

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 -

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

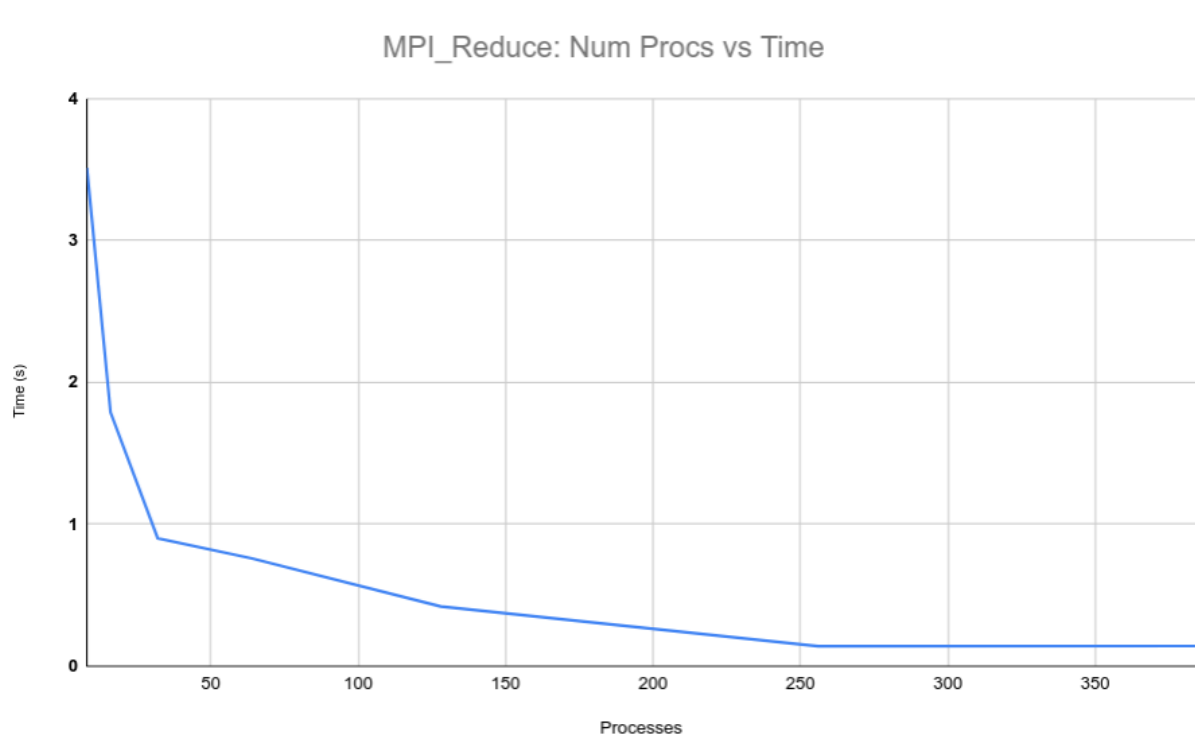


Figure 1: graph