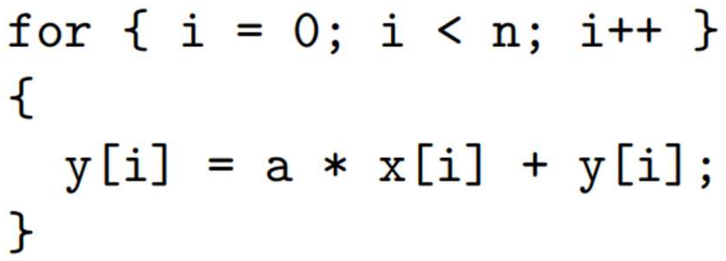
Tutorial 0

1. You are given an array `shared_array` of size `ARRAY_SIZE`. Multiple threads need to concurrently increment each element of the array.
2. Design a parallel program using OpenMP to achieve this, ensuring that the increments are performed atomically to avoid race conditions.

Tutorial Exercise 1

- The SAXPY program is to add a scalar multiple of a real vector to another real vector:
- s = a*x + y.
- Provided a serial SAXPY code, parallelize it using OpenMP directives.
- Compare the performance between serial and OpenMP codes.



```
for { i = 0; i < n; i++ }
{
  y[i] = a * x[i] + y[i];
}
```

- Check total wall clock execution time versus thread numbers:
  - export OMP_NUM_THREADS=1
  - time ./saxpy
  - export OMP_NUM_THREADS=2
  - time ./saxpy
  - export OMP_NUM_THREADS=4
  - time ./saxpy
  - export OMP_NUM_THREADS=8
  - time ./saxpy

Tutorial Exercise 2

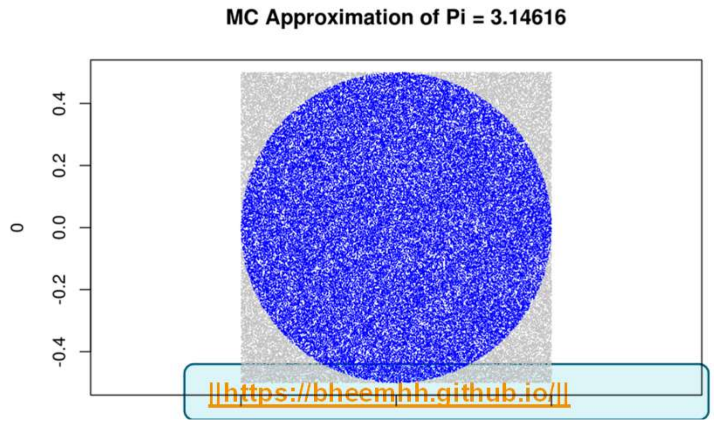
1. **Estimating the value of Pi using Monte Carlo**
    1. computational algorithms that rely on repeated random sampling to obtain numerical results

Estimation of Pi

We know that area of the square is 4r^2 unit sq while that of circle is pi*r^2.

The ratio of these two areas is as follows : pi/4

1. Generate a random point (x, y) inside a square of side 2 centered at the origin.
2. Determine whether the point falls inside the unit circle inscribed in the square by checking whether $x^2 + y^2 <= 1$.
3. Repeat steps 1 and 2 for a large number of points (e.g., 10^7).
4. Calculate the ratio of the number of points that fell inside the circle to the total number of points generated.
5. Multiply the ratio by 4 to estimate the value of pi.



Monte Carlo to estimate PI

```
#include <stdlib.h>
#include <stdio.h>
#include "omp.h"
int main(int argc, char *argv[])
{
 long int i, count; // count points
inside unit circle
 long int samples; // number of samples
 double pi;
 unsigned short xi[3] = {1, 5, 177}; //
random number seed
 double x, y;
 samples = atoi(argv[1]);
 count = 0;
 for(i = 0; i < samples; i++)
 {
 x = erand48(xi);
 y = erand48(xi);
 if(x*x + y*y <= 1.0) count++;
 }
 pi = 4.0*count/samples;
 printf("Estimate of pi: %7.5f\n", pi);
}
```

- Provided a serial code, parallelize it using OpenMP directives.
- Compare the performance between serial and OpenMP codes