Wrocław, dn. 10 czerwca 2015r.

Politechnika Wrocławska

Aplikacje Internetowe i Rozproszone

Obliczanie przybliżenia liczby Pi.

Dokumentacja końcowa

Grupa projektowa:
Paweł Sternik, 200623
Kamil Cichuta, ???
Mariusz Cebula, ???
Sławomir Sygut, ???

Prowadzący:
dr inż. Marek Woda

Termin spotkań:

Środa, godz. 07:30

Spis treści

1 Temat projektu.

Tematem projektu realizowanego w ramach kursu Aplikacje Internetowe i Rozproszone było wyznaczanie rozszerzenia liczby Pi z wykorzystaniem algorytmu Monte Carlo. Realizacja wymagała implementacji kilku modułów stanowiących cały system.

2 Cel projektu.

2.1 Cel dydaktyczny.

Głównym celem dydaktycznym kursu było zapoznanie się z technologiami wykorzystywanymi do tworzenia rozproszonych aplikacji połączonych z internetowymi klientami. Ponadto kurs wymagał zoorganizowania pracy w grupach kilkuosobowych co wymuszało zaplanowanie kolejnych etapów pracy i podział poszczególnych zadań między członków grupy. Wykorzystane zostały do tego odpowiednie narzędzia komunikacji, które zostaną opisane dokładniej w dalszej częsci dokumentu.

2.2 Cel merytoryczny.

Implementacja algorytmu Monte Carlo obliczającego rozszerzenie liczby Pi z wykorzystaniem technologi MPI (ang. Message Passing Interface) czyli protokołu komunikacyjnego służącego do przesyłania komunikatów pomiędzy procesami programów równoległych. Ponadtwo stworzenie aplikacji internetowej która poprzez stworzoną bazę dancyh łączy się z silnikiem obliczeniowym działającym na kilku niezależnych komputerach. Strona powinna posiadać elementy zmieniające się dynamicznie podczas realizacji zadania - przykładowo pasek postępu.

3 Opis zastosowanego algorytmu.

3.1 Opis słowny algorytmu.

Metoda Monte Carlo stosowana jest do problemów, które jest bardzo trudno rozwiązać za pomocą podejścia analitycznego. Najczęściej stosuje się ją do modelowania złożonych problemów takich jak:

1. Obliczanie całek.
2. Obliczanie łańcuchów procesów statystycznych .
3. Obliczanie złożonych symulacji.

Metoda opiera się na losowaniu nazywanym w tym przypadku wyborem przypadkowym. Losowanie dokonywane jest zgodnie z rozkładem, który jest znany. Przykładowo całkowanie metodą Monte-Carlo działa na zasadzie porównywania losowych próbek z wartością funkcji. Dokładność wyniku uzyskanego tą metodą jest zależna od liczby sprawdzeń i jakości użytego generatora liczb pseudolosowych. Zwiększanie liczby prób nie zawsze zwiększa dokładność wyniku, ponieważ generator liczb pseudolosowych ma skończenie wiele liczb losowych w cyklu. Przykładowo całkowanie tą metodą jest używane w przypadkach, kiedy szybkość otrzymania wyniku jest ważniejsza od jego dokładności (np. obliczenia inżynierskie).

W projekcie algorytm Monte Carlo zostanie wykorzystany do obliczenia rozszerzenia dziesiętnego liczby Pi. Dokładny opis realizacji tego algorytmu znajduję się w następnym punkcie.

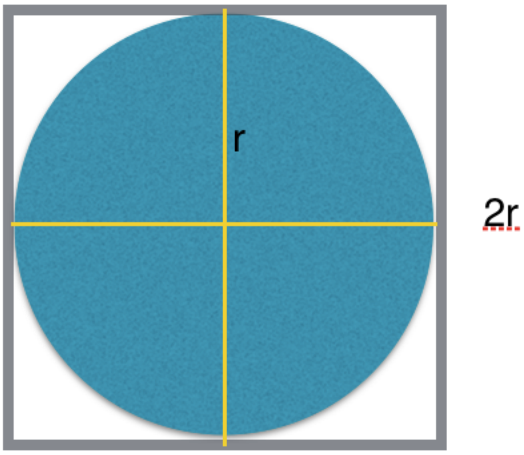
3.2 Obliczanie liczby Pi metodą Monte Carlo.

Liczbę Pi można obliczać na wiele różnych sposobów. Jednym z nich jest wykorzystanie metody Monte Carlo. Metoda ta charakteryzuję się przede wszystkim swoją stosunkową prostą procedurą jak i przystosowaniem do zaimplementowania mechanizmów zrównoleglenia - co jest jednym z głównych celów projektu. Pole kwadratu przedstawionego na Rysunku 1 wynosi:

$$(4\prod)^2$$

natomiast koła oczywiście:

$$(\prod)r^2$$



Rysunek 1: Koło o promieniu r wpisane w kwadrat - bok 2r.

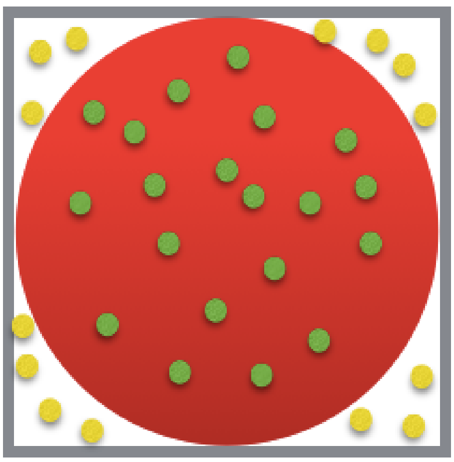
Znając zatem pole koła jak i kwadratu można zauważyć że stosunek pola koła do pola kwadratu wynosi:

$$\frac{PoleKola}{PoleKwadratu} = \frac{\prod r^2}{4r^2} = \frac{\prod}{4}$$

Następnie korzystając z tego, że pole koła jak i kwadratu zostały w jakiś sposób obliczone można wyciągnąć wzór na liczbę Pi:

$$\prod = 4\frac{PoleKola}{PoleKwadratu}$$

Dojście do tego punktu mogłoby się wydawać zatoczeniem koła i powrotem do punktu początkowego problemu. Tak nie jest. Ponieważ dopiero w tym miejscu pojawia się cała charakterystyka metody Monte Carlo. Abstrakcyjnie należy sobię teraz wyobraźić sytuację, w której rzucamy bardzo dużo razy rzutkami w tarczę wyglądająca jak Rysunek 1. Kończąc grę, plansza będzie pokryta rzutkami znajdującymi sie we wnętrzu koła jak i poza nim na obszarze kwadratu. Podsumowując liczba rzutek w i poza kołem będzie równy stosunkowi pola koła do pola kwadratu. Formalnie losowanie będzie odbywać się



Rysunek 2: Przykład losowania punktów.

pośród punktów należących do zbioru pokrytego współrzędnymi należacymi do przedziału [-2r,2r]. Stosunek liczby punktów zawierających się w kole o środku w punkcie ¡0,0¿ i promieniu r do wszystkich wylosowanych punktów będzie dążył w nieskończoności (z pewnym prawdopodobieństwem) do stosunku tego pola koła do koła kwadratu o boku 2r. Co więcej, stosunek ten będzie identyczny również do ćwiartki koła. Jeżeli pole koła podzielimy na cztery i tak samo podzielimy pole kwadratu, to ich stosunek będzie wciąż taki sam. Oznacza to, że wystarczy, jeżeli będziemy losowali punkty o współrzędnych od 0 do r. Cała metoda sprowadza się więc do tego, by losować punkty, sprawdzać, czy mieszczą się w kole, i następnie podstawiać liczby wylosowanych punktów do wzoru. Losując odpowiednio dużo punktów, powinniśmy otrzymać z pewnym prawdopodobieństwem rozsądne przybliżenie liczby Pi.

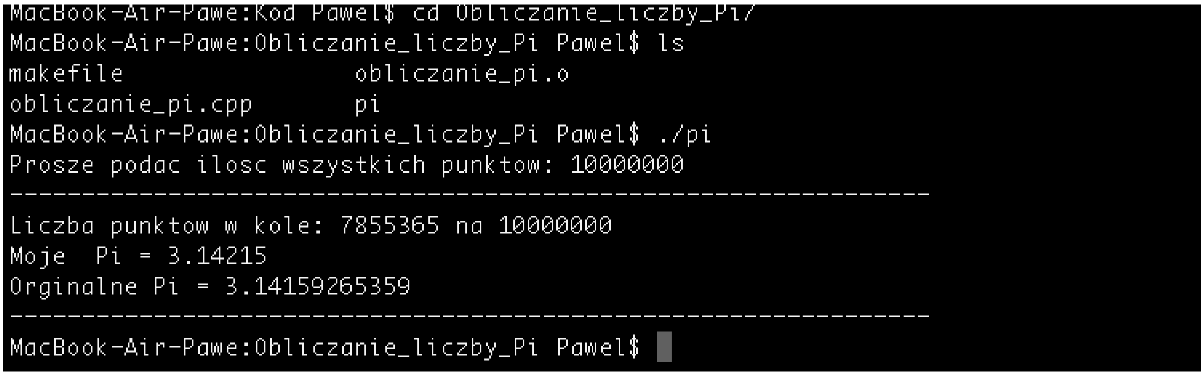
3.3 Implementacja algorytmu - program jednowątkowy.

Dla lepszego zrozumienia działania algorytmu i sprawdzenia jego rezultatów stworzony został program w języku C++ obliczający przybliżenie liczby Pi wykonujący wszystkie obliczenia szeregowo, czyli po prostu korzystający z jednego wątku. Program przyjmuje od użytkownika zadaną liczbę punktów. W rezultacie wyświetla obliczoną liczbę Pi oraz orginalne rozwinięcie.

```
// Obliczanie_liczby_pi.cpp :wersja zwykla -jednowatkowa
// Autor : Pawel Sternik
// Data : 17.03.2015
// Metoda Monte Carlo wyznaczania liczby Pi


#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

using namespace std;

void PobieranieDanych(int &iloscPunktow)
{
        cout << "Prosze podac ilosc wszystkich punktow: ";
        cin >> iloscPunktow;
}

int main( int argc, char * argv[] )
{
        // Delaracja wszystkich zmiennych
        int liczbaWszystkichPunktow, bokKwadratu,
            poleKwadratu, liczbaPunktowKolo = 0;
```

```
        double promienKola;
        double **wszystkiePunkty;
        long double mojePi;
        srand(time(NULL));
        // Pobieranie od uzytkownia liczby punktow
        PobieranieDanych(liczbaWszystkichPunktow);
        bokKwadratu = 1;
        poleKwadratu = 1;
        promienKola = bokKwadratu / 2;
        // Utworzenie macierzy do przechowywania punktow w kwadracie
        wszystkiePunkty = new double*[liczbaWszystkichPunktow];
                for(int i=0; i < liczbaWszystkichPunktow; i++)
                {
                        wszystkiePunkty[i] = new double[2];
                }

        // Losowanie wspolrzednych punktu w kwadracie z zakresu -0,5 do 0,5
                for(int i=0; i < liczbaWszystkichPunktow; i++)
                {
                                for(int j=0; j < 2 ; j++)
                                {
                                        double a = ( rand() % 10000 ) - 5000;
                                        a = a / 10000;
                                        wszystkiePunkty[i][j] = a;
                                }
                }

                for(int i=0; i < liczbaWszystkichPunktow; i++)
                {
                  double potega = pow(wszystkiePunkty[i][0], 2) + pow(wszystkiePunkty[i][1], 2);
                        double odleglosc = sqrt(potega);
                          if(odleglosc <= 0.5)
                                {
                                  liczbaPunktowKolo++;
                                }
                }

  // "WYSWIETLENIE REZULTATOW DZIALANIA PROGRAMU - POROWNANIE"
  cout << "
  ----------------------------------------------------------------------\n";
  cout << "Liczba punktow w kole: " << liczbaPunktowKolo
        <<" na " << liczbaWszystkichPunktow << "\n";
   mojePi = ((double)liczbaPunktowKolo/(double)liczbaWszystkichPunktow) * 4.0;
   cout << "Moje  Pi = " << mojePi << "\n";
   cout << "Orginalne Pi = 3.14159265359\n";
```

```
  cout << "
  ----------------------------------------------------------------\n";

        // Zwolnienie pamieci zaalokowanej przez macierz
    for(int i=0; i < liczbaWszystkichPunktow; i++)
        {
            delete [] wszystkiePunkty[i];
        }

    delete [] wszystkiePunkty;
    return 0;
}
```



Rysunek 3: Przykład działania programu jednowątkowego.

Jak widać na zamieszczonym zdjęciu ekranu program jednowątkowy obliczający rozwinięcie dziesiętne liczby Pi korzystając z metody Monte Carlo obliczył liczbę Pi równą 3.14215. Otrzymana dokładność sięga jedynie dwóch miejsc po przecinku przy stosunkowo już dużej ilości losowanych punktów - 10 mln. Program wyświetlił informację o tym, że 7855365 punktów znalazło się w kole.

4 Zastosowane technologie.

4.1 Framework Django.

4.2 Technologia MPI.

4.3 CSS i HTML.

4.4 Komunikacja grupy.

4.4.1 GitHub.

4.4.2 Trello.

5 Plan realizacji.

5.1 Podział pracy między członków grupy.

5.2 Terminarz realizacji zadań.

6 Implementacja silnika obliczeniowego.

7 Aplikacja internetowa.

8 Testy.

9 Podsumowanie i wnioski.