MPI: Practical session 2

1 Deadlock

Compile and run the following code on two processes (Fortran: see `deadlock.f08` on Moodle):

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

#define SIZE 10240

int main(){
        int rank;
        void *ptr;

        MPI_Init(NULL, NULL);
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);

        ptr=malloc(SIZE);

        if (rank == 0){
                MPI_Send(ptr, SIZE, MPI_CHAR, 1, 1, MPI_COMM_WORLD);
                MPI_Recv(ptr, SIZE, MPI_CHAR, 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
        if (rank == 1){
                MPI_Send(ptr, SIZE, MPI_CHAR, 0, 1, MPI_COMM_WORLD);
                MPI_Recv(ptr, SIZE, MPI_CHAR, 0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }

        std::cout << "Rank " << rank << " finished" << std::endl;
        MPI_Finalize();
        return 0;
}
```

Does the code complete? If it fails to complete, does it sit using no CPU time, or using CPU time? Next try decreasing the value of `SIZE` and retrying the same run. Does it give the same behaviour?

2 Send/Receive

Write a program that does the following:

- On process 0, initialize an integer v with value 0
- Send v from process 0 to process 1, and on process 1, add 1. Display the value of v.
- ...
- Send v from process n to process $n+1$, and on process $n+1$, add n. Display the value of v.

- ...
- Finally send this integer back to process zero and print out its final value.

The final value of v should be $\frac{N(N-1)}{2}$. Check that this is the case. Of course, this is a very contrived example, and would not be useful in practical code because of the inherent serialization that results.

3 Primes calculation

This question asks you to write a parallel program that computes the primes between 1 and 10000000, in two ways. In both versions, the prime numbers, as well as the number of primes found by each process should be sent to rank 0, who will eventually print the total number of primes found.

(To determine whether a number is prime, do not try anything fancy; just check all possible odd divisors less than $\sqrt{N}$)

3.1 Version 1: Static workload

First, divide the workload (approximately) equally between the processes. Hint: Use `MPI_Get_count` to determine the number of primes that each process has computed in each range.

3.2 Version 2: Dynamic workload

Now write a *master-slave* program that computes the number of primes in sets of 100 integers.
Note: In a master-slave program, the master does not do any computation. It simply sends tasks to workers and receives their results. Every time a worker sends their result back to the master, they are assigned a new task.