

Exercise 1 - MPI HelloWorld

1. Write the code in C.

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

int main(int argc, char *argv[]) {
    int rank, size, provided;
    MPI_Init_thread(&argc, &argv, MPI_THREAD_SINGLE, &provided);

    // Get rank ID
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    // Get number of MPI processes
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello World from rank %d from %d processes!\n", rank, size);

    MPI_Finalize();
    return 0;
}
```

2. How do you compile it, which compiler and flags have you used if any?

To compile on Dardel we use the following command: `cc HelloWorld.c -o HelloWorld` No extra compiler flags are needed as `cray-mpich` is loaded by default on Dardel.

To compile on a local machine we can use

```
mpicc HelloWorld.c -o HelloWorld
```

Here we use `mpicc` which is a compiler wrapper for MPI to make compilation easier. On my local machine I could equivalently use

```
gcc HelloWorld.c -o HelloWorld -lmpi
```

to compile.

3. How do you run the MPI code on Dardel?

To run MPI code on Dardel we simply use the command `srun -n 4 ./HelloWorld`, which will run 4 processes.

4. How do you change the number of MPI processes?

To change the number of processes we change the `-n`, which determines the number of processes that will be launched for the job.

5. Which functions do you use for retrieving the rank of an MPI process and the total number of processes?

The rank of a process is given by the `MPI_Comm_rank(MPI_Comm comm, int* rank)` method. Where `comm` is the MPI communicator, often `MPI_COMM_WORLD`, and the rank ID will be assigned to the given `rank` parameter.

The total number of processes is obtained similarly to above. We use the `MPI_Comm_size(MPI_Comm comm, int* size)` command. Where the total number of processes will be returned to the given `size` parameter.

6. What are the names of the most used MPI implementations?

The most commonly use implementations of MPI are:

- MPICH
- OpenMPI

Exercise 2 - Measure Network Bandwidth and Latency on Dardel with Ping-Pong

2. Using best fit (using Matlab, Python, or similar), calculate the bandwidth and latency for 1) and 2).

Below are the results averaged across 5 runs of the ping-pong benchmark.

Processes	Intra	Inter
Time (us)	190.32859	3.59037
Bandwidth (GB/s)	16.79437	23.94768

Note: These results are unexpected, it would be expected that the intra-node communication to be faster than inter-node communication.

3. Why the postal model is not the best performance model for communication?

The Postal model is not the best for modelling performance due to a number of simplifications the model makes about the network and node communication. One simplification it makes is to measure the communication rate for a single process, in reality, the network cannot sustain this rate of communication, giving skewed results.

Another simplification is how the model ignores the bandwidth limits at the interfaces of the nodes. This means in applications where bandwidth is a limiting factor the results will be inaccurate.

It also doesn't account for the different communication methods used for messages of different lengths, such as eager and rendezvous.

4. How you would you improve the ping pong test after have read the paper?

There are a number of ways you could improve the ping pong benchmark. Two which were already mentioned in the paper were to recognise the eager/rendezvous threshold for sending messages of different sizes, and to take the bandwidth of the interface between the nodes.

The methods mentioned in the paper, being

- Accounting for the eager/rendezvous threshold, and
- Taking the bandwidth limits of the interfaces of nodes

are a good step to improving the accuracy of the ping pong bench mark, one could increase the accuracy of the model by taking contention and congestion of multi-processes running on a single node which have shared resources and cache.

Exercise 3 - 1D Domain Decomposition with Blocking Communication

1. Assume periodic boundary conditions, e.g. the first and last process will communicate. Implement the communication for the 1D domain decomposition using

- MPI blocking point-to-point communication.
- MPI non-blocking point-to-point communication.

2. Test the results by checking the correct values are on the ghost cells and the derivative of $\sin(x)$ on the edges of the domain is correct (the derivative of $\sin(x)$ is $\cos(x)$). Show that your code produce the correct results (especially at the ghost cells).

3. Why MPI_Send and MPI_Recv are called “blocking” communication?

Exercise 4 - Calculate PI with MPI

4.1 Collective reduce

Processes	8	16	32	64	128	256	384
Seconds	3.510213	1.788219	0.901785	0.75959	0.42112	0.140007	0.142021

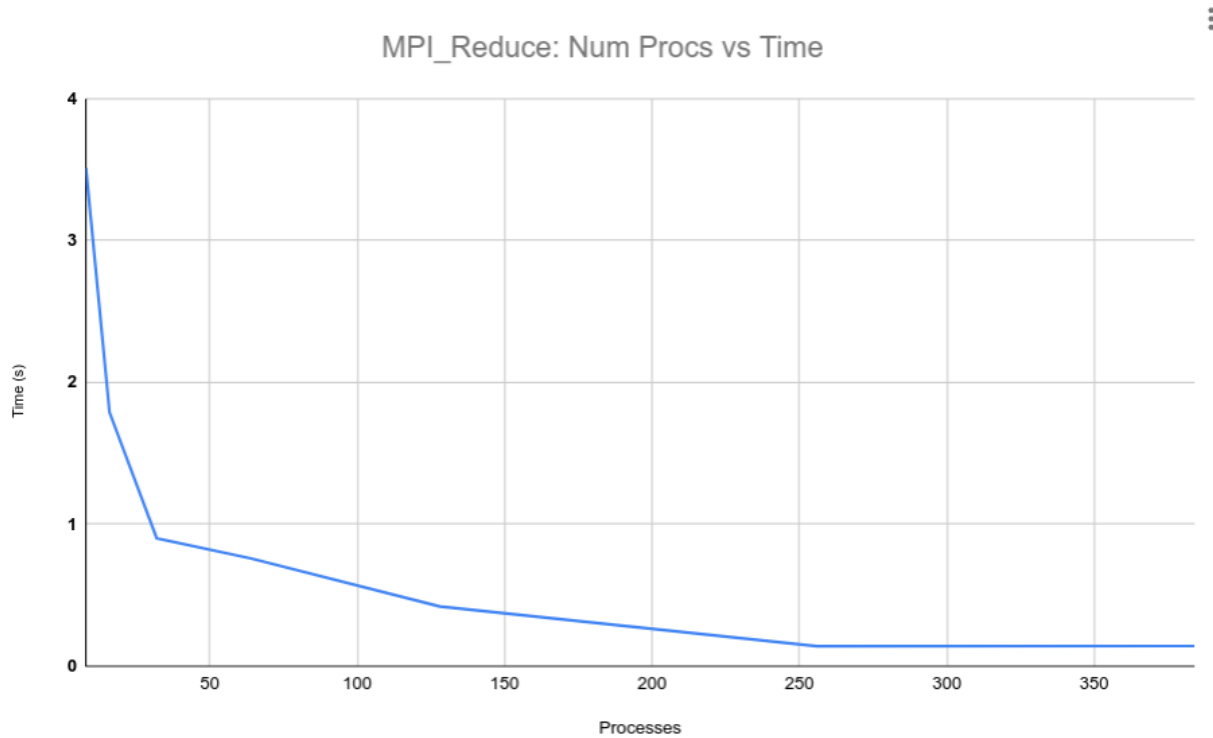


Figure 1: graph

As can be seen in the above graph, the performance improves with increasing the number of Processes up to 256 where the performance plateaus.