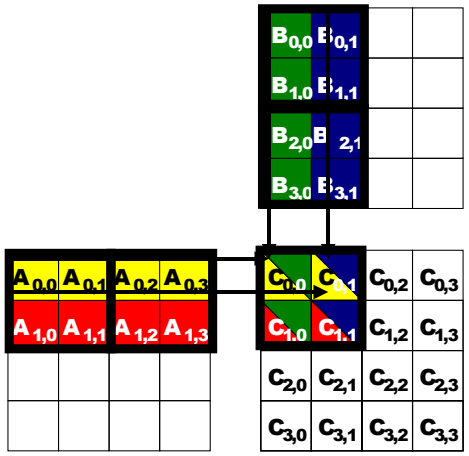
Podstawy OpenCL – część 2

1. Napisz program dokonujący mnożenia dwóch macierzy w wersji sekwencyjnej oraz OpenCL. Porównaj czasy działania obu wersji dla różnych wielkości macierzy, np. 16×16, 128×128, 1024×1024.

   Zmodyfikuj wersję OpenCL, tak aby obliczenia zostały podzielone na niezależne porcje, przy czym pojedyncza porcja dotyczy kwadratowego fragmentu macierzy wejściowych o wymiarach 16×16. Obliczenia dla porcji polegają na wspólnym załadowaniu elementów macierzy wejściowych odpowiadających przetwarzanemu fragmentowi do tablic pomocniczych w pamięci lokalnej, a następnie pomnożeniu tak powstałych „małych macierzy". Objaśnienie znajduje się na Rys. 2.1. Porównaj czasy działania pierwotnej i zmodyfikowanej wersji programu.



Rysunek 2.1: Ilustracja podziału obliczeń na porcje (kafelki). Rozmiar porcji odpowiada rozmiarowi grupy wątków (na rys. 2×2).

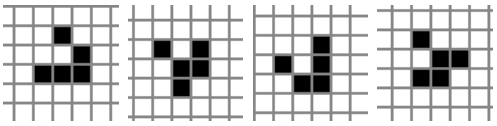
2. Zaimplementuj program symulujący *Grę w życie* Johna Conwaya. Program powinien umożliwić wygenerowanie stanu gry po wykonaniu zadanej liczby iteracji, zaczynając od zadanego stanu początkowego. Reguły gry są następujące:

   - 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 programu można wykorzystać wzorzec „glider", którego kolejne stany przedstawia Rys. 2.2. Jest to wzorzec, który jest „wiecznie żywy".



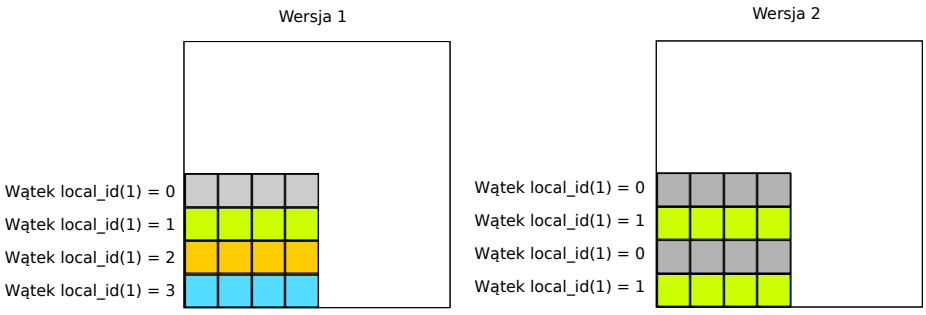
Rysunek 2.2: Stany fragmentu planszy zawierającego wzorzec „glider" w kolejnych iteracjach.

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

3. Napisz program kopiujący macierz, tj. z macierzy wejściowej do macierzy docelowej za pomocą kernela OpenCL w dwóch wersjach. W pierwszej każdy wątek kopiuje dokładnie jeden element z tablicy wejściowej do tablicy wynikowej o indeksie odpowiadającym indeksie wątku (grupa robocza odpowiada za kopiowanie odpowiadającego jej „kafelka" macierzy wejściowej). Rozmiar przestrzeni indeksowania powinien być równy rozmiarowi macierzy, np. $n \times n$.

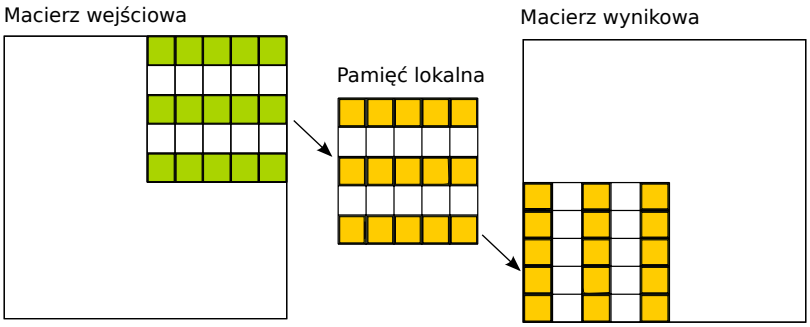
   W drugiej wersji każdy wątek w grupie roboczej powinien kopiować kilka elementów. Przykładowo, dla macierzy o wymiarach $n \times n$ rozmiar przestrzeni indeksowania można ustalić na $\frac{n}{2} \times n$ (tyle samo kolumn, ale o połowę mniej wierszy), wtedy każdy wątek w grupie kopiuje dwa elementy (patrz Rys. 2.3).

   Jak różnią się czasy działania obu wersji?



Rysunek 2.3: Ilustracja do zadania z kopiowaniem macierzy.

4. Napisz program dokonujący transpozycji macierzy za pomocą kernela OpenCL w dwóch wersjach. W pierwszej, elementy tablicy wejściowej są zapisywane w tablicy wynikowej bezpośrednio. W drugiej grupa wątków najpierw kopiuje (bez zmian) fragment tablicy wejściowej do tablicy w pamięci *lokalnej*, a następnie z tablicy lokalnej zapisuje transponowany fragment do tablicy wynikowej (patrz Rys. 2.4). Porównaj czasy działania obu wersji. Porównaj czas działania drugiej wersji algorytmu z czasem działania programu kopiującego macierz (bez transponowania).



Rysunek 2.4: Ilustracja transpozycji macierzy za pomocą kopiowania do tablicy w pamięci lokalnej urządzenia.

5. Napisz program dokonujący przekształcenia obrazu wejściowego za pomocą filtru kontekstowego o wymiarach 3×3. Wartość wynikowa koloru piksela jest wyznaczana poprzez przemnożenie wartości piksela oraz jego sąsiadów przez odpowiednie elementy filtra, a następnie zsumowanie tych wartości i podzielenie wyniku przez sumę wag filtru (patrz rys. 2.5). Bardziej formalnie:

$$g(x,y) = \sum_{i=-M}^{M} \sum_{j=-M}^{M} w(i,j) f(x+i, y+j) \ ,$$

gdzie:

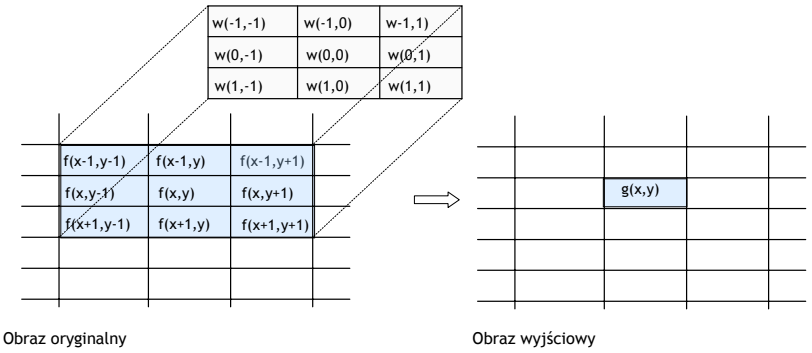
- $f(x,y)$ jest kolorem punktu (x,y) obrazu wejściowego,
- $g(x,y)$ jest kolorem punktu obrazu wynikowego,
- $w(i,j)$ jest wagą filtru,
- $M = \lfloor \frac{N}{2} \rfloor$ – gdzie N to rozmiar filtru.

Najprostszy filtr rozmywający $N \times N = 3 \times 3$ (górnoprzepustowy, *box filter*):

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Filtr Gaussa:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Rysunek 2.5: Ilustracja filtrowania obrazu.

6. Zaimplementować równoległy algorytm wyznaczający przybliżenie liczby π metodą Monte Carlo. Algorytm ten polega na losowaniu n punków należących do kwadratu o boku dł. a. W kwadrat ten wpisane jest koło o promieniu $r = a/2$. W celu wyznaczenia przybliżenia liczby π wyznaczamy stosunek liczby punktów leżących wewnątrz koła n_{in} do liczby wylosowanych punktów n.

   Prawdopodobieństwo tego, że wylosowany pkt. należy do koła jest równe stosunkowi pola koła do pola kwadratu:

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

   stąd:

   $$\pi \approx 4\frac{n_{in}}{n}\,.$$

7. Zaimplementuj równoległy algorytm wyznaczania przybliżonej wartości całki oznaczonej (rys. 2.6):

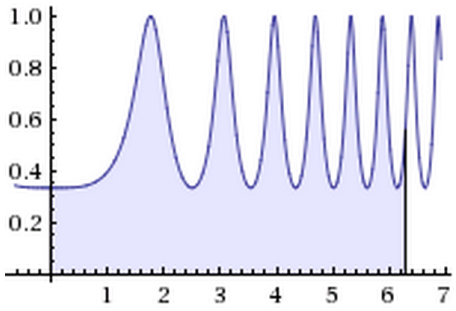
   $$\int_0^{2\pi} \frac{1}{2+\cos(x^2)}dx$$

   Przybliżona wartość całki oznaczonej to:

   $$\int_a^b f(x)dx \approx \frac{b-a}{n}\sum_{k=1}^{n} f(x_k)\,,$$

   gdzie $x_1, x_2, \ldots, x_n$ są liczbami losowymi o rozkładzie jednostajnym z przedziału $[a, b]$. Przybliżona wartość całki to: 3.43901. Sprawdzić dokładność w zależności od różnych wartości n. Porównać czas działania z implementacją sekwencyjną.

   Wskazówka: Przybliżenie π dostępne jest w stałej `M_PI`.



Rysunek 2.6: Wykres całki z Zad. 7