ASSIGNMENT NO 2 APPROXIMATING VALUE OF PI

Entry no: 2021MCS2132
Name: Gaurav Milind Chaudhari
Email: mcs212132@iitd.ac.in

For this assignment I checked various methods of approximating values of pi
Methods mostly classified into 3 types
1 .series method (infinite series sum methods)
2 .Integral Method (approximating pi using infinite sum of area under curve), 3 3.
3. random sampling method (monte calro method)
After trying various method some are easy to make parallel programs but some are tricky I decided two methods for this assignment which are

1 .infinite sum integral method

(can be converted into parallel programme with higher speedup)

2.series expansion using wallis series

(as most of the series are similar to the maclaurin series which provided in assignment while series lille viete is difficult to convert into parallel program understood with many failed attempts as difficult to distribute parallel work among thread I decided to go with Wallis method)

1 Integral Method of approximating pi

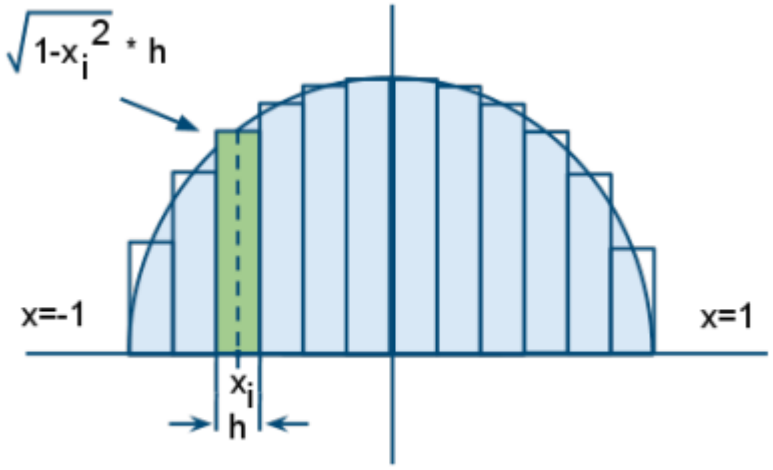

Figure 2: Approximating area under a curve using rectangles.

This figure shows rectangles of same length h that approximately cover the semicircular area. Each rectangle is positioned in a interval of the x-axis interval [-1,1], and the height of a rectangle is the function's value at some value x_i in that rectangle's subinterval. Thus, the area of a rectangle is $\sqrt{1-x_i^2} * h$. We must add up the areas of all these rectangles, then double that sum to get the approximation of π.
So we our approximation increases with no. of iteration increases .this is method to calculate pi in approximate area .

ASSIGNMENT NO 2 APPROXIMATING VALUE OF PI

Thread count vs output

Thread count 1

```
pi calculated = 3.141592653277105
time taken    = 0.107955
parellel execution
pi calculated = 3.141592653455990
time taken    = 0.111766
speed up 0.965903
```

Thread count 2

```
pi calculated = 3.141592648907237
time taken    = 0.010568
parellel execution
pi calculated = 3.141592654563925
time taken    = 0.006066
speed up 1.742237
```

Thread count 3

```
pi calculated = 3.141592653277105
time taken    = 0.115013
parellel execution
pi calculated = 3.141592653456526
time taken    = 0.042174
speed up 2.727119
```

Thread count 4

```
pi calculated = 3.141592653277105
time taken    = 0.108356
parellel execution
pi calculated = 3.141592653456541
time taken    = 0.035857
speed up 3.021896
```

Thread count 5

```
serial execution
pi calculated = 3.141592653277105
time taken    = 0.115264
parellel execution
pi calculated = 3.141592653456573
time taken    = 0.037516
speed up 3.072385
```

ASSIGNMENT NO 2 APPROXIMATING VALUE OF PI

As running same our algorithm with varying thread count we observed that speedup increases till thread count equals to 4 and stays nearly same after increasing number of threads

2 WALLIS SERIES

$$\frac{\pi}{2} = \prod_{n=1}^{\infty} \frac{4n^2}{4n^2-1} = \prod_{n=1}^{\infty} \left(\frac{2n}{2n-1} \cdot \frac{2n}{2n+1} \right)$$
$$= \left(\frac{2}{1} \cdot \frac{2}{3}\right) \cdot \left(\frac{4}{3} \cdot \frac{4}{5}\right) \cdot \left(\frac{6}{5} \cdot \frac{6}{7}\right) \cdot \left(\frac{8}{7} \cdot \frac{8}{9}\right) \cdot \cdots$$

Wallis series is infinite product series which gives approximate estimate pi to the very accurate digits
Serial code of the series can be converted easily as code given below

```
double serial_wallis(double n)
{
    double numerator=1,denominator=1, result=1;

   for (double i=1;i<=n;i++)
   {
        numerator=(4*i*i);
        denominator=((2*i)-1)*((2*i)+1);
        result *=(numerator/denominator);

   }
   return result*2;
}
```

Thread count 2

```
parellel execution PI=3.141592575061526
TIme taken = 0.027091
serial execution PI=3.141592575080853
TIme taken = 0.042392
speed up 1.564799
```

ASSIGNMENT NO 2 APPROXIMATING VALUE OF PI

Thread count 3

```
parellel execution PI=3.141592575031861
TIme taken = 0.027303
serial execution PI=3.141592575080853
TIme taken = 0.050561
speed up 1.851847
```

Thread count 4

```
parellel execution PI=3.141592575046205
TIme taken = 0.023584
serial execution PI=3.141592575080853
TIme taken = 0.040705
speed up 1.725947
```

Thread count 5

```
parellel execution PI=3.141592575047300
TIme taken = 0.031970
serial execution PI=3.141592575080853
TIme taken = 0.059268
speed up 1.853875
```

Thread count 6

```
parellel execution PI=3.141592575045384
TIme taken = 0.023932
serial execution PI=3.141592575080853
TIme taken = 0.043270
speed up 1.808036
```

As opposed to previous algorithm this algorithm doesn't scales up well and speed up remains constant even with increase in the number of threads