Podstawy MPI

1. Napisać program równoległy korzystający z MPI, którego zadaniem będzie obliczenie przybliżenia liczby π. W tym celu należy wykorzystać równość:

$$\int_0^1 \frac{4.0}{(1+x^2)} dx = \pi$$

   Całka w powyższej formule może zostać przybliżona za pomocą sumy prostokątów:

$$\sum_{i=1}^{N} F(x_i)\Delta x \approx \pi$$

   gdzie każdy prostokąt ma szerokość Δx oraz wysokość $F(x_i)$, x_i to wartość na środku i-tego przedziału. Przykładowo, dla $N = 100$ mamy:

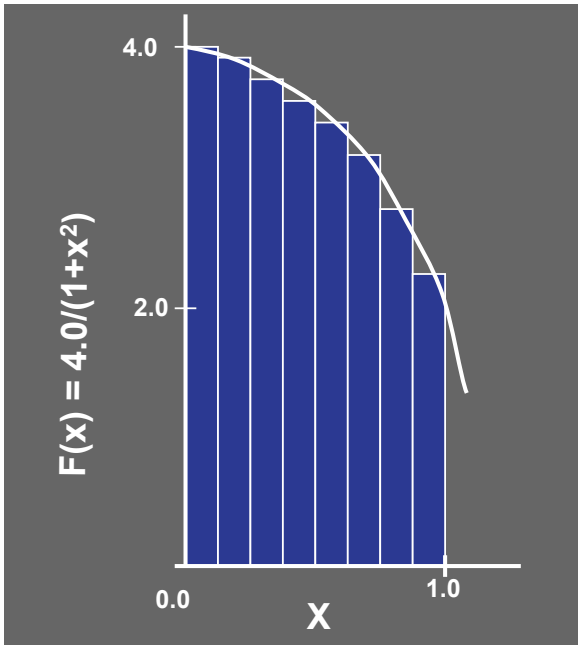
   - $\Delta x = 0.01$
   - $x_1 = (1 - 0.5)\Delta x$, $x_1 = (2 - 0.5)\Delta x$, ..., $x_{100} = (100 - 0.5)\Delta x$
   - $F(x_1) = \frac{4.0}{(1+0.005^2)}$

   Obliczenia należy podzielić równo między wszystkie procesy – tj. każdy proces powinien obliczać sumę dla innego zakresu indeksów. Sumy częściowe należy przesłać do procesu o identyfikatorze 0. Proces 0. powinien wyświetlić wynik wyznaczony na podstawie wszystkich sum częściowych. Sprawdzić czas działania programu w zależności od przyjętej liczby kroków (od 10^2, aż do 10^8).

2. Zrównoleglić za pomocą MPI algorytm mnożenia macierzy. Macierze źródłowe generowane są przez proces główny o identyfikatorze 0, a następnie rozsyłane do pozostałych procesów. Każdy proces odpowiedzialny jest za wyznaczenie wyniku mnożenia dla zadanego fragmentu macierzy wynikowej, a następnie przesłanie go do procesu głównego, ktory powinien go wyświetlić.

   Przykład. Niech dane będą macierze A oraz B o wymiarach $n \times n$, gdzie $n = 100$. Jeżeli program wykonuje się na dwóch procesach, to proces o id 0 odpowiedzialny jest za wyznaczenie wierszy macierzy wynikowej, $C = A \cdot B$, o indeksach od 1 do 50, natomiast proces o id 1 za wiersze o indeksach od 51 do 100.

3. Napisać program równoległy (z zastosowaniem MPI) generujący wizualizację zbioru Julii na podstawie poniższej funkcji, która dla zadanych współrzędnych sprawdza, czy punkt ten należy do zbioru. Punkty należące do zbioru należy pokolorować na biało, natomiast pozostałe na czarno.



Rysunek 8.1: Rys. do zadania 1.

Wygenerowany obraz zapisać w formacie PNG za pomocą biblioteki `lodepng` dostępnej na stronie `https://github.com/lvandeve/lodepng`

Zebraniem wyników obliczeń od wszystkich procesów i połączeniem ich w kompletny obraz zajmuje się proces główny (o identyfikatorze równym 0).

```
/*
Funkcja dla podanego pkt. (x, y) we współrzędnych obrazu (ekranu) sprawdza
przynależność do zbioru Julii i zwraca 1 w przypadku przynależności, a 0 w
przeciwnym przypadku.

x, y - współrzędne sprawdzanego pkt.
size - liczba pkt. obrazu w poziomie i w pionie (zakładamy, że jest kwadratowy)
*/
char in_julia(int x, int y, int size) {
    const float scale = 1.5f;
    const float jx = scale * (float)(size / 2 - x)/(size / 2);
    const float jy = scale * (float)(size / 2 - y)/(size / 2);
    float c[] = { -0.8f, 0.156f };
    float a[] = { jx, jy };
    const int max_iter = 1000;
    for (int i = 0; i < max_iter; ++i) {
        float a_next[] = { a[0]*a[0] - a[1]*a[1] + c[0],
                           a[0]*a[1] + a[1]*a[0] + c[1] };
        a[0] = a_next[0];
        a[1] = a_next[1];
        if (a[0] * a[0] + a[1] * a[1] > 4) {
            return 0;
        }
    }
    return 1;
}
```

Przykładowa struktura programu (program sekwencyjny):

```
#include <vector>
#include "lodepng.h"

using namespace std;

// ...

int main() {
    vector<unsigned char> pixels(width * height); // Bufor obrazu

    // Obliczenia dla każdego pkt. obrazu
    // ...

    // Zapis obrazu wynikowego, każdy pkt. w 8 bitowej skali szarości, tj.
    // 0 - kolor czarny, 255 - biały, wartości pomiędzy - odcienie szarości
    lodepng_encode_file("julia.png",
                        (const unsigned char*)pixels.data(),
                        width, height, LCT_GREY, 8);
    return 0;
}
```