UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **Geoinformatyka**
Specjalność:

Michał Jan Wilkos
Nr albumu: 307370

Projekt i implementacja oprogramowania do analizy przestrzenno-czasowej danych treningowych z urządzeń GPS

Design and implementation of software for spatial-temporal analysis of training data from GPS devices

Praca magisterska
napisana w Katedrze Oprogramowania Systemów Informatycznych
Instytutu Informatyki
pod kierunkiem dr Michała Klisowskiego

Lublin 2026

Spis treści

Wstęp

W czasach rosnącej świadomości na temat zdrowego trybu życia oraz dynamicznego rozwoju technologii pomiaru sesji treningowych coraz więcej osób decyduje się na monitorowanie swoich aktywności fizycznych, a popularność zegarków sportowych i urządzeń z modułami GPS rośnie z roku na rok. Jednak generowanie ogromnych ilości danych o przebytych dystansach i parametrach treningowych niesie ze sobą problemy nie tylko w kontekście interpretacji, ale przede wszystkim obliczeń, szczególnie w przypadku przetwarzania informacji przestrzenno-czasowych.

W procesie tworzenia oprogramowania analitycznego ważną rolę odgrywa wybór odpowiedniej metody przetwarzania dużych zbiorów danych. Chociaż język Python oferuje mnogość narzędzi, wiele rozwiązań różni się pod względem efektywności czasowej i pamięciowej. Często stosowane standardowe struktury danych, takie jak listy, mogą okazać się niewystarczające przy skomplikowanych i obciążających pamięć obliczeniach. Te zagadnienia oraz chęć sprawdzenia różnic w wydajności stanowiły inspirację do stworzenia aplikacji, która posłuży jako środowisko testowe dla tych metod.

Praca będzie się koncentrować na opracowaniu oraz zaimplementowaniu oprogramowania do analizy przestrzenno-czasowej danych treningowych, z uwzględnieniem porównania efektywności różnych metod algorytmów analitycznych. Głównym założeniem jest nie tylko stworzenie funkcjonalnego narzędzia, ale także zbadanie i zestawienie wydajności operacji wykonywanych na podstawowych strukturach danych w porównaniu do rozwiązań wektorowych.

Struktura pracy zostanie podzielona na trzy rozdziały. Pierwszy z nich omówi wykorzystaną technologię oraz charakterystykę przetwarzania danych GPS, ze wskazaniem na różnice pomiędzy klasycznym modelem programowania a podejściem wektorowym. Drugi rozdział będzie przedstawiać założenia projektowe aplikacji oraz metodologię przeprowadzania testów porównawczych. W ostatnim rozdziale przedstawione zostaną aspekty implementacyjne, w tym organizacja kodu oraz napotkane problemy, a jego ważną część będzie stanowić prezentacja wyników analizy porównawczej wydajności zastosowanych rozwiązań w odniesieniu do czasu przetwarzania i wykorzystania zasobów systemowych.

Rozdział 1

Teoria (do poprawy)

1.1 Pamięć (do poprawy)

1.1.1 Pamięć podręczna i hierarchia pamięci

Współczesne procesory wykonują obliczenia szybciej, niż system pamięci jest w stanie dostarczyć im dane. Jest to problem znany jako „wąskie gardło" występujący w architekturze von Neumanna, według której dane są przechowywane w pamięci wraz z instrukcjami kodu programu[1]. Problem ten określa się również jako ścianę pamięci (ang. *memory wall*). To sprawia, że w aplikacjach przetwarzających duże ilości danych problemem staje się opóźnienie w dostępie do pamięci RAM, a nie moc obliczeniowa rdzenia.

Aby złagodzić ten problem, architektury komputerowe wykorzystują wielopoziomową pamięć podręczną, która stanowi swoisty bufor pomiędzy procesorem a pamięcią RAM. Dzieli się ona na trzy poziomy, z czego pierwszy z nich jest najszybszy, a trzeci – najwolniejszy[1]. Jest to mniejsza pamięć, ale znacznie szybsza od pamięci operacyjnej komputera. W celu uzyskania wysokiej wydajności często zarządza się danymi tak, aby procesor jak najczęściej znajdował potrzebne informacje w pamięci podręcznej, ograniczając kosztowne pobrania z pamięci głównej RAM.

Fundamentem wydajnego wykorzystania pamięci podręcznej jest zasada lokalności odwołań:

- Lokalność czasowa: Dane, do których uzyskano dostęp, prawdopodobnie będą potrzebne ponownie w niedługim czasie.
- Lokalność przestrzenna: Jeśli program odwołuje się do określonego adresu pamięci, z dużym prawdopodobieństwem wkrótce odwoła się do sąsiednich adresów[2].

Z perspektywy programisty i wyboru struktur danych w projekcie, lokalność przestrzenna ma bardzo ważne znaczenie. Procesory nie pobierają z pamięci pojedynczych bajtów, lecz całe linie pamięci podręcznej. Struktury takie jak tablice dynamiczne, gdzie

elementy są ułożone w pamięci w kolejności jeden za drugim, są pod tym względem bardzo wydajne. Pobranie pierwszego elementu automatycznie ładuje do cache'u kolejne, co pozwala na płynne dostarczanie danych[5].

Struktury oparte na węzłach i wskaźnikach, takie jak listy połączone lub drzewa charakteryzują się rozrzuceniem elementów po stercie pamięci. Przejście do kolejnego elementu wymaga skoku do losowego adresu, co często skutkuje błędem braku strony w pamięci podręcznej i wstrzymaniem pracy procesora na czas pobrania danych z RAM[6].

1.1.2 Równoległe wykonywanie operacji

Przez dekady wzrost wydajności aplikacji opierał się na zwiększaniu częstotliwości taktowania procesorów, lecz skok technologiczny sprawił, że dalsze zwiększanie prędkości pojedynczego rdzenia stało się nieefektywne. Zamiast tego, zaczęto zwiększać liczbę rdzeni w procesorach, co doprowadziło do zmiany w projektowaniu systemów i aplikacji – przyspieszania pojedynczych instrukcji w stronę równoległych obliczeń[4].

Do klasyfikacji architektur równoległych stosuje się taksonomię zaproponowaną przez Michaela Flynna w 1966 roku

- SISD (ang. *Single Instruction, Single Data*): Architektura von Neumanna, gdzie jeden procesor wykonuje jedną instrukcję na jednej danej. Jest to model sekwencyjny.
- SIMD (ang. *Single Instruction, Multiple Data*): Jedna instrukcja sterująca wykonuje tę samą operację na wielu elementach danych jednocześnie. Jest to podstawa wektoryzacji.
- MISD (ang. *Multiple Instruction, Single Data*): Wiele instrukcji operuje na jednej danej.
- MIMD (ang. *Multiple Instruction, Multiple Data*): Wiele procesorów lub rdzeni wykonuje niezależnie różne instrukcje na różnych danych. Jest to podstawowy model równoległości[3].

Architektura MIMD jest realizowana w obecnych komputerach za pomocą procesorów wielordzeniowych. W kontekście analizy danych treningowych pozwala to na podział pracy na mniejsze, niezależne zadania. System może analizować kilka niezależnych plików jednocześnie, gdzie każdy plik obsługiwany jest przez osobny wątek procesora. W związku z tym system może działać w przykładowy sposób:

- Rdzeń 1 analizuje trasę biegu z poniedziałku.
- Rdzeń 2 analizuje trasę rowerową z wtorku.
- Rdzeń 3 analizuje spacer ze środy.

Ponieważ wynik analizy jednego działania nie zależy od wyniku drugiego, rdzenie te nie muszą się ze sobą komunikować ani na siebie czekać.

Bibliografia

[1] Michał Malinowski, *Architektura komputera*, https://www.drmalinowski.edu.pl/posts/3094-architektura-komputera, [dostęp: 14.01.2026]

[2] Arkadiusz Chrobot, *Systemy Operacyjne — Pamięć wirtualna cz. 1*, https://achilles.tu.kielce.pl, [dostęp: 14.01.2026]

[3] Luka Leskovec, *Flynn's Taxonomy: A Guide To Computer Architectures*, https://doc.sling.si/en/hpc-guide/02-computer-architecture/01-flynns-taxonomy/01-flynns-taxonomy/, [dostęp: 14.01.2026]

[4] Zbigniew Koza, *Komputery równoległe*, http://www.ift.uni.wroc.pl/koma/pr/2012/wyklad1.pdf, [dostęp: 14.01.2026]

[5] Jerzy Wałaszek, *Wskaźniki i tablice dynamiczne*, https://eduinf.waw.pl/inf/utils/010_2010/0513.php, [dostęp: 14.01.2026]

[6] Grzegorz Jagiella, *Listy łączone, hermetyzacja*, https://math.uni.wroc.pl/jagiella/p2python/skrypt_html/wyklad4-2.html, [dostęp: 14.01.2026]