MYCHP203 — TOP : Lab 4

Gabriel Dos Santos
gabriel.dossantos@cea.fr
gabriel.dos-santos@uvsq.fr

Hugo Taboada
hugo.taboada@cea.fr

April 23rd, 2024

MPI optimizations: synchronization and communication overhead, tips & tricks.

I Generalities

1. Give the definition of network latency.

2. Give the definition of network bandwidth.

3. Give their respective units.

4. Give the name and version of the MPI library you are using. Give the command used to retrieve this information.

5. To make sure your installation works, you can use the `hostname` command, which returns the name of the current process's host:

```
mpirun -np 2 hostname
```

What happens when all processes are executed on the same host?

II Ping Pong

II.1 Set up

In this section, you will do performance measurementes using the ping-pong benchamrk to familiarize with MPI's characteristics communication times.
Start by writing a ping-pong between two MPI processes. Rank 0 will send a message, and rank 1 will receive it. Write your code so that you can set the message size as a command-line argument.

6. The code hereafter is a simple ping from rank 0 to rank 1 (no answer). We have voluntarily added a `sleep(2)` in the rank 1's code.

```
if (rank == 0) {
    t0 = MPI_Wtime();
    MPI_Send(buffer, size, MPI_CHAR, 1, 0, MPI_COMM_WORLD);
    t1 = MPI_Wtime();
    printf("%lu\t%g\n", size, t1 - t0);
} else if (rank == 1) {
    sleep(2);
    MPI_Recv(buffer, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
```

Using OpenMPI 2.0.1, we measured the execution time as a function of message size. We obtain 53.1654 ms for a size of 4,000 Bytes, and 2.0006 s for a size of 4,096 Bytes.
How do you explain this difference? What did we actually measure?

7. Find the message size for which this gap appears in *your* MPI implementation. This size might be different than the one we observed using OpenMPI 2.0.1 (4,096 B). Explain the reason why.

8. We decide to remove the call to `sleep()`. Does the measure make sense now? Why?

9. Propose an alternative to measure the *actual* sending time of a message (i.e. the time it took to actually receive it on the target rank). Write the corresponding program.
It is strongly advised that you read the MPI documentation of the different modes of communications: Standard, Buffered, Ready, Synchronous, etc.

10. Write an actual MPI ping-pong (rank 1 answers back to rank 0). Explain why the measured times corresponds to a message exchange.

II.2 First measures

11. Do multiple measures with multiples of 32 Bytes for the message sizes. What do you see?

12. We propose adding a barrier before the message exchange pahase. Explain what is the interest of this barrier with regards to the accuracy of the measurements.

13. Rerun the measures of question **11**. Are the changes from the previous question sufficient? Why?

II.3 Repetitions

When taking a measurement, it is important to remember that our environment does not allow us to reproduce the exact conditions between each run. What is more, time measurement itself is fraught with uncertainty. uncertainty. To solve this problem, we prefer to repeat a measurement and calculate an average (mean) and/or a median.

14. Update the program to add repetitions. You should now print the average execution time.

II.4 Impact of message size

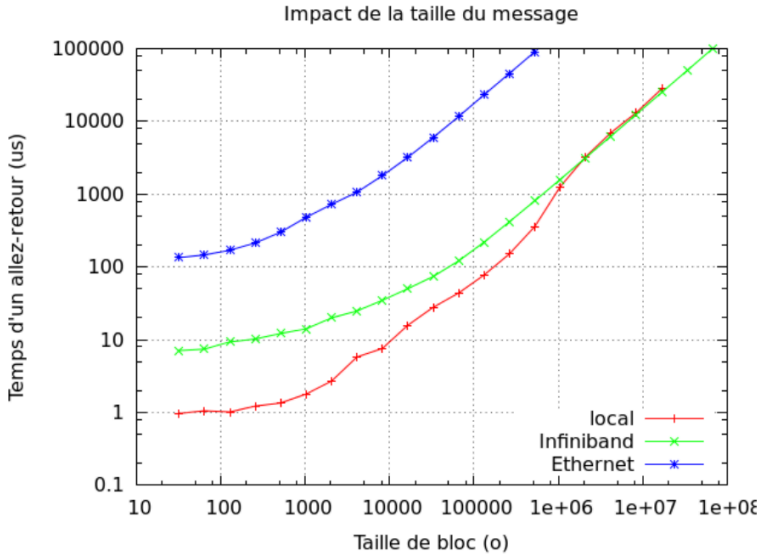

Fig. 1. – Communication time as a function of message size

Fig. 1 shows the evolution of ping-pong time according to message size between intra-node (local) and inter-node (Ethernet and Infiniband) exchanges. Scales are logarithmic.

15. Explain why there is such a big gap between local and Infiniband for small sizes.
Explain this gap shrinks progressively once the message size increases.

16. From the previous results, should we send distinct sets of data separately? Or should we try to group them together? Is it true for all sizes?

III Experimental evaluation of scalability

Let's start with a very simple benchmark. Our MPI program will initially make no communication. Have node 0 measure the program's execution time. It will perform a simple sum of two vectors, in the following form:

$$X_i^{t+1} = X_i^t + c$$

with X a vector of size N and c a real constant.
Clearly, such an equation can be distributed over several MPI processes without any communication. So we'll simply implement the sum method and execute it in a loop to obtain a suitable execution time for our measurement.

17. Implement this benchmark like so:
- Initialize the MPI context
- Pass the vector size and the number of repetitions as parameters of the program
- Allocate and initialize memory for two vectors. Each node shall compute the sum on a vector of size $\frac{N}{\text{nb}_{\text{tasks}}}$
- Make sure to measure the execution time as seen previously in this lab

18. What does strong scalability represent?

19. What does weak scalability represent?

20. To introduce communications in our program, we want to consider a case inspired by finite elements method (FEM). Our equation is now:

$$X_i^{t+1} = \frac{X_{i-1}^t + 2X_i^t + X_{i+1}^t}{4}$$

Here, MPI slicing requires the addition of communications for border elements. Modify the application in this way and evaluate scalability.

21. Introduce a global synchronization in the repetitions loop and reevalute the application's scalability. What do you see? What remarks can you make about communications and the use of barriers in MPI applications? What should you do to avoid this problem?

IV Collectives and algorithms in OpenMPI

Install a recent version of OpenMPI (4.0.5+) on your system. The command `ompi_info -all` gives you multiple informations about you installation. We will use to know which algorithms are available in the `coll tuned` module of OpenMPI, where blocking collectives are implemented.

22. Find what are the usable algorithms for the `MPI_Bcast` routine. Compare their performance depending on the number of MPI processes and buffer size.

23. Find what are the usable algorithms for the `MPI_Gather` routine. Compare their performance depending on the number of MPI processes and buffer size.

24. Find what are the usable algorithms for the `MPI_Reduce` routine. Compare their performance depending on the number of MPI processes and buffer size.

25. Find what are the usable algorithms for the `MPI_Alltoall` routine. Compare their performance depending on the number of MPI processes and buffer size.

26. For each collective, why do we need multiple algorithms?