ЛАБОРАТОРНА РОБОТА 5

OpenMP. Обчислення констант

Завдання.

1. Написати програму, що обчислює число π використовуючи технології OpenMP. Програму треба спочатку реалізувати послідовно, а потів розпаралелити 1-3 участки коду.

Директива ***parallel***

Паралельна область задається за допомогою директиви ***parallel***

```
#pragma omp parallel [опция[[,] опция]...]
```

- **Private (**список**)-** задає список змінних, для яких робиться локальна копія для кожної нитки.
- **Shared(**список**)-** Задає список змінних, що спільний для всіх ниток.
- **Reduction**(оператор:список) – задає оператори та список спільних змінних, для кожної змінної створюється локальна копія в кожній нитці. Локальні копії ініціалізуються згідно типу оператору (адитивний-0, мультиплікативний-1).

```
#include <stdio.h>
double f(double y) {return(4.0/(1.0+y*y));}
int main()
{
   double w, x, sum, pi;
   int i;
   int n = 1000000;
   w = 1.0/n;
   sum = 0.0;
#pragma omp parallel for private(x) shared(w)\
            reduction(+:sum)
   for(i=0; i < n; i++)
   {
      x = w*(i-0.5);
```

```
        sum = sum + f(x);
    }
    pi = w*sum;
    printf("pi = %f\n", pi);
}
```

2. Написати програму, що обчислює число π методом Монте Карло та розпаралелити її за допомогою OpenMP.

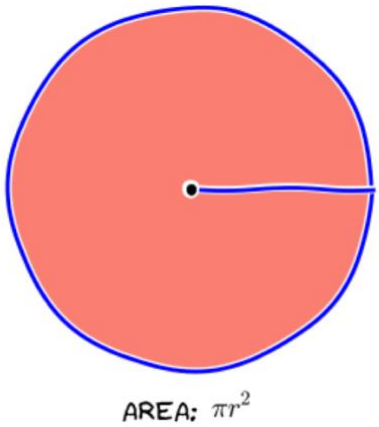
Метод Монте Карло для обчислення числа Пі

Методи Монте-Карло - загальна назва групи числових методів, заснованих на одержанні великої кількості реалізацій стохастичного (випадкового) процесу, який формується у той спосіб, щоб його ймовірнісні характеристики збігалися з аналогічними величинами задачі, яку потрібно розв'язати.

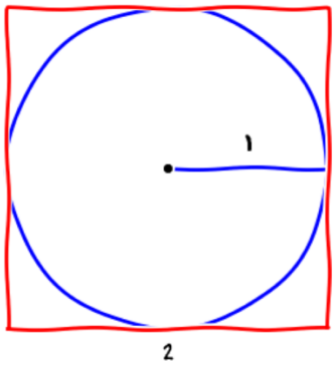
Наша мета - ознайомитись із методом Монте-Карло для обчислення числа Pi (π). Спочатку ми реалізуємо послідовну версію методу. А потім ми будемо використовувати OpenMP для паралелізації послідовної версії.

Обчислення Пі

Якщо ви не знаєте, що таке π, це число, яке має щось спільне з колами. Точніше, площа диска радіусом r дорівнює $\pi * r^2$.



Якщо радіус кола дорівнює одиниці, то площа диска дорівнює π.
Далі впишемо коло в квадрат. Якщо довжина сторони квадрата дорівнює двом, то радіус вписаного кола дорівнює одиниці.

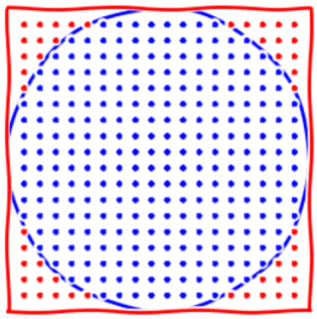


Тепер ми ділимо площу кола на площу квадрата:

$$\frac{\text{AREA OF THE CIRCLE}}{\text{AREA OF THE SQUARE}} = \frac{\pi}{4}$$

Якщо помножити цей дріб на чотири, ми отримаємо значення π.

Простий спосіб обчислення дробу - це генерування точок у квадраті та підрахунок кількості точок, що лежать у внутрішній частині кола.



Наближення π дорівнює

$$\pi = 4\frac{\text{NUMBER OF POINTS INSIDE THE CIRCLE}}{\text{NUMBER OF ALL POINTS}}$$

Метод Монте-Карло для Пі

Метод Монте-Карло використовує попередню ідею для обчислення значення π. Метод випадковим чином генерує точки в квадраті [-1, 1] x [-1, 1] і підраховує кількість точок усередині одиничного кола.

Результат в чотири рази перевищує частку між кількістю точок, які лежать у внутрішній частині кола, і кількістю всіх сформованих точок.

Впровадження

Рівномірні точки

Першим кроком реалізації є рівномірне генерування випадкових точок. Для такого завдання підходить стандартна бібліотека <random>. Для того, щоб генерувати випадкові точки з бібліотекою, нам потрібно вказати дві речі:

- генератор, який генерує рівномірно розподілені числа і
- розподіл, який перетворює числа з генератора в числа, що слідують за певним випадковим розподілом.

В якості генератора ми використовуємо std :: default_random_engine, а як розподіл - std :: uniform_real_distribution <double>. Ми ініціалізуємо внутрішній стан генератора за допомогою функції std :: default_random_engine :: seed. Нарешті, ми генеруємо рівномірну випадкову вибірку (x, y) в [-1, 1] x [-1, 1].

```
std::default_random_engine generator;
auto seed = std::chrono::system_clock::now().time_since_epoch().count();
generator.seed(seed);
std::uniform_real_distribution<double> distribution(-1.0, 1.0);


double x = distribution(generator);
double y = distribution(generator);
```

Ми інкапсулюємо випадкове формування точок у класі UniformDistribution. Конструктор дбає про ініціалізацію генератора та розподіл. Функція - UniformDistribution :: sample генерує випадкову вибірку в інтервалі [-1, 1]. Тоді наступний код еквівалентний наведеному вище коду.

```
UniformDistribution distribution;


auto x = distribution.sample();

auto y = distribution.sample();
```

Monte Carlo method

```
double approximatePi(const int numSamples)

{

    UniformDistribution distribution;


    int counter = 0;

    for (int s = 0; s != numSamples; s++)

    {

        auto x = distribution.sample();

        auto y = distribution.sample();


        if (x * x + y * y < 1)

        {

            counter++;

        }

    }


    return 4.0 * counter / numSamples;

}
```