

## **Programowanie Równoległe i Rozproszone**

### **Tematy projektów**

Każdy projekt powinien być zrealizowany w dowolnym języku programowania z wykorzystaniem trzech technologii z dziedziny programowania równoległego: OpenMP, MPI oraz CUDA/OpenCL. Mogą to być bezpośrednie odpowiedniki tych technologii w wybranym języku programowania. Każdy z projektów jest maksymalnie 2-osobowy. Do każdego projektu powinna być dołączona dokumentacja w postaci krótkiego sprawozdania z wykonania projektu, wraz z rozpisaniem tabeli z podziałem pracy pomiędzy autorów, szczegółowego opisu konfiguracji komputerów na których zostały przeprowadzone testy, objaśnienia kluczowych fragmentów kodu, testów wydajnościowych oraz instrukcji obsługi interfejsu. Konieczne jest wykonanie wykresów przyśpieszenia i efektywności zrównoleglenia dla rosnącej liczby wątków dla każdej z technologii oraz wykonanie wykresu porównawczego dla najlepszych uzyskanych wyników z każdej z technologii.

#### **Harmonogram Projektu:**

07.10.2025/10.10.2025 – konsultacje projektowe (obowiązkowe)  
21.10.2025/24.10.2025 – konsultacje projektowe  
28.10.2025/31.10.2025 – konsultacje projektowe (obowiązkowe)  
04.11.2025/07.11.2025 – konsultacje projektowe  
18.11.2025/21.11.2025 – konsultacje projektowe (obowiązkowe)  
02.12.2025/05.12.2025 – konsultacje projektowe  
09.12.2025/12.12.2025 – termin oddania projektu

#### **1. Rozprzestrzenianie koloru (prosta dyfuzja na siatce)**

##### **Opis:**

Projekt przedstawia prostą symulację rozlewania się koloru po dwuwymiarowej siatce (np. tablice  $1000 \times 1000$ ). Na początku w kilku punktach siatki znajdują się źródła koloru (np. czerwony, zielony, niebieski), które z każdą iteracją rozlewają się na sąsiednie pola. Wartość koloru w każdej komórce jest obliczana jako średnia z jej sąsiadów, dzięki czemu barwy płynnie się przenikają. Efekt nie wymaga stosowania fizyki – wystarczy iteracyjne uśrednianie wartości, podobne do działania filtra rozmycia. Wynik obrazuje rozchodzenie się ciepła, farby lub innej substancji na powierzchni.

##### **Zakres implementacji:**

OpenMP – równoległa aktualizacja komórek siatki. MPI – podział siatki na bloki i wymiana danych brzegowych między procesami. CUDA/OpenCL – obliczenia na GPU w oparciu o szablon sąsiedztwa (5/9 punktów) z wykorzystaniem pamięci współdzielonej. Wynik powinien być prezentowany jako mapa kolorów lub krótka animacja.

##### **Dodatkowe wyniki do raportu:**

Porównanie czasów (CPU/GPU), wykres przyspieszenia dla różnych rozmiarów siatki oraz zrzuty ekranu/klatki animacji.

## **2. Gra w życie Conwaya (Game of Life)**

### **Opis:**

Projekt dotyczy implementacji klasycznego automatu komórkowego, w którym komórki siatki są żywe lub martwe. Zmiany stanu wynikają z prostych zasad zależnych od liczby sąsiadów, co prowadzi do powstawania wzorców (glidery, oscylatory). Wynik może być pokazywany w formie animacji kolejnych generacji.

### **Zakres implementacji:**

OpenMP – równoległa aktualizacja komórek z użyciem dwóch buforów. MPI – podział planszy na fragmenty i wymiana krawędzi po każdej iteracji. CUDA/OpenCL – realizacja kroku gry na GPU z wykorzystaniem pamięci współdzielonej oraz bitowego zapisu stanu.

### **Dodatkowe wyniki do raportu:**

Czasy wykonania i przyspieszenie dla różnych rozmiarów planszy; przykładowe obrazy/animacje; krótkie wnioski o skalowaniu.

## **3. Symulacja cząstek 2D (ruch losowy z kolizjami)**

### **Opis:**

Projekt pokazuje zachowanie wielu prostych cząstek poruszających się po planszy 2D ruchem losowym. Cząstki odbijają się od krawędzi i mogą zmieniać kolor lub licznik przy zderzeniach. Zadanie pozwala przećwiczyć równoległe obliczenia na dużych zbiorach obiektów oraz proste metody zawężania par do sprawdzenia (siatka/kafle).

### **Zakres implementacji:**

OpenMP – równoległa aktualizacja pozycji i obsługa kolizji. MPI – podział planszy na bloki i przekazywanie cząstek między procesami. CUDA/OpenCL – obliczanie wektorów ruchu i sprawdzanie kolizji w obrębie kafli na GPU.

### **Dodatkowe wyniki do raportu:**

FPS przetwarzania, skalowanie (liczba cząstek vs wątki/procesy/bloki), krótki film lub seria zrzutów ekranu.

#### **4. Filtrowanie obrazów (rozmycie, wyostrzanie, krawędzie)**

##### **Opis:**

Projekt obejmuje zastosowanie klasycznych filtrów splotowych do obrazów (m.in. rozmycie Gaussa, wyostrzanie, operator Sobela/Canny). Celem jest przyspieszenie wsadowego przetwarzania wielu plików oraz porównanie jakości efektów.

##### **Zakres implementacji:**

OpenMP – równoległe przetwarzanie wierszy i/lub kafli obrazu. MPI – rozdzielanie plików między procesy i scalanie wyników. CUDA/OpenCL – implementacja filtrów jako jąder z pamięcią współdzieloną; rozważ separowalny Gauss ( $2 \times 1D$ ).

##### **Dodatkowe wyniki do raportu:**

Czasy działania dla różnych rozdzielczości, porównanie PSNR/SSIM, przykłady obrazów przed/po.

#### **5. Segmentacja obrazu metodą K-means**

##### **Opis:**

Projekt ilustruje segmentację obrazu przez grupowanie pikseli na K klas o podobnych kolorach. Wynik to uproszczony obraz z mniejszą liczbą barw, co bywa użyteczne w grafice i analizie obrazów.

##### **Zakres implementacji:**

OpenMP – równoległe przypisywanie pikseli do najbliższych centroidów i sumowanie częściowe. MPI – podział danych i wspólna aktualizacja centroidów (Allreduce). CUDA/OpenCL – zbiorcze obliczanie odległości i aktualizacje na GPU.

##### **Dodatkowe wyniki do raportu:**

Czasy działania i przyśpieszenie dla różnych K; wizualne porównanie obrazu oryginalnego i po segmentacji.

#### **6. Histogram i wyrównywanie kontrastu (equalizacja)**

##### **Opis:**

Projekt skupia się na zliczaniu histogramu jasności/koloru i poprawie kontrastu obrazu metodą equalizacji.

##### **Zakres implementacji:**

OpenMP – lokalne histogramy w wątkach i redukcja do globalnego. MPI – sumowanie histogramów częściowych. CUDA/OpenCL – histogram per blok + łączenie i prefix-sum; wersja kolorowa (po kanałach).

##### **Dodatkowe wyniki do raportu:**

Porównanie jakości (przed/po) oraz czasu obliczeń; wpływ liczby przedziałów na wydajność.

## **7. Filtr medianowy (odszumianie obrazów)**

### **Opis:**

Projekt dotyczy usuwania szumu impulsowego za pomocą filtru medianowego. Dla każdego piksela należy pobrać wartości z jego najbliższego otoczenia (np.  $3 \times 3$  lub  $5 \times 5$ ), posortować je i wybrać wartość środkową jako nowy kolor piksela — dzięki temu pojedyncze zakłócenia są skutecznie eliminowane bez rozmywania krawędzi.

### **Zakres implementacji:**

OpenMP – równoległe przetwarzanie wierszy/kafli. MPI – rozdział obrazów i scalenie wyników. CUDA/OpenCL – jądro medianowe ( $3 \times 3/5 \times 5$ ) z pamięcią współdzieloną i minimalizacją odczytów globalnych.

### **Dodatkowe wyniki do raportu:**

PSNR/SSIM i czasy dla różnych rozmiarów okna; przykłady obrazów przed/po.

## **8. Detekcja ruchu w wideo**

### **Opis:**

Projekt polega na wykrywaniu ruchu metodą różnicowania kolejnych klatek i prostego modelu tła.

### **Zakres implementacji:**

OpenMP – analiza pikseli, progowanie i prosta morfologia (otwarcie/zamknięcie). MPI – rozdział klatek/pliki wideo między procesy. CUDA/OpenCL – różnicowanie i progowanie na GPU; opcjonalne tło.

### **Dodatkowe wyniki do raportu:**

FPS przetwarzania dla różnych rozdzielczości i liczb wątków; przykładowe klatki z zaznaczonym ruchem.

## **9. Spektrogram audio (STFT)**

### **Opis:**

Projekt przekształca pliki dźwiękowe w obrazy spektrogramów, pokazując rozkład energii w czasie i częstotliwości.

### **Zakres implementacji:**

OpenMP – równoległe okienkowanie i FFT partii próbek. MPI – rozdział plików i agregacja wyników. CUDA/OpenCL – grupowe (wsadowe) przekształcenie Fouriera wykonywane na karcie graficznej oraz skalowanie amplitudy w skali logarytmicznej.

### **Dodatkowe wyniki do raportu:**

Czasy dla różnych rozmiarów FFT i okna; przykładowe spektrogramy; porównanie CPU vs GPU.

## **10. Filtracja audio Finite Impulse Response (direct vs FFT)**

### **Opis:**

W projekcie należy zaimplementować filtrację audio FIR w dwóch wariantach: bezpośrednia konwolucja (direct) oraz szybka metoda w dziedzinie częstotliwości (FFT, overlap-save), z możliwością wyboru typu filtra (dolno-/górnoprzepustowy) i długości impulsu  $h[k]$ . Trzeba przygotować trzy wersje obliczeń: OpenMP (równoległa konwolucja blokowa), MPI (podział plików/segmentów między procesy i scalanie wyników) oraz CUDA/OpenCL (implementacja overlap-save na GPU lub direct FIR dla krótszych filtrów), dbając o poprawne łączenie bloków na granicach. Program powinien przyjmować plik WAV/FLAC, parametry filtra (częstotliwości odcięcia, długość, okno) i tryb (direct/FFT), zapisywać wynik do pliku oraz raportować czasy przetwarzania. Dodatkowo należy wygenerować wykresy widma przed/po filtracji i wskazać punkt, od którego metoda FFT staje się szybsza niż direct.

### **Zakres implementacji:**

OpenMP – równoległa konwolucja blokowa. MPI – podział plików/segmentów i scalanie wyników. CUDA/OpenCL – implementacja overlap-save na GPU lub bezpośrednie FIR dla krótszych filtrów.

### **Dodatkowe wyniki do raportu:**

Czasy działania obu metod, wizualizacja widm przed/po, wnioski o progu opłacalności FFT.

## **11. Analiza logów systemowych (zliczanie i filtrowanie)**

### **Opis:**

Projekt dotyczy równoległego przetwarzania dużych plików tekstowych – np. dzienników systemowych (logów), raportów z serwerów, wyników pomiarów czy danych z czujników IoT. Zadaniem programu jest analiza zawartości plików w celu:

- zliczenia częstości występowania określonych słów lub fraz (np. „ERROR”, „WARNING”, „INFO”),
- wyodrębnienia wierszy spełniających określone kryteria (np. tylko błędy lub tylko określony zakres dat),
- stworzenia statystyk dotyczących zdarzeń w czasie (np. liczba błędów na godzinę).

### **Zakres implementacji:**

OpenMP – równoległa tokenizacja i lokalne słowniki; końcowa redukcja. MPI – podział plików między procesy, łączenie wyników. CUDA/OpenCL – przyspieszenie zliczania/histogramów na GPU.

### **Dodatkowe wyniki do raportu:**

Wykres top-N słów, przepustowość GB/s, porównanie CPU/GPU; krótkie wnioski o wąskich gardłach I/O.

## **12. Szyfrowanie plików XOR (pipeline)**

### **Opis:**

W projekcie należy napisać program, który szyfruje duże pliki metodą XOR, przetwarzając dane blokami w sposób równoległy. Każdy blok powinien być odczytywany, szyfrowany i zapisywany niezależnie – w zależności od wersji: przez różne wątki (OpenMP), procesy (MPI) lub bloki GPU (CUDA/OpenCL). Program powinien umożliwiać podanie ścieżki pliku wejściowego, klucza szyfrującego oraz nazwę pliku wynikowego, a następnie zweryfikować poprawność przez odszyfrowanie (powtórne zastosowanie XOR).

### **Zakres implementacji:**

OpenMP – przetwarzanie bloków i pipeline (czytanie–szyfrowanie–zapis). MPI – rozdział dużych plików na segmenty i rekonstrukcja. CUDA/OpenCL – blokowe XOR na GPU.

### **Dodatkowe wyniki do raportu:**

Przepustowość MB/s dla CPU i GPU; wpływ rozmiaru bloków i buforów; kontrola poprawności (sumy kontrolne).

## **13. Sortowanie równoległe (Quick/Merge/Radix)**

### **Opis:**

Projekt porównuje różne metody sortowania w środowisku równoległym i rozproszonym.

### **Zakres implementacji:**

OpenMP – QuickSort/MergeSort z ograniczeniem głębokości rekurencji. MPI – sample sort lub inny wariant rozproszony. CUDA/OpenCL – Thrust/CUB radix sort lub własne jądra.

### **Dodatkowe wyniki do raportu:**

Czas sortowania vs rozmiar danych i rozkład (losowy, prawie posortowany, z duplikatami); przyśpieszenie CPU/GPU.

## **14. Redukcje i skany (sumy, min/max, prefix-sum)**

### **Opis:**

W projekcie należy zaimplementować operacje redukcji (sumy, min, max) oraz skan prefix-sum dla bardzo dużych tablic. Dane powinny być dzielone na kafle/bloki, z obsługą typów int64 i float/double, a poprawność należy sprawdzić porównaniem z wynikiem sekwencyjnym (różnica względna/absolutna dla zmiennoprzecinkowych).

### **Zakres implementacji:**

OpenMP – redukcje i skan równoległy (Blelloch). MPI – Allreduce/Reduce-scatter + Allgather. CUDA/OpenCL – CUB/Thrust lub własne jądra w pamięci współdzielonej.

### **Dodatkowe wyniki do raportu:**

Przepustowość GB/s i efektywność względem teoretycznego maksimum; wpływ rozmiarów bloków i ułożenia pamięci.

## **15. Benchmark modeli równoległych (map/filter/reduce/sort)**

### **Opis:**

W projekcie należy przygotować jednolity benchmark czterech prymitywów: map, filter, reduce i sort, zaimplementowanych w trzech wersjach: OpenMP, MPI oraz CUDA/OpenCL (ten sam interfejs wejścia/wyjścia i te same dane). Trzeba dodać generator danych (np. liczby całkowite i zmiennoprzecinkowe, rozkład losowy/prawie posortowany/z duplikatami), konfigurowalne parametry wykonania (rozmiar danych, liczba wątków/procesów, rozmiary bloków GPU) oraz wspólne API testowe uruchamiające każdy prymityw. Program ma automatycznie mierzyć czas/przepustowość, zapisywać wyniki do CSV i generować proste wykresy porównawcze dodatkowo dodać autotuning (przeszukanie kilku wartości parametrów i wybór najlepszej).

### **Zakres implementacji:**

OpenMP, MPI i CUDA/OpenCL – implementacja tych samych operacji na identycznych danych; prosty autotuning parametrów (wątki, procesy, rozmiary bloków).

### **Dodatkowe wyniki do raportu:**

Wykresy przyspieszenia i efektywności; analiza wąskich gardeł (obliczenia vs komunikacja vs I/O).

## **16. Mozaika obrazu (image tiling)**

### **Opis:**

Projekt polega na złożeniu dużego obrazu z miniatur wybieranych na podstawie podobieństwa kolorystycznego. Z obrazu docelowego należy wyciąć kafle o zadanym rozmiarze, a następnie dla każdego kafla dobrać miniaturę z bazy (folder z obrazami) na podstawie podobieństwa kolorystycznego/cech. Program powinien umożliwiać konfigurację rozmiaru kafla, metryki (np. średni kolor, histogram, ewentualnie lokalny wzorzec binarny - LBP), limitu powtórzeń tej samej miniatury i zapisywać końcowy obraz mozaiki oraz statystyki czasu/przepustowości.

### **Zakres implementacji:**

OpenMP – obliczanie cech (średni kolor/histogram) i dopasowania. MPI – rozdział bazy miniatur i scalanie mapy kafli. CUDA/OpenCL – przyspieszenie obliczeń odległości i k-NN (k-najbliższych sąsiadów).

### **Dodatkowe wyniki do raportu:**

Jakość mozaiki (SSIM/MSE względem oryginału), czas generowania; podgląd różnic względem oryginału.

## **17. Symulacja kolejki M/M/c**

### **Opis:**

Projekt dotyczy symulacji systemu kolejkowego typu M/M/c, w którym klienci przychodzą losowo, oczekują w kolejce i są obsługiwani przez wiele niezależnych stanowisk. Program powinien generować zdarzenia przyjścia i zakończenia obsługi, utrzymywać stan kolejki oraz zliczać podstawowe statystyki: średni czas oczekiwania, zajętość serwerów i liczbę klientów w kolejce. Implementacja ma obejmować trzy wersje: OpenMP (równoległa obsługa wielu symulacji lub zdarzeń), MPI (rozdział eksperymentów między procesy i agregacja wyników), oraz CUDA/OpenCL (symulacja wielu kolejek jednocześnie na GPU), z możliwością konfiguracji liczby stanowisk, intensywności przyjść i czasu obsługi.

### **Zakres implementacji:**

OpenMP – równoległa obsługa zdarzeń i zliczanie metryk. MPI – wiele replik symulacji z agregacją wyników. CUDA/OpenCL – akceleracja generowania zdarzeń/losowań i kolejek na GPU.

### **Dodatkowe wyniki do raportu:**

Porównanie z wartościami teoretycznymi (Erlang-C), throughput symulacji, wnioski o skalowaniu.

## **18. PageRank dla grafu sieciowego**

### **Opis:**

W projekcie należy zaimplementować równoległe obliczanie algorytmu PageRank, który określa znaczenie (ranking) węzłów w grafie skierowanym — np. stron internetowych, połączeń sieciowych lub relacji między obiektami.

Program powinien wczytywać graf zapisany w postaci listy krawędzi lub macierzy sąsiedztwa, a następnie wykonywać iteracyjne przeliczenia wartości PageRank aż do osiągnięcia zbieżności (czyli gdy zmiany między iteracjami są bardzo małe). Program powinien umożliwiać zmianę współczynnika tłumienia (zwykle 0.85) i zapisywać końcowy ranking do pliku.

### **Zakres implementacji:**

OpenMP – równoległe iteracje metody potęgowej i redukcje. MPI – rozdział wierszy macierzy/wektora i komunikacja (Allreduce). CUDA/OpenCL – implementacja SpMV (CSR/ELL) i aktualizacji wektora na GPU.

### **Dodatkowe wyniki do raportu:**

Liczba iteracji do zbieżności, MTEPS/GFLOP/s dla SpMV, skalowanie dla różnych rozmiarów grafu.

## **19. Symulacja ruchu pojazdów na skrzyżowaniu (model agentowy)**

### **Opis:**

Projekt modeluje proste skrzyżowanie z sygnalizacją świetlną, pasami ruchu i pojazdami jako agentami. Każdy pojazd przestrzega prostych zasad (zatrzymanie na czerwonym, odstęp, skręt), co pozwala obserwować płynność ruchu i powstawanie korków.

### **Zakres implementacji:**

OpenMP – równoległa aktualizacja agentów i kolizji logiki. MPI – podział skrzyżowania/siatki dróg na sektory, wymiana agentów między procesami. CUDA/OpenCL – równoległe aktualizacje stanu wielu pojazdów na GPU.

### **Dodatkowe wyniki do raportu:**

Średni czas przejazdu, liczba zatrzymań i przepustowość (pojazdy/min); krótka animacja symulacji.

## **20. Równoległe wyszukiwanie wzorców w tekście (Aho–Corasick / PFAC)**

### **Opis:**

Projekt wyszukuje wiele wzorców w dużych plikach tekstowych. Aho–Corasick zapewnia szybkie dopasowania na CPU, natomiast PFAC (Parallel Failureless Aho–Corasick) umożliwia prostą równoległość na GPU. Zadanie ma charakter praktycznego przetwarzania danych tekstowych.

### **Zakres implementacji:**

OpenMP – tekst dzielony na fragmenty analizowane przez wątki, MPI – każdy proces przeszukuje osobny blok z uwzględnieniem wzorców na granicach fragmentów, CUDA/OpenCL – zastosowanie uproszczonego algorytmu PFAC (Parallel Failureless Aho–Corasick), gdzie każdy wątek GPU analizuje inne przesunięcie tekstu.

### **Dodatkowe wyniki do raportu:**

Przepustowość GB/s, skalowanie na liczbę wzorców i długość tekstu; porównanie CPU vs GPU.