Zadanie 1.
Mnożenie macierzy

Celem zadania jest napisanie, przetestowanie i porównanie wydajności kilku różnych wersji kernela CUDA obliczającego iloczyn dwóch macierzy. Dla uproszczenia mają to być macierze kwadratowe NxN.

Jeśli macierz **C** powstaje jako iloczyn macierzy **A** i B $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$ to każdy element macierzy C jest dany wzorem:

$$c_{ij} = \sum_k a_{ik} \cdot b_{kj} \qquad (1)$$

gdzie pierwszy indeks jest indeksem wiersza a drugi kolumny. Czyli element macierzy **C** o indeksie {i,j} powstaje jako iloczyn skalarny i-tego wiersza macierzy **A** i j-tej kolumny macierzy **B**. Ten sam wzór można również interpretować jako sumę N kolejny macierzy C^k powstałych jako iloczyny zewnętrzne kolejnych kolumn macierzy A i kolumn macierzy B: $\mathbf{C^k} = A^k \otimes B^k$, czyli

$$c_{ij}^k = a_{ik} \cdot b_{jk} \qquad (2)$$

Należy opracować następujące wersje kodu:

1. Referencyjna wersja na CPU napisana w języku C/C++ implementująca wzór (1).
2. Kernel CUDA (**wersja #1**) implementujący bezpośrednie przełożenie kodu z CPU, zastępująca podwójną pętlę przebiegającą przez tablicę wynikową numerowaniem wątków.
3. Kernel CUDA (**wersja #2**) implementujący iloczyn macierzy jako zewnętrzny iloczyn wektorów. W tej wersji każdy blok o rozmiarze K wątków liczy kwadratowy kawałek macierzy wynikowej o rozmiarze KxK implementując wzór nr 2. Kernel wykorzystuje pamięć współdzieloną do przechowywania odpowiednich wektorów kolumnowych z macierzy A i macierzy B, oraz do przechowywania odpowiedniego fragmentu tablicy wynikowej.
4. Wersję kernela #2, w której usunięto konflikty banków pamięci (**wersja #3**).
5. Wersję kernela #3, w której oba wektory kolumnowe są przechowywane w rejestrach, natomiast tablica wynikowa jest przechowywana w pamięci współdzielonej (**wersja #4**). Należy użyć instrukcji wymiany zawartości rejestrów (instrukcje shuffle). Należy zastosować bloki o składające się z 32 wątków.
6. Wersję kernela #4, w której również fragment macierzy przechowywany jest w rejestrach (**wersja #5**). Każdy wątek przechowuje jeden fragment wiersza macierzy wynikowej.

Dla każdej wersji należy sprawdzić zgodność wyników z wersją CPU, podając dwie metryki:
1. Liczbę elementów różnych
2. oraz, jeżeli ta liczba jest różna od zera, to również sumę wartości bezwzględnych odchyleń między macierzami $Err = \sum_{ij} | C_{ij}^{CPU} - C_{ij}^{GPU} |$

W raporcie należy podać średnie przyspieszenie każdej z wersji w stosunku do referencyjnej implementacji CPU, uzyskane z co najmniej 10 powtórzeń wywołania kodu. Dla zapewnienia porównywalnych wyników w każdym powtórzeniu należy zmierzyć czas wykonania wersji CPU i wszystkich testowanych wersji kernela. Dla wybranych wersji kodu należy również zbadać

zależność wydajności od konfiguracji kernela. Wyniki należy podać w wersji tabelarycznej wg poniższego wzoru.

Tabela 1. Porównanie wydajności różnych wersji kernela

Wersja kodu	Konfiguracja kernela	Średni czas wykonania (ms)	Średnie przyspieszenie względem wersji CPU
CPU	—		—
Kernel #1	8 x 8		
Kernel #1	16 x 16		
Kernel #1	32 x 32		
Kernel #2	32 x 1		
Kernel #2	64 x 1		
Kernel #2	96 x 1		
Kernel #2	128 x 1		
Kernel #3	32 x 1		
Kernel #3	64 x 1		
Kernel #4	32 x1		
Kernel #5	32 x1		

W raporcie należy przeprowadzić dyskusję wyników - skąd wynikają zaobserwowane różnice wydajności różnych wersji kodu.