Distributed Memory Programming with MPI
Programming Assignment – 1

Exercise – 1 [2 points]

Suppose we toss darts randomly at a square dartboard, whose bullseye is at the origin, and whose sides are 0.6m in length. Suppose also that there's a circle inscribed in the square dartboard. The radius of the circle is 0.3m, and it's area is 0.093π square meters. If the points that are hit by the darts are uniformly distributed (and we always hit the square), then the number of darts that hit inside the circle should approximately satisfy the equation:

$$\text{number in circle / total number of tosses} = \pi/4$$

since the ratio of the area of the circle of the square is π/4. We can use this formula to estimate the value π with a random number generator:

```
number_in_circle = 0;
for (toss = 0; toss < number_of_tosses; toss++) {
x = random double between −1 and 1;
y = random double between −1 and 1;
distance_squared = x * x + y * y;
if (distance_squared <= 1) number_in_circle++;
}
pi_estimate = 4 * number_in_circle/((double) number_of_tosses);
```

This is called a "Monte Carlo" method, since it uses randomness (the dart tosses). Write an MPI program that uses a Monte Carlo method to estimate π. Process 0 should read in the total number of tosses and broadcast it to the other processes. Use `MPI_Reduce` to find the global sum of the local variable `number_in_circle`, and have process 0 print the result. You may want to use **long long int**s for the number of hits in the circle and the number of tosses, since both may have to be very large to get a reasonable estimate of π. Demonstrate with an example the correct operation of your programme.

Exercise – 2 [4 points]

Write two MPI programs that perform the computation of a global sum; the first using tree-structured computation and the second using a butterfly structure. First, write your programs for the special case in which `comm_sz` is a power of two. Subsequently, modify your program so that it will handle any number of processes. Identify and run examples that demonstrate the correct implementation of your code and comment on the results.

Exercise – 3 [4 points]

A **ping-pong** is a communication in which two messages are sent, first from process A to process B (ping) and then from process B back to process A (pong). Timing blocks of repeated ping-pongs is a common way to estimate the cost of sending messages. Time the ping-pong program using the C `clock` function on your system.

a) How long does the code have to run before `clock` gives a non-zero run-time? How do the times you got with the `clock` function compare to times taken with `MPI_Wtime`? [2 points]

b) What happens on your system when the `count` argument is 0? Can you explain why you get a non-zero elapsed time when you send a zero-byte message?

Identify and run appropriate examples to support your answers.