MPI: Practical session 4

1 Scatter and Gather

Write a program that carries out the following operations:

- Given an MPI program running on N processors, generate an $N \times N$ matrix on process zero, with integer elements $A_{i,j} = i \times N + j$. This should use a contiguous block of memory.
- Use `MPI_Scatter` to distribute each row of the matrix to its own process (e.g. $A_{p,i}$ for $0 \leq i < N$ will end up on process p).
- On each process, sum the elements in the row and print this sum to the screen.
- Gather the sum from each process back onto rank 0.
- Print out the total sum of elements on rank 0 to the screen.

2 Transpose

This example may prove somewhat mind-taxing; it requires the ability to think carefully about what data is held by what process.

Note that `MPI_Scatter` permits elements with various offsets to be scattered to each process.

Write a program that uses `MPI_Scatter` exactly N times and `MPI_Gather` exactly once to transpose a matrix of size $N \times N$ (initially stored on process 0), where N is the number of processors on which the program is run.

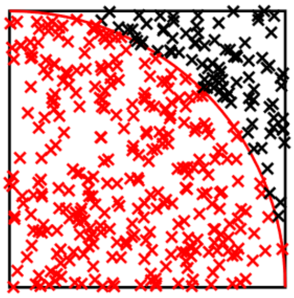
Hint: Each process should (temporarily) hold one row of the matrix.

You are strongly encouraged to implement careful checking, to ensure that the correct result is obtained.

2.1 Extension

Re-write the program, now replacing the N calls to `MPI_Scatter` by one `MPI_Scatter` and one `MPI_AllToAll`.

3 Calculation of π



$\pi \approx 3.09$

In this example, you are going to estimate $\pi/4$ using a Monte Carlo technique. The serial code `montecarlo_pi.cpp` picks `COUNT` points at random in the unit square $0 \leq x, y \leq 1$ and sees what fraction lies within unit distance of the origin - which, in theory, should be $\pi/4$.

Parallelise the program, so that each process picks `COUNT` random points.

Hint: the `rand` function will give the same random points in each process, ending up precisely the same answer in all processes. You should explicitly set the random number seed to different values on each process.

3.1 Extension

One might reasonably wish to perform quite a large number of iterations in this program for calculating $\pi/4$, as the iterations are fast and convergence is slow. Can you write something which will run with more than 2^{32} iterations, remembering that the total iteration count will no longer fit into a default 32-bit integer?

In C and C++ one can use the `MPI_LONG` datatype in MPI calls, and this is probably 64 bits. In Fortran there is no simple MPI call for sending non-default integers. One solution is to remember that a double precision variable can reliably store integers up to about 2^{53}, so one can convert Fortran's longer integers to doubles before sending them without losing accuracy provided that they are less than that.

In Fortran, `integer(selected_int_kind(12)) :: count` gives an integer capable of storing integers in the range $\pm 10^{12}$. This cannot be a 32 bit integer, so will probably be a 64 bit one. Do your results actually keep improving with larger iteration counts, or do you suffer from a random number source with a short periodicity?