Zestaw 1

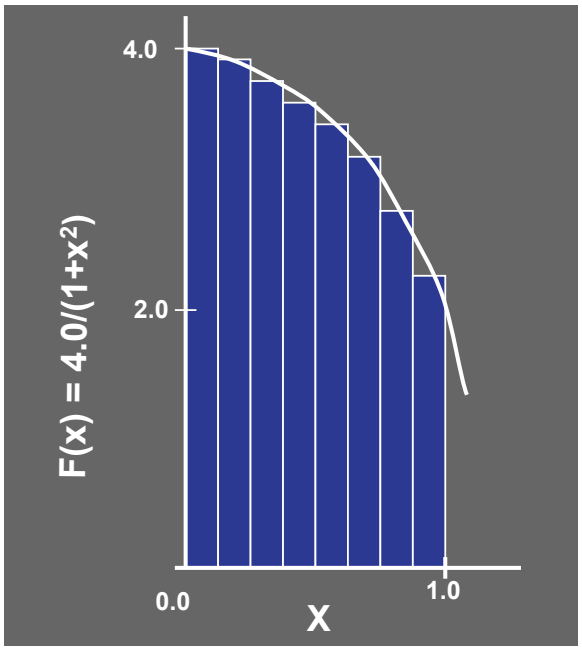
1. Napisać program wyznaczający przybliżenie liczby π. W tym celu należy wykorzystać równość:
$$\int_0^1 \frac{4.0}{1+x^2} dx = \pi$$
Całka w powyższej formule może zostać przybliżona za pomocą sumy prostokątów (Rys. 1.1)
$$\sum_{i=1}^{N} F(x_i)\Delta x \approx \pi$$
gdzie każdy prostokąt ma szerokość Δx oraz wysokość $F(x_i)$, x_i to wartość na środku i-tego przedziału. Przykładowo, dla $N = 10$ mamy:
   - $\Delta x = \frac{1}{N} = 0.1$
   - $x_1 = (1 - 0.5)\Delta x = 0.05\,,$
     $x_2 = (2 - 0.5)\Delta x = 0.15\,,$
     $\ldots\,,$
     $x_{10} = (10 - 0.5)\Delta x = 0.95\,.$
   - $F(x_1) = \frac{4}{1+0.05^2} \approx 3.99\,,$
     $F(x_2) = \frac{4}{1+0.15^2} \approx 3.912\,,$
     $\ldots\,,$
     $F(x_{10}) = \frac{4}{1+0.95^2} \approx 2.103\,.$
   - $\sum_{i=1}^{10} F(x_i)\Delta x \approx 3.142425985\,.$

   Funkcję obliczającą przybliżenie π należy zrównoleglić za pomocą OpenMP. Sprawdzić czas działania programu dla wartości $N \in \{10^1, 10^2, \ldots, 10^8\}$.

2. Napisać program wyznaczający *liczby pierwsze* metodą prostego poszukiwania dzielników liczby i zrównoleglić go za pomocą OpenMP. Liczba naturalna $n \in \mathbb{N}$, $n > 1$, jest pierwsza wtedy i tylko wtedy, gdy ma dokładnie dwa dzielniki, tj. 1 oraz n.

3. Napisać program wyznaczający liczby pierwsze metodą sita Eratostenesa i zrównoleglić go za pomocą OpenMP.

4. Zrównoleglić „naiwny" algorytm mnożenia dwóch macierzy $\mathbf{A}$ o wymiarach $n \times m$ oraz $\mathbf{B}$ o wymiarach $m \times p$:



Rysunek 1.1: Rys. do zadania 1.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}.$$

Wynikiem jest macierz $\mathbf{C}$:

$$\mathbf{C} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix}$$

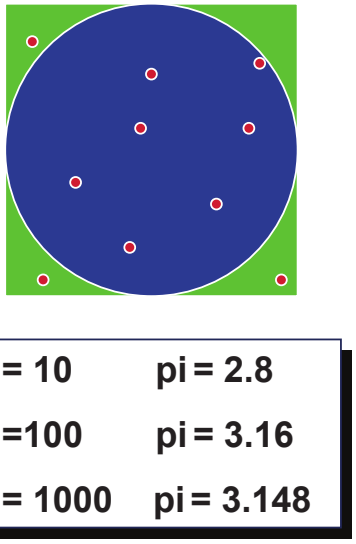
taka, że: $c_{ij} = a_{i1}b_{1j} + \cdots + a_{im}b_{mj} = \sum_{k=1}^{m} a_{ik}b_{kj}$. Proszę sprawdzić czas działa programu dla różnych wymiarów macierzy.

Czy czas działania zmieni się, gdy macierz $\mathbf{B}$ najpierw *transponujemy* (zamienimy miejscami wiersze z kolumnami), tak aby odczyt odbywał się „wiersz po wierszu" zamiast „kolumna po kolumie", tj. $c_{ij} = a_{i1}b^T_{j1} + \cdots + a_{im}b^T_{jm} = \sum_{k=1}^{m} a_{ik}b^T_{jk}$, gdzie b^T_{ij} oznacza element transponowanej macierzy $\mathbf{B}$.

5. Przybliżenie liczby π można otrzymać za pomocą algorytmu losowego typu *Monte Carlo*. Załóżmy, że w kwadrat o boku dł. 2 wpisano koło o promieniu $r = 1$. Algorytm polega na sukcesywnym losowaniu punktów należących do kwadratu (Rys. 1.2). Prawdopodobieństwo, że wylosowany punkt należy do koła jest równe stosunkowi pola koła do pola kwadratu, czyli:

$$P = \frac{r^2\pi}{4r^2} = \frac{\pi}{4}$$

Dzieląc liczbę wylosowanych punktów znajdujących się wewnątrz koła przez liczbę wszystkich wylosowanych punktów otrzymujemy oszacowanie wartości P, na podstawie której można wyznaczyć przybliżenie liczby π. Napisz program równoległy generujący przybliżenie liczby π wg opisanej metody. Zwróć uwagę na to, żeby każdy wątek miał dostęp do *odrębnego* ciągu liczb pseudolosowych, tak aby poszczególne wątki nie generowały tego samego ciągu liczb.



Rysunek 1.2: Ilustracja obliczania przybliżenia liczby π metodą Monte Carlo.

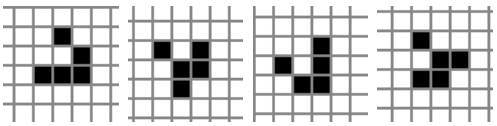
6. Zaimplementuj program symulujący *Grę w życie* Johna Conwaya. Program powinien umożliwić wygenerowanie stanu gry po wykonaniu zadanej liczby iteracji,

zaczynając od zadanego stanu początkowego. Reguły gry są następujące:

- gra odbywa się na prostokątnej planszy podzielonej na komórki, z których każda jest „żywa" albo „martwa";
- każda komórka ma dokładnie 8 sąsiadów;
- na podstawie bieżącego stanu gry generowany jest kolejny zgodnie z regułami:
  - martwa komórka, która ma dokładnie 3 żywych sąsiadów, staje się żywa (rodzi się);
  - żywa komórka z 2 albo 3 żywymi sąsiadami pozostaje nadal żywa; przy innej liczbie sąsiadów umiera (z "samotności" albo "zatłoczenia").

Na potrzeby symulacji należy założyć, że plansza jest skończona i przechowywana w tablicy p o wymiarach $n \times n$, przy czym komórki brzegowe sąsiadują z komórkami z przeciwległego brzegu, tak aby każda z nich miała 8 sąsiadów. Przykładowo, komórka $p[i][0]$ sąsiaduje z komórką $p[i][n-1]$, a komórka $p[0][i]$ z komórką $p[n-1][i]$.

Na potrzeby testowania poprawności działania programu można wykorzystać wzorzec „glider", którego kolejne stany przedstawia Rys. 1.3. Jest to wzorzec, który jest „wiecznie żywy", a więc jeżeli znajduje się na planszy startowej, to wystarczy sprawdzić, czy przetrwa np. 1000 iteracji.



Rysunek 1.3: Stany fragmentu planszy zawierającego wzorzec „glider" w kolejnych iteracjach.

Należy porównać czasy działania programu w wersji sekwencyjnej i równoległej dla różnych wielkości plansz.