1. Informations about computer system used:

CPU: Intel® Core™ i5-8300H

NUMBER OF PHYSICAL CORRES: 4

NUMBER OF LOGICAL CORRES: 8

CLOCK RATE: 2300 - 4000 MHz

CPU CACHE: 8 MB

2. Source code:

TASK 2

```
#include <stdio.h>
#include <math.h>
#include <mpi.h>
#include <iostream>
#include <Windows.h>

//SUB-TASK 1
void point_to_point(int argc, char* argv[])
{
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    int out_number = 1;
    int in_number;

    if (rank == 0)
    {
        MPI_Send(&out_number, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        MPI_Recv(&in_number, 1, MPI_INT, MPI_ANY_SOURCE, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        std::cout << "Process: " << rank << " send: " << out_number << " and receive: "<< in_number
<< std::endl;
    }
    else if(rank == 1)
    {
        MPI_Recv(&in_number, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Send(&out_number, 1, MPI_INT, 0, 1, MPI_COMM_WORLD);
        std::cout << "Process: " << rank << " send: " << out_number << " and receive: " << in_number
<< std::endl;
    }
}

//SUB-TASK 2
void non_blocking(int argc, char* argv[])
{
    int rank;
    int size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    MPI_Request request;
    MPI_Status  status;

    int request_finished;
    int number = 1;

    if (rank == 0)
    {
        for (int i = 1; i < size; i++)
        {
            number = number * i;
            MPI_Isend(&number, 1, MPI_INT, i, 0, MPI_COMM_WORLD, &request);
            MPI_Wait(&request, &status);
            MPI_Test(&request, &request_finished, &status);
            std::cout << "Process: " << rank << " send: " << number << std::endl;
        }
    }
    else
    {
        MPI_Irecv(&number, 1, MPI_INT, MPI_ANY_SOURCE, 0, MPI_COMM_WORLD, &request);
        MPI_Wait(&request, &status);
        std::cout << "Process: " << rank << " receive: " << number << std::endl;
    }
    MPI_Barrier(MPI_COMM_WORLD);
}

int main(int argc, char** argv)
{
    MPI_Init(&argc, &argv);
    point_to_point(argc, argv);
    non_blocking(argc, argv);
    MPI_Finalize();
    return 0;
}
```

Command:

```
mpiexec -n 2 ThirdLaboratory.exe
```

Example output:

```
Process: 1 send: 1 and receive: 1
Process: 0 send: 1 and receive: 1
Process: 1 receive: 1
Process: 0 send: 1
```

TASK 3

```
#include "mpi.h"
#include <math.h>
#include <iostream>
#include <chrono>

int is_prime(int nr)
{
    if (nr < 2)
        return 0;
    for (int i = 2; i <= sqrt(int(nr)); i++)
        if ((nr % i) == 0)
                return 0;
    return 1;
}

void prime_numbers(int argc, char* argv[],int number)
{
    MPI_Init(&argc, &argv);

    int size,rank,start,step,result,tmp_result;
    float start_time, end_time;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    start = (rank * 2) + 1;
    step = size * 2;
    tmp_result = 0;

    start_time = MPI_Wtime();
    if (rank == 0)
    {
        for (int i = start; i <= number; i = i + step)
        {
            if (is_prime(i))
            {
                tmp_result++;
            }
        }
        MPI_Reduce(&tmp_result, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
        result++;
        std::cout << "In given number there is: " << result << " prime numbers" << std::endl;;
        end_time = MPI_Wtime();
        std::cout << "Execution time: " << end_time-start_time << std::endl;;
    }

    if (rank > 0)
    {
        for (int i = start ; i <= number; i = i + step)
        {
            if (is_prime(i))
            {
                tmp_result++;
            }
        }
        MPI_Reduce(&tmp_result, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
    }
    MPI_Barrier(MPI_COMM_WORLD);

    MPI_Finalize();
}


int main(int argc, char* argv[])
{
    int number = 1000000;
    prime_numbers(argc, argv, number);
    return 0;
}
```

Command:

```
mpiexec -n 2 ThirdLaboratory.exe
```

Example output:

```
In given number there is: 78498 prime numbers
Execution time: 0.882813
```

TIME REPORT

Number to check: 100000

Obtained value:	**9592**	**9592**	**9592**	**9592**	**9592**	**9592**
Number of processes:	**1**	**2**	**4**	**8**	**16**	**32**
Time [s]:	**0.107422**	**0.0449219**	**0. 0.03125**	**0.0195313**	**0.00976563**	**0.00976563**

Number to check: 1000000

Obtained value:	**78498**	**78498**	**78498**	**78498**	**78498**	**78498**
Number of processes:	**1**	**2**	**4**	**8**	**16**	**32**
Time [s]:	**1.77148**	**0.882813**	**0.515625**	**0.361328**	**0.277344**	**0.189453**

Number to check: 10000000

Obtained value:	**664579**	**664579**	**664579**	**664579**	**664579**	**664579**
Number of processes:	**1**	**2**	**4**	**8**	**16**	**32**
Time [s]:	**41.9434**	**20.6133**	**12.1602**	**8.98242**	**8.22266**	**8.06055**

3. **Briefly comment for task 3.**

- The presented solution allows to search for prime numbers starting from different numbers depending on the number of processes used.
- Individual processes breaks given number into smaller chunks which allows for a parallel search for a solution.
- Gradual increase in the number of processes used to perform the task, significantly reduces overall program execution time.
- The biggest increase in performance was caused by use of more than one process