Projekt zaliczeniowy

Raport końcowy

Zespół Projektowy nr 1

Adrian Rybaczuk 318483

Bartek Cylwik 325457

12 czerwca 2025

Streszczenie

Niniejszy raport przedstawia podsumowanie projektu grupowego, systemu do równoległego przetwarzania obrazów satelitarnych Sentinel-2, umożliwiającego efektywną analizę zmian roślinności z wykorzystaniem wskaźników NDVI i NDMI oraz przyjaznego interfejsu graficznego. Projekt skupia się na implementacji i porównaniu różnych strategii przetwarzania równoległego, w tym obliczeń na GPU oraz wielowątkowych obliczeń na CPU, w celu zbadania ich wydajności i praktycznej użyteczności w zależności od charakterystyki problemu.

Spis treści

1 Wprowadzenie

Celem projektu była implementacja (wraz z GUI) systemu równoległego / rozproszonego mechanizmu wyliczania indeksów wilgotności (NDMI) i wegetacji (NDVI) roślin na zobrazowaniach satelitarnych Sentinel-2 do stwierdzenia występowania lub zaniku roślinności na wskazanym obszarze. Na podstawie tak zdefiniowanego zadania mogliśmy wyznaczyć poszczególne elementy projektu do realizacji:

- Uzyskanie wskaźników NDVI i NDMI
  - Research na temat wskaźników NDVI i NDMI
  - Pobranie niezbędnych wskaźników z zobrazowania satelitarnego Sentinel-2
  - Zapisanie danych pobranych z API jako "cache"w celu uniknięcia ponownego pobierania danych
  - Identyfikacja potęcjalnych problemów związanych z przetwarzaniem danych
  - Przeprowadzenie obliczeń wskaźników
- Interfejs graficzny
  - Ekran uwierzytelniania z API Sentinel-Hub
  - Ekran pozwalający na wybór zakresu czasowego oraz obszaru analizy
  - Ekran z wynikami analizy w postaci mapy z zaznaczonymi obszarami roślinności
- Równoległe przetwarzanie
  - Wybranie obszarów programu wymagających przetworzenia równoległego / rozproszonego
  - Wybranie pomiedzy przetwarzaniem na CPU a GPU
  - Implementacja mechanizmu równoległego / rozproszonego przetwarzania
- Wybranie technologi do realizacji projektu

2 Kroki podjęte na początku pracy

2.1 Research wskaźników NDVI i NDMI

Aby uzyskać wskaźniki NDVI i NDMI, rozpoczęliśmy od analizy literatury i dokumentacji dotyczącej tych indeksów, przechodząc od ogólnych informacji do szczegółowych aspektów ich wyznaczania.

NDVI (Normalized Difference Vegetation Index) [1] jest prostym wskaźnikiem ilościowym służącym do klasyfikacji wegetacji roślin. Jego wartość mieści się w przedziale od -1 do 1:

- Wartości ujemne (zbliżone do 0) wskazują na obecność wody.
- Zakres -0.1 do 0.1 odpowiada obszarom jałowym (skały, piasek, śnieg).
- Przedział 0.2 do 0.4 oznacza niską roślinność (zarośla, łąki).
- Wysokie wartości (bliskie 1) wskazują na bujną roślinność (np. lasy deszczowe).

Dzięki temu NDVI jest dobrym wskaźnikiem obecności i kondycji roślinności na danym obszarze.

Wskaźnik NDVI definiuje się wzorem:

$$\text{NDVI} = \text{Index}(\text{NIR}, \text{RED}) = \frac{\text{NIR} - \text{RED}}{\text{NIR} + \text{RED}}$$

W przypadku danych Sentinel-2, indeks ten wyznaczamy na podstawie kanałów B8 (NIR) i B4 (RED):

$$\text{NDVI} = \text{Index}(\text{B8}, \text{B4}) = \frac{\text{B8} - \text{B4}}{\text{B8} + \text{B4}}$$

NDMI (Normalized Difference Moisture Index) [2] jest znormalizowanym wskaźnikiem wilgotności który do wyznaczenia wilgotności wykorzystuje pasma NIR i SWIR.

- Wartosci ujemne (zblizone do -1) wskazuja na bardzo suche obszary (brak roslnosci, pustynia, obszary zabudowane)
- Wartosci od -0.2 do 0.2 wskazuja na glebe wysychającą (rzadka trawa, trawy wysychające, krzewy w stanie stresu wodnego)
- Wartosci ponizej do 0.2 do 0.4 odpowiadaja umiarkowanemu nawodnieniu (trawy, pastwiska)
- Wartosci od 0.4 Wskazują na obszary o zdrowy stan nawodnienia (uprawy w pełni sezonu wegetacyjnego, lasy liściaste w strefie umiarkowanej)
- Wartosci powyzej 0.4 wskazuja na obszary o wysokiej wilgotnosci (lasy deszczowe, obszary podmokłe)

Wskaźnik NDMI definiujemy wzorem:

$$\text{NDMI} = \text{Index}(\text{NIR}, \text{SWIR}) = \frac{\text{NIR} - \text{SWIR}}{\text{NIR} + \text{SWIR}}$$

W przypadku danych Sentinel-2, indeks ten wyznaczamy na podstawie kanałów B8 (NIR) i B11 (SWIR):

$$\text{NDMI} = \text{Index}(\text{B8}, \text{B11}) = \frac{\text{B8} - \text{B11}}{\text{B8} + \text{B11}}$$

2.2 RED, NIR, SWIR czyli B4, B8, B11

RED to czerwony kanał światła, silnie odbijany przez martwe liście. Wykorzystuje się go do identyfikacji typów roślinności, gleb oraz obszarów zabudowanych. Charakteryzuje się ograniczoną penetracją w wodzie i słabym odbiciem od liści zawierających chlorofil (żywe liście).

W satelitach Sentinel-2 kanał RED odpowiada pasmu B4: [3]

- Rozdzielczość: 10 m/px
- Centralna długość fali: 665 nm
- Szerokość pasma: 30 nm

NIR (Near Infrared) to bliska podczerwień, która dobrze obrazuje linie brzegowe oraz zawartość biomasy.

W Sentinel-2 kanał NIR odpowiada pasmu B8: [4]

- Rozdzielczość: 10 m/px
- Centralna długość fali: 842 nm
- Szerokość pasma: 115 nm

SWIR (Short-Wave Infrared) jest to fala wysokowrażliwa na zawartość wody w obiektach. Dlatego jest dobrym wskaźnikiem wilgotności.

W Sentinel-2 kanał SWIR odpowiada pasmu B11: [5]

- Rozdzielczość: 20 m/px
- Centralna długość fali: 1610 nm
- Szerokość pasma: 130 nm

2.3 Problem Różnych Rozdzielczości

Istotną informacją na temat pasm dostarczanych przez Sentinel-2 jest ich natywna rozdzielczość. Możemy zauważyć różnice w rozdzielczości między kanałami, np. B4 i B8 (10m) a B11 (20m). Naszą początkową hipotezą była konieczność ręcznego przeskalowania danych (downscalingu) w celu ich ujednolicenia.

Jednakże, w toku prac odkryliśmy, że API Sentinel Hub, z którego korzystamy, dostarcza wszystkie zamówione pasma już przeskalowane do wspólnej, najwyższej rozdzielczości (w naszym przypadku 10m). Mechanizm ten, znany jako co-rejestracja, jest standardową praktyką w przetwarzaniu danych satelitarnych i znacznie uprościł nasz proces przetwarzania wstępnego. Pozwoliło nam to operować na danych o najwyższej możliwej szczegółowości bez ryzyka utraty informacji, która mogłaby wystąpić przy ręcznym zmniejszaniu rozdzielczości.

2.4 Zrównoleglenie / rozproszenie obliczeń

W związku z tym, że chcieliśmy, by aplikacja działała na jednym komputerze, wybraliśmy ścieżkę zrównoleglenia obliczeń. Zadanie obliczania wskaźników polega na wykonaniu prostych operacji arytmetycznych dla milionów pikseli. Operacje te są powtarzalne i niezależne od siebie, co czyni je idealnym kandydatem do zrównoleglenia.

Naszą początkową hipotezą było, że obliczenia na GPU, ze względu na ich masowo równoległą architekturę, będą najwydajniejszym rozwiązaniem. Jednakże, aby przeprowadzić rzetelną analizę i zrozumieć kompromisy związane z różnymi podejściami, postanowiliśmy zaimplementować i porównać trzy odrębne strategie obliczeniowe:

1. **Przetwarzanie na GPU** z wykorzystaniem języka Taichi, aby zbadać potencjał masowego zrównoleglenia.

2. **Równoległe przetwarzanie na CPU** z wykorzystaniem modułu `multiprocessing` i pamięci dzielonej, aby ocenić skalowalność na wielordzeniowych procesorach.

3. **Jednowątkowe przetwarzanie na CPU** z wykorzystaniem wysoce zoptymalizowanej biblioteki NumPy, służące jako kluczowy punkt odniesienia (baseline) do oceny narzutu (overhead) metod równoległych.

Taki zakres implementacji pozwolił na dogłębną analizę wydajności i zidentyfikowanie, która metoda jest optymalna w zależności od charakterystyki problemu, co stanowi główny trzon niniejszego raportu.

2.5 Następnym krokiem będzie uzyskanie dostępu do danych z Sentinel-2

Do tego celu skorzystaliśmy z API Sentinel-Hub. [6] Udostępnia on dane z satelity poprzez API dopiero po uwierzytelnieniu. W tym celu musielismy przejść przez process opisany tutaj [7]. Po jego przejsciu mamy dostep do Client ID oraz Client Secret które wykorzystamy do połączenia się z API.

API posiada reate limiting które są w miare restrykcyjne. [8] Z tego też powodu zamierzamy wykorzystać cache do przechowywania pobranych już danych. Z natury projektu i dania sobie możliwości testowania wyników zamierzamy przechowywać nieprzetworzone dane zamiast obliczonych już wyników. Pomoże nam to przeprowadzić testy i wprowadzić poprawki w przyszłości.

2.6 Wybór technologii

Jako że określiliśmy już dokładniej ramy projektu możemy potwierdzić wybrane przez nas technologie. Ogólny język implementacji to Python. Do pobierania danych z api wykorzystaliśmy bibliotekę SentinelHub. Która udosptępnia już gotowe funkcje pozwalające zautoryzować się z api oraz pobrać dane. Do gui wykorzystaliśmy Tkinter podstawową i dosyć prostą bibliotekę do tworzenia interfejsu graficznego. Do prezentacji mapy wykorzystaliśmy TkinterMapView. Do zrównoleglenia obliczeń skorzystamy z Taichi Lang [9] którego zaembedujemy w pythonie. Jest to język programowania osadzony w pythonnie stworzony do wysokowydajnych obliczeń. Jego zaletą jest to że kod napisany raz przekąpiluje się na wiele różnych platform.

3 Przebieg procesu w aplikacji

Aplikacja została zaprojektowana z myślą o prostocie i intuicyjności obsługi. Interakcja użytkownika z systemem przebiega w kilku logicznych krokach:

1. **Uwierzytelnianie:** Po uruchomieniu aplikacji, użytkownik jest proszony o podanie danych uwierzytelniających do API Sentinel Hub. Aplikacja bezpiecznie przechowuje te dane w zaszyfrowanym pliku, aby przy kolejnych uruchomieniach wymagać jedynie podania hasła.

2. **Główny widok mapy:** Po pomyślnym zalogowaniu, użytkownik przechodzi do głównego interfejsu, który składa się z dwóch głównych paneli: interaktywnej mapy satelitarnej oraz panelu na wyniki.

3. **Wybór obszaru i parametrów:** Użytkownik może swobodnie nawigować po mapie (przesuwać, przybliżać), aby wybrać interesujący go obszar. W górnym panelu dostępne są kontrolki do wyboru zakresu dat oraz metody przetwarzania (GPU lub CPU).

4. **Pobieranie danych:** Po kliknięciu przycisku "Fetch Data", aplikacja wysyła zapytanie do API Sentinel Hub o pasma B4, B8, B11 oraz maskę danych dla widocznego na ekranie obszaru. Pobrane dane są zapisywane w lokalnym folderze `data`, który pełni rolę pamięci podręcznej (cache), aby uniknąć ponownego pobierania tych samych danych.

5. **Obliczenia i wizualizacja:** Po pobraniu danych, aktywują się przyciski "Calculate NDVI"i "Calculate NDMI". Kliknięcie jednego z nich uruchamia obliczenia w osobnym wątku (aby nie blokować interfejsu), wykorzystując wybraną przez użytkownika metodę (GPU lub CPU). Po zakończeniu obliczeń, w prawym panelu wyświetlana jest mapa cieplna (heatmap) wskaźnika wraz z legendą.

6. **Moduł testowy:** Z widoku mapy użytkownik może przejść do dedykowanego panelu testowego, który pozwala na uruchomienie serii testów wydajnościowych i wygenerowanie szczegółowych wykresów porównujących zaimplementowane metody obliczeniowe.

4 Równoległe przetwarzanie

4.1 Które obliczenia i dlaczego postanowiliśmy zrównoleglić

Głównym kandydatem do zrównoleglenia w naszym projekcie były obliczenia wskaźników NDVI i NDMI. Decyzja ta wynikała z charakterystyki problemu:

- **Niezależność danych (Data Parallelism):** Wartość wskaźnika dla każdego piksela obrazu satelitarnego jest obliczana wyłącznie na podstawie wartości pasm dla tego samego piksela. Nie ma żadnych zależności między sąsiednimi pikselami.

- **Duża skala problemu:** Obrazy satelitarne mogą składać się z milionów, a nawet dziesiątek milionów pikseli. Wykonywanie operacji na tak dużej liczbie danych w sposób sekwencyjny może być czasochłonne.

- **Powtarzalność operacji:** Ten sam, prosty wzór matematyczny $(a-b)/(a+b)$ jest stosowany dla każdego piksela.

Te cechy sprawiają, że problem ten jest klasyfikowany jako "wstydliwie równoległy"(embarrassingly parallel), co czyni go idealnym do zastosowania technik przetwarzania równoległego w celu skrócenia czasu obliczeń.

4.2 Jak dokonaliśmy zrównoleglenia obliczeń

Aby przeprowadzić kompleksową analizę, zaimplementowaliśmy trzy różne strategie obliczeniowe:

Przetwarzanie na GPU z użyciem Taichi Lang

Taichi to język programowania osadzony w Pythonie, działający jako kompilator JIT (Just-In-Time). Pozwala on na pisanie funkcji w składni zbliżonej do Pythona, które są następnie kompilowane do wysoce zoptymalizowanego kodu maszynowego dla różnych architektur, w tym GPU (poprzez backendy takie jak CUDA, DirectX, Metal). W naszej implementacji, kernel Taichi wykonuje operację $(a-b)/(a+b)$ dla całej tablicy danych, wykorzystując tysiące rdzeni GPU do jednoczesnego przetworzenia wielu pikseli.

Równoległe przetwarzanie na CPU z użyciem ‘multiprocessing‘

Aby wykorzystać wielordzeniową architekturę nowoczesnych procesorów, użyliśmy wbudowanego w Pythona modułu `multiprocessing`. Kluczowym wyzwaniem w tym podejściu jest narzut związany z komunikacją między procesami i kopiowaniem danych. Aby go zminimalizować, zaimplementowaliśmy mechanizm oparty na **pamięci dzielonej (Shared Memory)**. Zamiast wysyłać kopie fragmentów obrazu do każdego procesu roboczego, tworzymy w pamięci RAM jeden, duży segment, do którego wszystkie procesy mają bezpośredni dostęp. Każdy proces otrzymuje jedynie informację o fragmencie (zakresie wierszy), który ma przetworzyć, co drastycznie redukuje koszt komunikacji.

Jednowątkowe przetwarzanie na CPU z użyciem NumPy

Jako punkt odniesienia, stworzyliśmy również implementację jednowątkową. Nie jest to jednak prosta pętla w Pythonie. Wykorzystaliśmy bibliotekę NumPy, której operacje na tablicach (np. dodawanie, odejmowanie) nie są wykonywane przez interpreter Pythona, ale przez wysoce zoptymalizowane, prekompilowane procedury w językach C i Fortran. Dzięki temu, operacje te są wykonywane bardzo szybko na jednym rdzeniu CPU i stanowią silny baseline do porównań z metodami równoległymi.

5 Platforma Testowa

Wszystkie testy wydajnościowe zostały przeprowadzone na jednolitej platformie sprzętowej, aby zapewnić spójność i porównywalność wyników. Poniższa specyfikacja dokumentuje konfigurację komputera użytego do testów. Zrozumienie kontekstu sprzętowego jest kluczowe dla poprawnej interpretacji wyników, ponieważ wydajność zależy od wielu czynników, takich jak moc obliczeniowa CPU i GPU, szybkość pamięci RAM czy przepustowość magistrali PCIe.

System Operacyjny: Windows 10 (AMD64)

Procesor (CPU): AMD Ryzen 9 3900X

Szczegóły CPU: 12 rdzeni fizycznych / 24 wątki, Taktowanie bazowe: 3.8 GHz, Cache L2: 6 MB, Cache L3: 64 MB

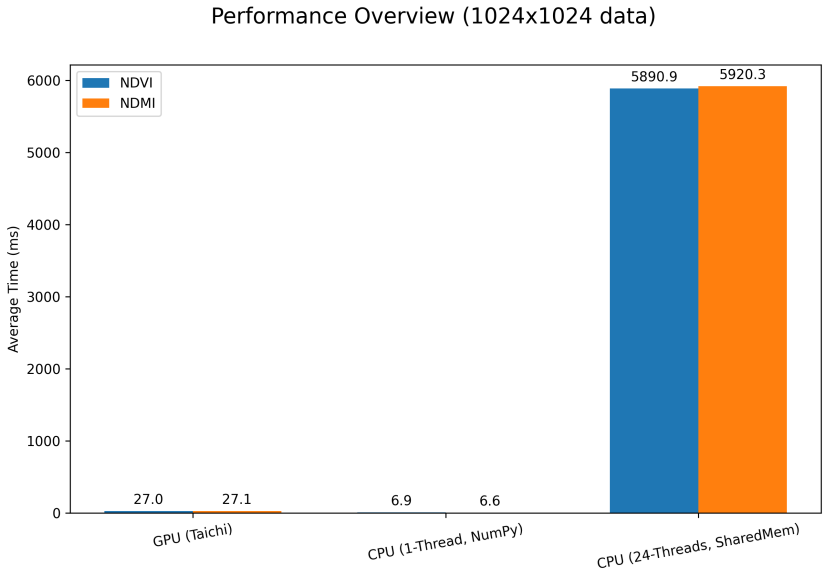
Pamięć RAM: 64 GB DDR4

Karta Graficzna (GPU): NVIDIA GeForce RTX 2070 SUPER

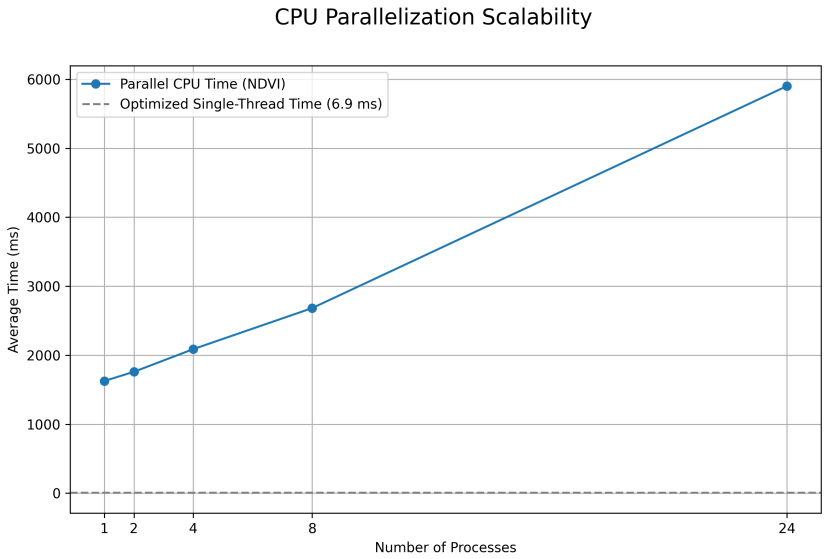
Uwaga: Na wydajność, zwłaszcza w przypadku obliczeń na GPU, mają również wpływ czynniki nieujęte w powyższej specyfikacji, takie jak wersja i przepustowość magistrali PCIe, szybkość pamięci VRAM oraz ogólne obciążenie systemu podczas testów.

6 Jakie wyniki uzyskaliśmy

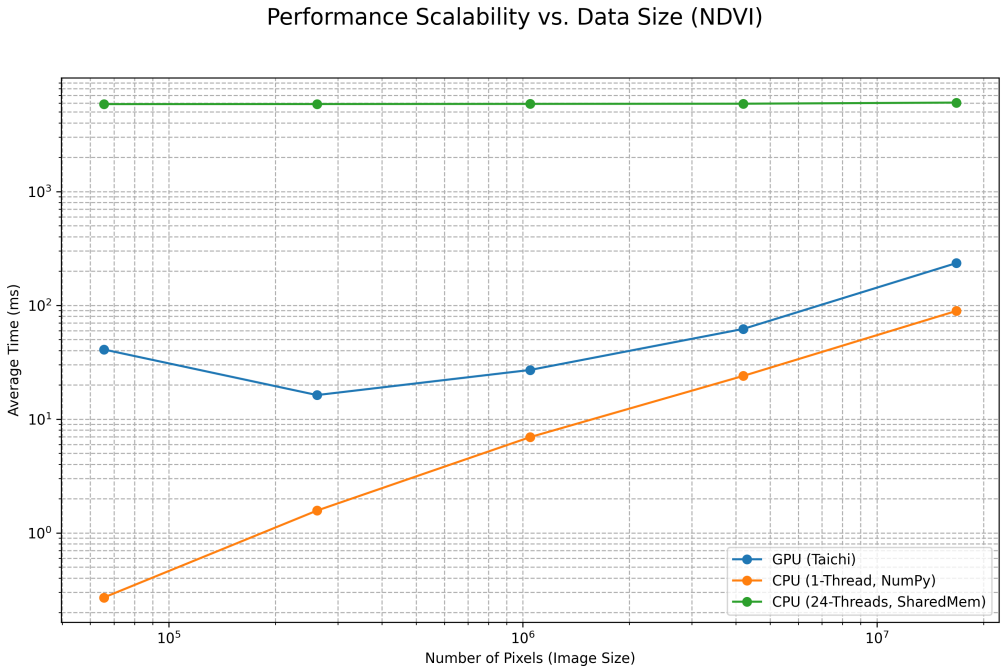
W celu oceny wydajności zaimplementowanych metod, przeprowadziliśmy serię testów na dedykowanej platformie testowej. Testy polegały na mierzeniu czasu wykonania obliczeń dla danych o różnej rozdzielczości oraz analizie zużycia pamięci. Poniższe wykresy prezentują zebrane wyniki.



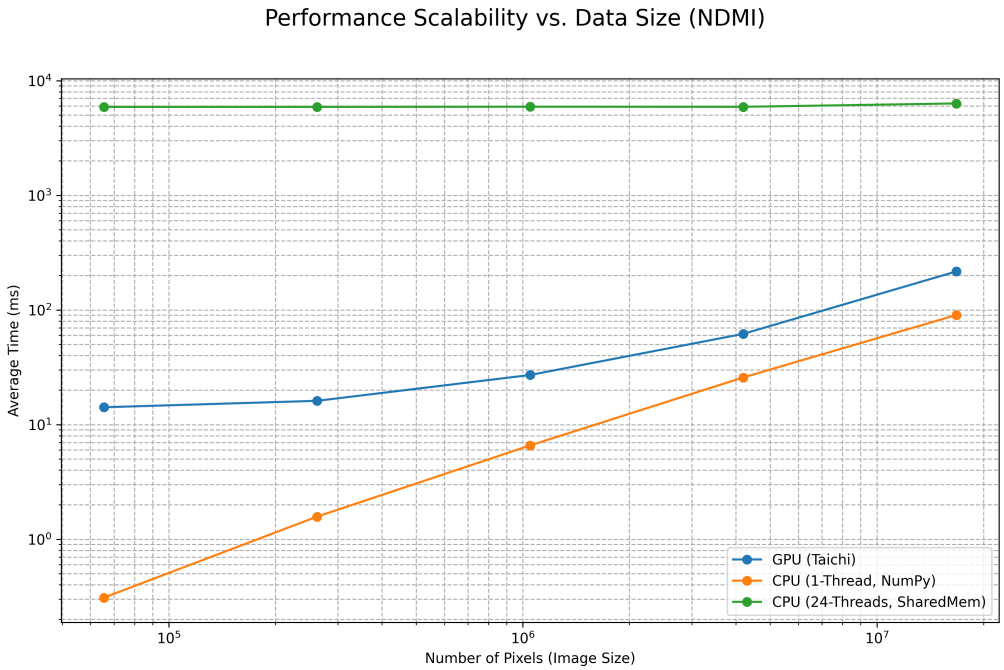
Rysunek 1: Przegląd wydajności dla danych 1024x1024.



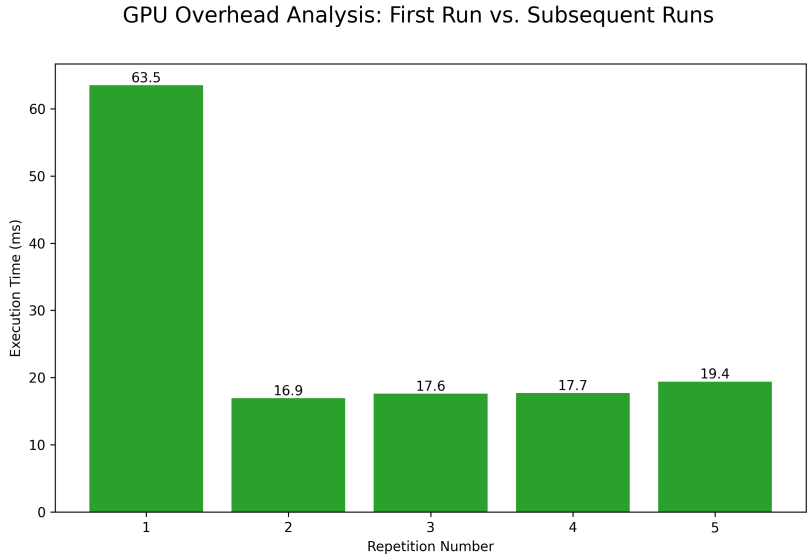
Rysunek 2: Analiza skalowalności implementacji równoległej na CPU.



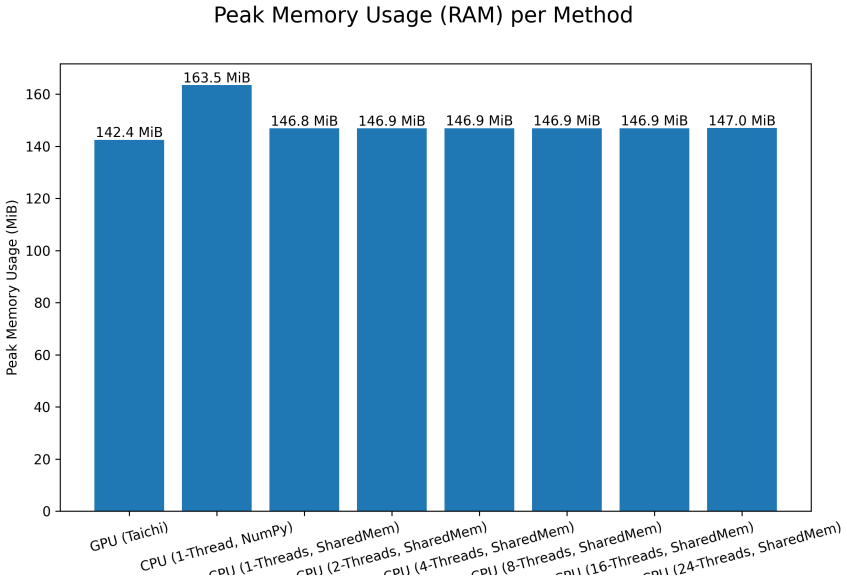
Rysunek 3: Skalowalność wydajności w zależności od rozmiaru danych (NDVI).



Rysunek 4: Skalowalność wydajności w zależności od rozmiaru danych (NDMI).



Rysunek 5: Analiza narzutu inicjalizacyjnego (warm-up) dla GPU.



Rysunek 6: Szczytowe zużycie pamięci RAM przez proces główny.

7 Interpretacja i wnioski

Przeprowadzone testy dostarczyły szeregu wnikliwych obserwacji, które pozwalają na sformułowanie kluczowych wniosków dotyczących wydajności i zasadności stosowania różnych technik przetwarzania równoległego dla analizowanego problemu.

7.1 Wniosek 1: Zoptymalizowany CPU jest najszybszy dla małych i średnich zadań

Jak widać na Rysunku 1, dla standardowego zadania (obraz 1024x1024), wysoce zoptymalizowana, jednowątkowa implementacja oparta na NumPy jest bezkonkurencyjnie najszybsza. Dzieje się tak, ponieważ obliczenia są na tyle proste, że koszt narzutu (overhead) związany z transferem danych do GPU lub zarządzaniem pulą procesów na CPU jest znacznie większy niż zysk z samego zrównoleglenia.

7.2 Wniosek 2: Przetwarzanie równoległe na CPU ponosi znaczący koszt narzutu

Rysunek 2 doskonale ilustruje problem narzutu w ‘multiprocessing‘. Czas wykonania dla jednego procesu w puli jest drastycznie wyższy niż dla wersji jednowątkowej bez puli. Ten ogromny koszt wynika z konieczności tworzenia nowych procesów, konfiguracji pamięci dzielonej i komunikacji międzyprocesowej. Mimo że implementacja poprawnie skaluje się wraz z dodawaniem kolejnych procesów (czas maleje), nigdy nie jest w stanie zniwelować początkowego narzutu i pokonać wydajności czystego NumPy dla tego zadania.

7.3 Wniosek 3: GPU staje się opłacalne dopiero przy odpowiednio dużym rozmiarze problemu

Analiza skalowalności w zależności od rozmiaru danych (Rysunki 3 i 4) jest najważniejszym wynikiem projektu. Obserwujemy, że linia wydajności GPU (niebieska) ma mniejsze nachylenie niż linia CPU (pomarańczowa), co oznacza, że lepiej radzi sobie z rosnącym obciążeniem. Widzimy wyraźny **punkt przełamania (break-even point)** w okolicach $2 \cdot 10^6$ pikseli. Poniżej tego progu, zoptymalizowany CPU jest szybszy. Powyżej tego progu, zysk z masowego zrównoleglenia na GPU zaczyna przewyższać stały koszt transferu danych, czyniąc GPU metodą wydajniejszą dla bardzo dużych zbiorów danych.

7.4 Wniosek 4: Analiza narzutu GPU i zużycia pamięci

Rysunek 5 potwierdza istnienie kosztu "rozgrzewki"GPU. Pierwsze uruchomienie jest znacznie wolniejsze z powodu kompilacji JIT. Rysunek 6 pokazuje z kolei kompromisy w zarządzaniu pamięcią. Metoda GPU odciąża systemową pamięć RAM, przenosząc dane do VRAM, co może być zaletą w systemach z ograniczonymi zasobami.

7.5 Wniosek ogólny: Wybór metody zależy od kontekstu

Nie istnieje jedna, uniwersalnie najlepsza metoda. Wybór optymalnej strategii zależy od konkretnego przypadku użycia:

- **Interaktywna analiza małych obszarów:** Zdecydowanie najlepszym wyborem jest zoptymalizowany, jednowątkowy CPU (NumPy) ze względu na minimalny czas latencji.
- **Przetwarzanie wsadowe bardzo dużych obrazów:** Dla zadań przekraczających "punkt przełamania", GPU oferuje najwyższą przepustowość i jest metodą preferowaną.
- **Efektywność energetyczna:** Dla zadań, gdzie CPU jest szybsze, jest ono również znacznie bardziej efektywne energetycznie.

Nasze wyniki podkreślają, jak kluczowe jest zrozumienie charakterystyki problemu (złożoność obliczeniowa vs. obciążenie I/O) oraz narzutów platformy przy projektowaniu systemów o wysokiej wydajności.

8 Jak uruchomić program

Aby uruchomić aplikację, należy postępować zgodnie z poniższymi krokami:

1. Sklonować repozytorium projektu i przejść do jego głównego katalogu.
2. Zainstalować wszystkie wymagane biblioteki za pomocą menedżera pakietów pip:

   ```
   pip install -r requirements.txt
   ```

3. Stworzyć plik `.env` w głównym katalogu projektu. Plik ten będzie zawierał **jedynie** sól kryptograficzną, która jest niezbędna do odszyfrowania danych logowania. Dla celów testowych i odtworzenia wyników, należy użyć poniższej wartości:

   ```
   SENTINEL_SALT="385
       ↪ d0b98fb81618a8d5165a2c25300e26217c22661ca9ada7cde877075184219
       ↪ "
   ```

4. Przygotować własne dane uwierzytelniające (Client ID oraz Client Secret) do API Sentinel Hub. Instrukcje, jak je uzyskać, znajdują się w oficjalnej dokumentacji: https://docs.sentinel-hub.com/api/latest/api/overview/authentication/.
5. Przy pierwszym uruchomieniu, aplikacja poprosi o podanie uzyskanych danych oraz o hasło, które posłuży do ich zaszyfrowania i bezpiecznego przechowania w pliku `sentinel_config.enc`. Dla celów ewaluacji projektu, w repozytorium znajduje się już gotowy plik `sentinel_config.enc`, do którego hasło zostanie udostępnione osobno.
6. Uruchomić główną aplikację za pomocą polecenia z **głównego katalogu projektu**:

   ```
   python -m src.main
   ```

   Uwaga: Użycie flagi `-m` jest kluczowe, ponieważ zapewnia, że Python poprawnie rozpoznaje strukturę pakietów w folderze `src`.

8.1 Uruchamianie skryptów testowych (opcjonalne)

Aby przeprowadzić testy wydajności i pamięci, należy najpierw wygenerować dane testowe, a następnie uruchomić skrypt profilujący. Oba skrypty należy uruchamiać z **głównego katalogu projektu**.

1. Generowanie danych testowych o różnej skali:

   ```
   python -m src.scripts.generate_scaled_data
   ```

2. Uruchomienie testów zużycia pamięci (wymaga `memory-profiler`):

   ```
   python -m memory_profiler src.scripts.run_memory_tests
   ```

Bibliografia

[1] Sentinel Hub. *Normalized Difference Vegetation Index (NDVI)*. URL: https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/ndvi/ (term. wiz. 11.06.2024).

[2] Sentinel Hub. *Normalized Difference Moisture Index (NDMI)*. URL: https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/ndmi/ (term. wiz. 11.06.2024).

[3] Sentinel Hub. *Sentinel-2 Band Documentation B4*. URL: https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/bands/#band-4 (term. wiz. 11.06.2024).

[4] Sentinel Hub. *Sentinel-2 Band Documentation B8*. URL: https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/bands/#band-8 (term. wiz. 11.06.2024).

[5] Sentinel Hub. *Sentinel-2 Band Documentation B11*. URL: https://custom-scripts.sentinel-hub.com/custom-scripts/sentinel-2/bands/#band-11 (term. wiz. 11.06.2024).

[6] Sentinel Hub. *Sentinel-2 API Documentation*. URL: https://docs.sentinel-hub.com/api/latest/data/sentinel-2-l2a/ (term. wiz. 11.06.2024).

[7] Sentinel Hub. *Sentinel-2 API Documentation Authentication*. URL: https://docs.sentinel-hub.com/api/latest/api/overview/authentication/ (term. wiz. 11.06.2024).

[8] Sentinel Hub. *Sentinel-2 API Documentation Rate Limiting*. URL: https://docs.sentinel-hub.com/api/latest/api/overview/rate-limiting/ (term. wiz. 11.06.2024).

[9] Taichi Lang. *Taichi Lang Documentation*. URL: https://www.taichi-lang.org/ (term. wiz. 11.06.2024).