MPI: Practical session 3

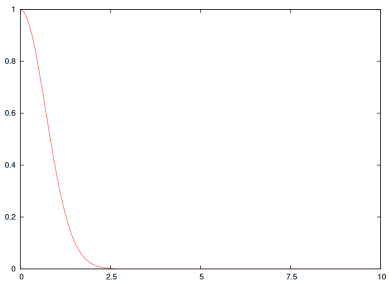
1 Gaussian quadrature

The code in `quadrature.cpp` attempts to evaluate

$$\int_0^\infty \exp\left(-x^2\right) dx = 0.5\sqrt{\pi}$$

by approximating it by integral

$$\int_0^{10} \exp\left(-x^2\right) dx.$$



It does so by using Gaussian quadrature. This code is serial. In this practical, we will use various different approaches to parallelise it.

1.1 Reduction operator

First, divide the interval $[0, 10]$ approximately equally between the processes and use an MPI reduction operation to compute the total sum.

When you are convinced that your parallel program works, increase the number of steps by a factor of 1,000 (to make it take a reasonable amount of time) and time the program:

```
$ time mpiexec -n 16 ./quadrature_reduce
```

1.1.1 Speed

Compare the speed of the program as run on 1 core with that run on 2, 4, 6, 8, 10, 12, and 16 (or more) cores. Do you attain a factor of 16 speed-up by running on 16 cores? If not, why not?

1.1.2 Accuracy

Is the final result identical regardless of the number of processors the code is run on? If you find yourself getting different results, consider why this might be happening. Is this a problem?

1.1.3 Modifications

- Try changing the code so that the result is made available on process `N - 1`, where `N` is the number of processes.
- Change the code so that the result is available on all processes simultaneously
- What happens if you replace `MPI_DOUBLE` by `MPI_FLOAT`, `MPI_INTEGER`, or `MPI_LONG`? What does this tell you about type-checking in MPI?

1.2 Gather operator

Modify your program to use `MPI_Gather` instead of `MPI_Reduce`. The final sum would therefore be calculated by a single process.

1.3 Send/Receive

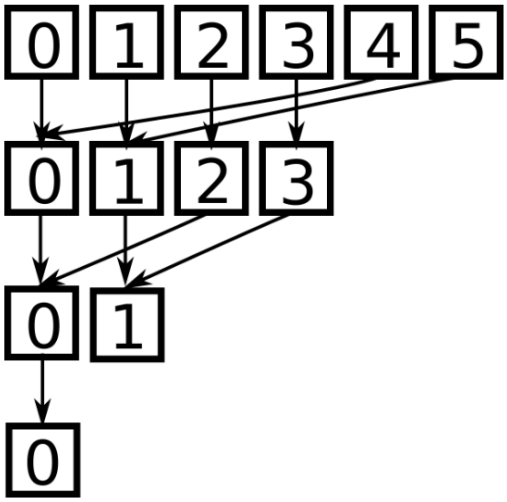
Now use point-to-point communication (`MPI_Send` and `MPI_Recv`), instead of collective communication, to send each process' result to process 0 and perform the sum on process 0.

1.4 Extra: Tree-based sum

Now implement this using a tree-based sum:
Compute $p = \lfloor \log_2 N \rfloor$.

- On process n find the sum of the total from process n and process $n + 2^p$
- On process n find the sum of the total from process n and process $n + 2^{p-1}$
- Repeat until the final total is on process 0 and print it out.

Illustrated below is the tree diagram for 6 processes, where the arrows denote which pairs of processors values are added together:



Which of these three approaches do you think is more efficient? If it helps, imagine different computers with wildly different costs for communication between processes and for performing the addition.

This thought experiment (and the effort involved in getting the tree reduction correct) suggests that you should always use MPI's collective operations where possible. The MPI implementation writers will have spent much more time optimizing the collective operations for various numbers of processors, interconnect, etc. than you are able to.