Delft University of Technology

Introduction to High Performance Computing
WI4049TU

Lab Report

Author:
Elias Wachmann (6300421)

October 8, 2024

0 Introduction

In the introductory lab session, we are taking a look at some basic features of MPI.
We start out very simple with a hello world program on two nodes.

Hello World

```c
#include "mpi.h"
#include <stdio.h>

int np, rank;

int main(int argc, char **argv)
{
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &np);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);

  printf("Node %d of %d says: Hello world!\n", rank, np);

  MPI_Finalize();
  return 0;
}
```

This program can be compiled with the following command:

```
mpicc -o helloworld1.out helloworld1.c
```

And run with:

```
srun -n 2 -c 4 --mem-per-cpu=1GB  ./helloworld1.out
```

We get the following output:

```
Node 0 of 2 says: Hello world!
Node 1 of 2 says: Hello world!
```

From now on I'll skip the compilation and only mention on how many nodes the program is run and what the output is / interpretation of the output.

0.a) Ping Pong

I used the template to check how long `MPI_Send` and `MPI_Recv` take. The code can be found in the appendix for this section.

I've modified the printing a bit to make it easier to gather the information. Then I piped the program output into a textfile for further processing in python. I ran it first on one and then on two nodes as specified in the assignment sheet. Opposed to the averaging over 5 send / receive pairs, I've done 1000 pairs. Furthmore I reran the whole programm 5 times to gather more data. All this data is shown in the following graph:

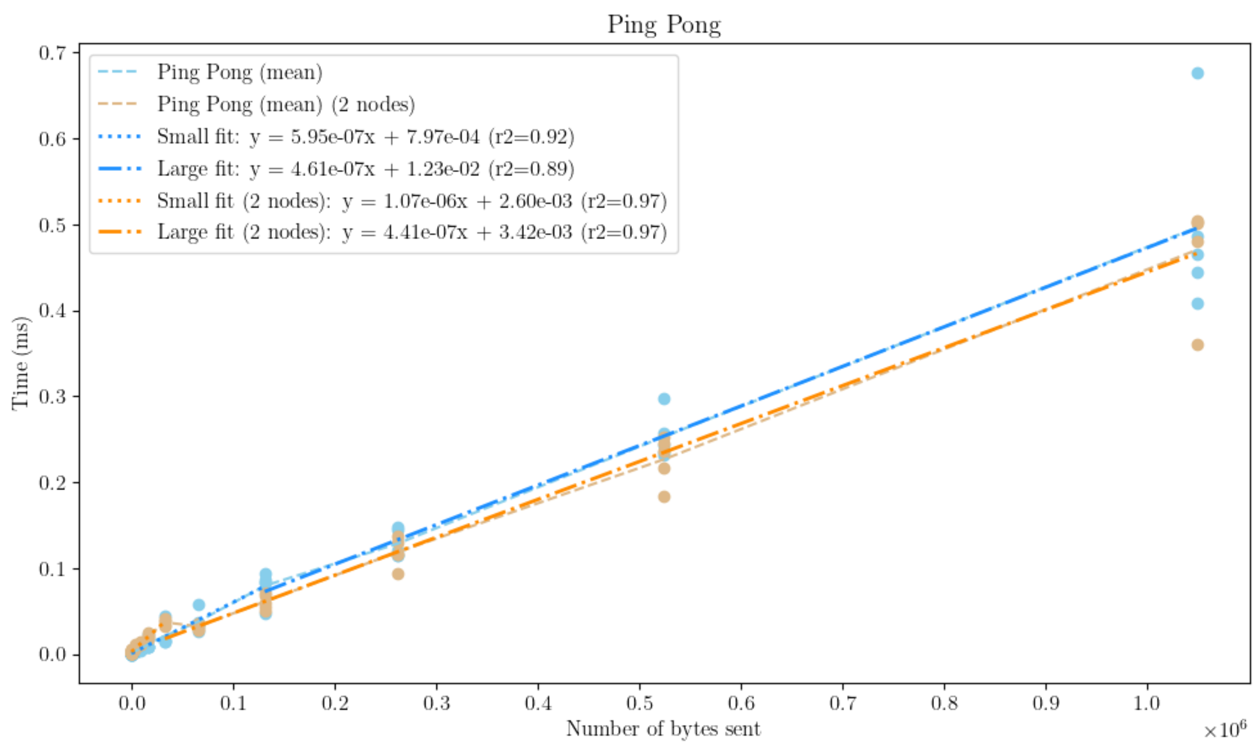

Figure 1: Ping Pong: Number of bytes sent vs. average time taken from 1000 pairs of send / receive. 5 runs shown for each size as scatter plot. Mean of these 5 runs shown as line. Blue small fit includes all data points up to 131072 bytes, blue large from there. Red small fit includes all data points up to 32768 bytes, red large from there.

As can be seen in the data and the fits, there are outliers especially for the larger data sizes.
For our runs we get the following fits and R^2 values:

Run Type	**Data Size**	**Fit Equation**	**R^2 Value**
Single Node	Small (<=131072)	$5.95 \times 10^{-7} \cdot x + 7.97 \times 10^{-4}$	0.92
Single Node	Large (>= 131072)	$4.61 \times 10^{-7} \cdot x + 1.23 \times 10^{-2}$	0.89
Two Node	Small (<=32768)	$1.07 \times 10^{-6} \cdot x + 2.60 \times 10^{-3}$	0.97
Two Node	Large (>=32768)	$4.41 \times 10^{-7} \cdot x + 3.42 \times 10^{-3}$	0.97

Table 1: Fit Equations and R^2 Values for Single Node and Two Node Runs

TODO: Further analysis needed?

Extra: Ping Pong with MPI_SendRecv

We do the same analysis for the changed program utilizing `MPI_SendRecv`. The code can be found in the appendix for this section.
We get the following graph from the measurements which were performed in the same way as for the previous program:

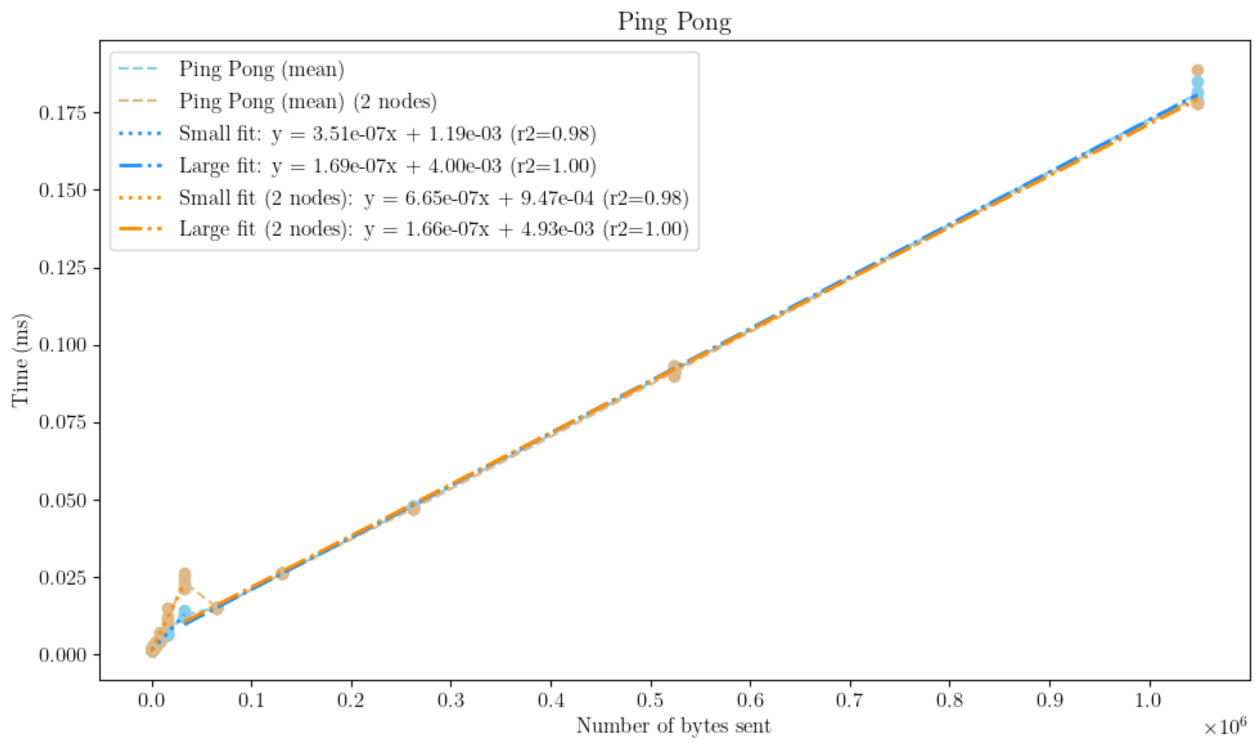

Figure 2: Ping Pong with MPI_SendRecv: Number of bytes sent vs. average time taken from 1000 pairs of send / receive. 5 runs shown for each size as scatter plot. Mean of these 5 runs shown as line. Blue small fit includes all data points up to 32768 bytes, blue large from there. Red small fit includes all data points up to 32768 bytes, red large from there.

We get the following fits and R² values for the runs:

Run Type	**Data Size**	**Fit Equation**	**R² Value**
Single Node	Small (<=32768)	$3.51 \times 10^{-7} \cdot x + 1.19 \times 10^{-3}$	0.98
Single Node	Large (>=32768)	$1.69 \times 10^{-7} \cdot x + 4.00 \times 10^{-3}$	1.00
Two Node	Small (<=32768)	$6.65 \times 10^{-7} \cdot x + 9.47 \times 10^{-4}$	0.98
Two Node	Large (>=32768)	$1.66 \times 10^{-7} \cdot x + 4.93 \times 10^{-3}$	1.00

Table 2: Fit Equations and R² Values for Single Node and Two Node Runs

TODO: Further analysis - less variance etc

0.b) MM-product

0 Appendix - Introduction

The following code was used for the ping pong task:

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

// Maximum array size 2^20= 1048576 elements
#define MAX_EXPONENT 20
#define MAX_ARRAY_SIZE (1<<MAX_EXPONENT)
#define SAMPLE_COUNT 1000

int main(int argc, char **argv)
{
    // Variables for the process rank and number of processes
    int myRank, numProcs, i;
    MPI_Status status;

    // Initialize MPI, find out MPI communicator size and process rank
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myRank);


    int *myArray = (int *)malloc(sizeof(int)*MAX_ARRAY_SIZE);
    if (myArray == NULL)
    {
        printf("Not enough memory\n");
        exit(1);
    }
    // Initialize myArray
    for (i=0; i<MAX_ARRAY_SIZE; i++)
        myArray[i]=1;

    int number_of_elements_to_send;
    int number_of_elements_received;

    // PART C
    if (numProcs < 2)
    {
        printf("Error: Run the program with at least 2 MPI tasks!\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    double startTime, endTime;

    // TODO: Use a loop to vary the message size
    for (size_t j = 0; j <= MAX_EXPONENT; j++)
    {
        number_of_elements_to_send = 1<<j;
        if (myRank == 0)
        {
            myArray[0]=myArray[1]+1; // activate in cache (avoids possible delay when sending the 1st element)
            startTime = MPI_Wtime();
            for (i=0; i<SAMPLE_COUNT; i++)
            {
                MPI_Send(myArray, number_of_elements_to_send, MPI_INT, 1, 0,
                    MPI_COMM_WORLD);
                MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
                MPI_Get_count(&status, MPI_INT, &number_of_elements_received);

                MPI_Recv(myArray, number_of_elements_received, MPI_INT, 1, 0,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            } // end of for-loop

            endTime = MPI_Wtime();
            printf("Rank %2.1i: Received %i elements: Ping Pong took %f seconds\n", myRank, number_of_elements_received,(endTime - startTime)/(2*SAMPLE_COUNT));
        }
        else if (myRank == 1)
        {
            // Probe message in order to obtain the amount of data
            MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status);
            MPI_Get_count(&status, MPI_INT, &number_of_elements_received);

            for (i=0; i<SAMPLE_COUNT; i++)
            {
                MPI_Recv(myArray, number_of_elements_received, MPI_INT, 0, 0,
                MPI_COMM_WORLD, MPI_STATUS_IGNORE);
                MPI_Send(myArray, number_of_elements_to_send, MPI_INT, 0, 0,
                MPI_COMM_WORLD);
            } // end of for-loop
        }
    }

    // Finalize MPI
    MPI_Finalize();

    return 0;
```

```
}
```

For the bonus task, the following code was used:

```c
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

// Maximum array size 2^20= 1048576 elements
#define MAX_EXPONENT 20
#define MAX_ARRAY_SIZE (1<<MAX_EXPONENT)
#define SAMPLE_COUNT 1000

int main(int argc, char **argv)
{
    // Variables for the process rank and number of processes
    int myRank, numProcs, i;
    MPI_Status status;

    // Initialize MPI, find out MPI communicator size and process rank
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myRank);


    int *myArray = (int *)malloc(sizeof(int)*MAX_ARRAY_SIZE);
    if (myArray == NULL)
    {
        printf("Not enough memory\n");
        exit(1);
    }
    // Initialize myArray
    for (i=0; i<MAX_ARRAY_SIZE; i++)
        myArray[i]=1;

    int number_of_elements_to_send;
    int number_of_elements_received;

    // PART C
    if (numProcs < 2)
    {
        printf("Error: Run the program with at least 2 MPI tasks!\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }
    double startTime, endTime;

    // TODO: Use a loop to vary the message size
    for (size_t j = 0; j <= MAX_EXPONENT; j++)
    {
        number_of_elements_to_send = 1<<j;
        if (myRank == 0)
        {
            myArray[0]=myArray[1]+1; // activate in cache (avoids possible delay when sending the 1st element)
            startTime = MPI_Wtime();
            for (i=0; i<SAMPLE_COUNT; i++)
            {
                MPI_Sendrecv(myArray, number_of_elements_to_send, MPI_INT, 1,0,myArray, number_of_elements_to_send, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
            }

            endTime = MPI_Wtime();
            printf("Rank %2.1i: Received %i elements: Ping Pong took %f seconds\n", myRank, number_of_elements_to_send,(endTime - startTime)/(2*SAMPLE_COUNT));
        }
        else if (myRank == 1)
        {
            for (i=0; i<SAMPLE_COUNT; i++)
            {
                MPI_Sendrecv(myArray, number_of_elements_to_send, MPI_INT, 0,0,myArray, number_of_elements_to_send, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
            }
        }
    }

    // Finalize MPI
    MPI_Finalize();

    return 0;
}
```