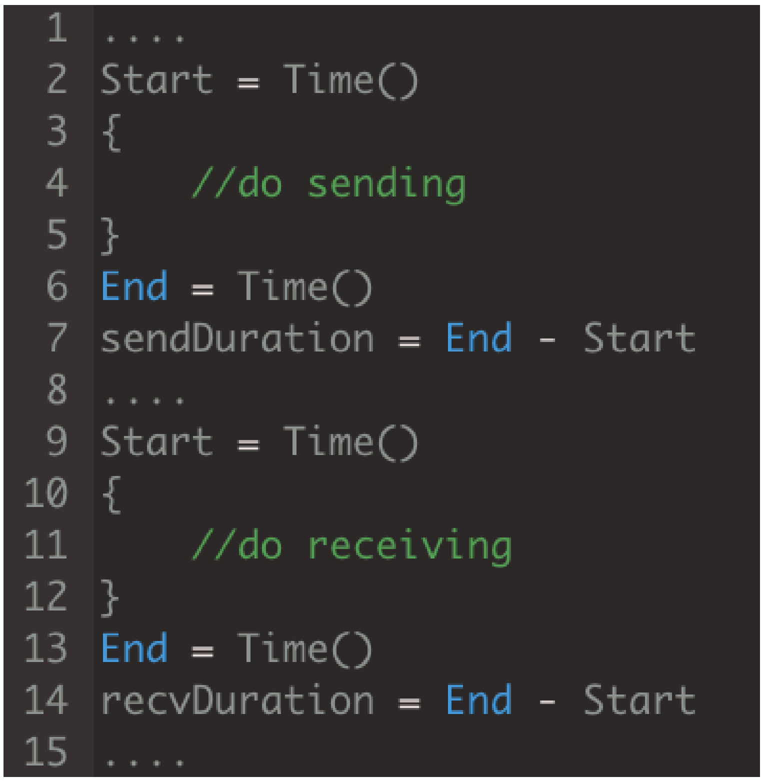
To measure the time and compare the different methods used to transfer the data we need to measure the time from sending to receiving the data for each one of the different methods. We need to calculate the time before the send function, (which sends the data from master), and after the receive function, (which receives the data sent from the master). After that we try another method, to check the time in the same way.

2.1 SendRecv method

In this function the single process can send and receive data using the same function. This method allow sending and receiving the data in one call only. And this should make it faster.
For example:
• MPISendRecv(const void *sendbuf, int sendcount, MPIDatatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPIDatatype recvtype, int source, int recvtag, MPIComm comm, MPIStatus *status)
It has 12 parameter:
1- sendbuf: Initial address of send buffer (choice).
2- sendcount: Number of elements to send (integer).
3- sendtype: Type of elements in send buffer (handle).
4- dest: Rank of destination (integer).
5- sendtag: Send tag (integer).
6- recvbuf: Initial address of recv buffer (choice).
7- recvcount: Maximum number of elements to receive (integer).
8- recvtype: Type of elements in receive buffer (handle).
9- source: Rank of source (integer).
10- recvtag: Receive tag (integer).
11- comm: Communicator (handle).
12- status: This refers to the receive operation.

This function has been used as the following, the master process will use the above function as sender function, all the sending data will be set the corresponding parameters, as for the receiving parameters, they will be nulls, because we are using it as a sender only. And if we want to use the above function as a receiver, the receiving data will be set and the sending parameters will be set to nulls, because we are dealing with it as a sender and receiver function at once.

This is an example of how the flow is working, first we take the current time then we call the send function and then we stop the time and take the time difference. Same goes for the receiving time.



```
....
Start = Time()
{
    //do sending
}
End = Time()
sendDuration = End - Start
....
Start = Time()
{
    //do receiving
}
End = Time()
recvDuration = End - Start
....
```

Figure 2.1: Time Stamp

2.2 OneSidedComm method

In this function the single process can write to the other process window and the other process can read from his window to get the data from the master process. This method allow each process to create a window in the current process, this window allows the current process to write/put on the buffer (window) and to read/get from the buffer (window).
For example:
• MPI-Put (const void *originAddr, int originCount, MPIDatatype originDatatype, int targetRank, MPIAint targetDisp, int targetCount, MPIDatatype target-Datatype, MPIWin win)
• MPI-Get (void *originAddr, int originCount, MPIDatatype originDatatype, int targetRank, MPIAint targetDisp, int targetCount, MPIDatatype target-Datatype, MPIWin win) [2] . These two functions share some parameters but each one of these functions has a different approach. the MPI-Put function is for writing and the MPI-Get function is for reading.

There is eight parameters:
1- origin-addr: Initial address of origin buffer (choice).
2- origin-count: Number of entries in origin buffer (nonnegative integer).
3- origin-datatype: Data type of each entry in origin buffer (handle).

4- target-rank: Rank of target (nonnegative integer).
5- target-disp: Displacement from window start to the beginning of the target buffer (nonnegative integer).
6- target-count: Number of entries in target buffer (nonnegative integer).
7- target-datatype: datatype of each entry in target buffer (handle).
8- win: window object used for communication (handle).

2.2.1 Master

The master process will PUT the data to the window and will GET the result from the window.

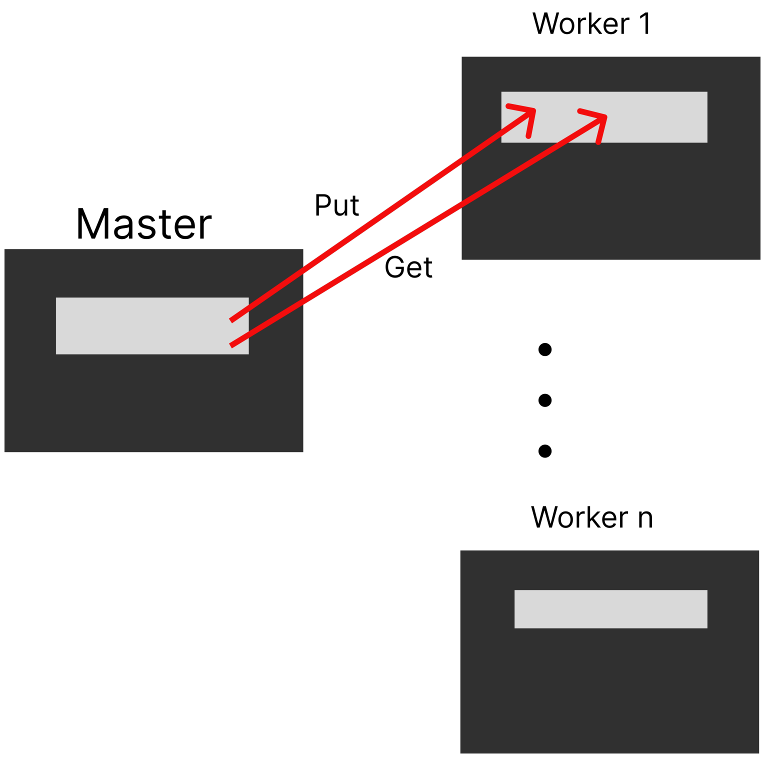


Figure 2.2: Master

2.2.2 Worker

The worker process will GET the data from the master's window and will PUT the result in the window.

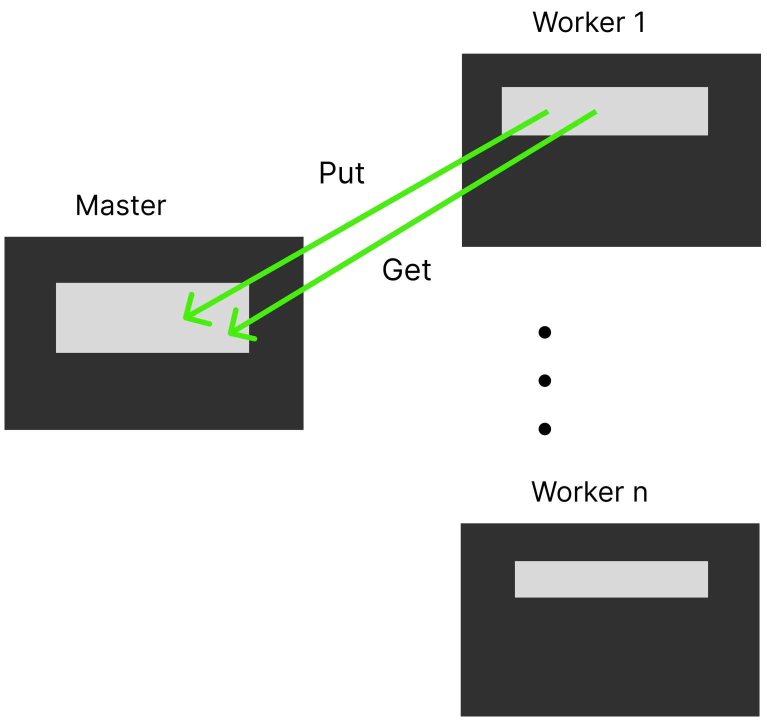


Figure 2.3: Worker

Chapter 3

Results

3.1 SendRecv function

From a theoretical point of view the sendRecv method could be a fast method for sending and receiving the data, since it has a shared pool with the other processes. But in reality it has kind of latency in comparison with other methods used in the same project. In the contrary of what was expected, It has a very high latency during the sending process, as for the receiving process, it was one of the fastest methods in receiving time in compression with the other methods in the project.

3.2 OneSidedComm function

One sided communication was the best one so far, it was the fastest in sending and receiving the data from multiple processes. This was expected, because the one sided communication has a different idea about sending and receiving, it is more like writing and reading from a shred buffer.

The following plot shows the sending and receiving time for each method used in the project. The process number is fixed to 15 and the iterations are fixed to 10000.

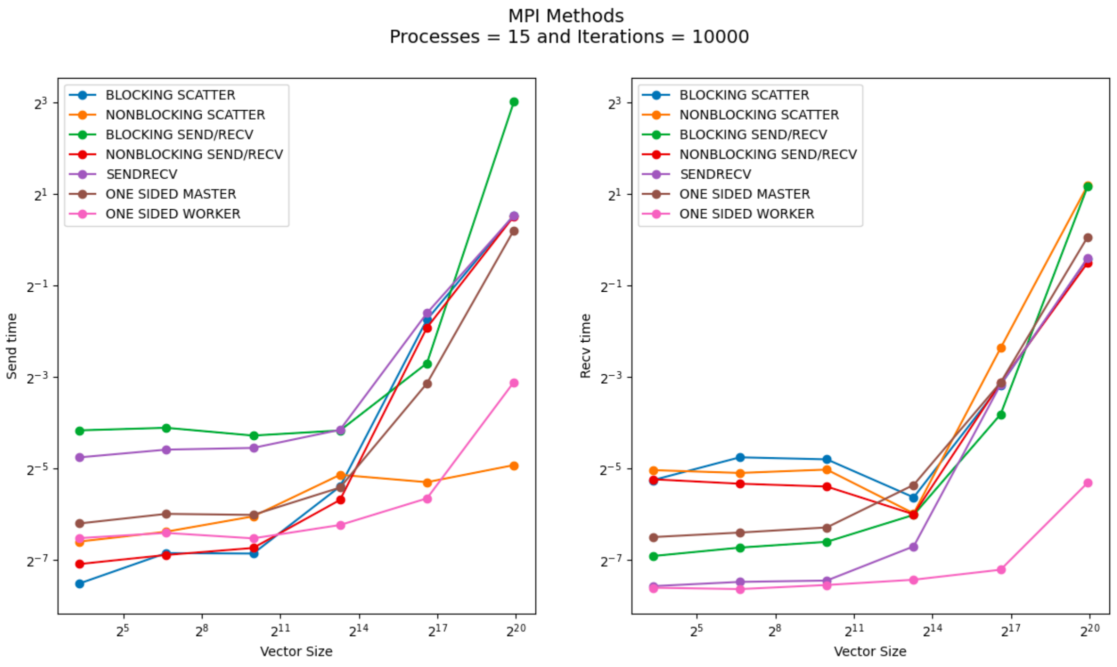


Figure 3.1: Results

Chapter 4

Conclusion

In conclusion, Improving the time taken to transfer the data is not an easy task. Yet it is very essential to improve the program performance. Based on the results, the most efficient method was the one sided communication because it is a non blocking method so it does not need to wait for other process and it has a shred buffer among all the processes. The most inefficient method was the blocking send because it is a blocking method so it needs to wait for other processes.
Some of the lessons learned, MPI is better when we need to send data. Non blocking has the best performance. Finally, profiling and analysis are mandatory when it comes to measure performance.

References

[1] CERN. https://home.cern/science/experiments/cms. *experiments*, 2023.

[2] Open MPI. https://www.open-mpi.org/doc/current/. *Organization*, 2023.