|  |  |
| --- | --- |
| Wojciech Cwojdziński | 95959 |
| Liliana Jaśkiewicz | 95098 |
| Informatyka inż III semestr | Lab 7, 19.11.2024r. |

zad1

Skupiliśmy się na analizie i zrozumieniu kluczowych zagadnień związanych z przetwarzaniem rownoległym. Rozpocząłem od porównania tego typu przetwarzania z przetwarzaniem sekwencyjnym zauważając dużo różnic pomiędzy jednym a drugim a szczególnie w sposobie realizacji zadan, wydajności oraz zarządzaniu zasobami. Następnie zapoznaliśmy się z zastosowaniami przetwarzania równoległego w różnych dziedzinach, takich jak obliczenia naukowe, grafika komputerowa, analiza dużych zbiorów danych, a także w telekomunikacji czy blockchainie. Omówiliśmy również korzyści, takie jak zwiększona wydajność i lepsza skalowalność, oraz wyzwania, takie jak synchronizacja danych czy zarządzanie współbieżnością. W trakcie pracy skupiłem się na zrozumieniu, jak taka technologia mogą być stosowana w praktyce oraz jakie trudności mogą pojawić się podczas implementacji.

zad2

W trakcie realizacji zadania nr2 nauczyliśmy się, jak obliczać sumę kwadratów liczb na jednym rdzeniu procesora oraz na wielu rdzeniach. Zaczynając od obliczeń na jednym rdzeniu, zaimplementowałem funkcję, która iteruje przez liczby w zadanym zakresie, obliczając ich kwadraty i sumując je. Następnie, aby przyspieszyć obliczenia, użyliśmy modułu `multiprocessing`, aby podzielić zakres liczb na cztery części i przypisać każdą z nich do osobnego procesu. W tym celu stworzyliśmy funkcję `podziel\_i\_oblicz`, która dzieli zakres na fragmenty, a następnie uruchamia równoległe procesy za pomocą `Pool` i `starmap`. Każdy proces oblicza sumę kwadratów dla swojego fragmentu, a wyniki są sumowane, co pozwala na szybsze przetwarzanie danych.

zad3

W trakcie przepisywania kodu zidentyfikowaliśmy kilka błędów które uniemożliwiały jego poprawne działanie. W funkcji monte\_carlo\_simulation wystąpił błąd w pętli for gdzie brakowało zmiennej, co wymagało zamiany na for \_ in range(num\_points). Kolejnym problemem było przekazywanie argumentów w funkcji pool.map, gdzie zapis [points\_per\_process] num\_processes był nieprawidłowy, a właściwe rozwiązanie polegało na utworzeniu listy `[points\_per\_process] \* num\_processes`, przypisując każdemu procesowi odpowiednią liczbę punktów. W obliczeniach wartości liczby pi błąd wynikał z niewłaściwego traktowania zmiennej inside\_circle która powinna być sumą wyników z wszystkich procesów co naprawiono przez odpowiednie zsumowanie wyników w funkcji parallel\_monte\_carlo. Dodatkowo w linii wyświetlania wyniku liczby pi, występował nieprawidłowy f-string, wymagający poprawy na: f"Przybliżona wartość liczby pi: {pi\_estimate}". Na koniec usunięto zbędne duplikaty obliczeń, aby uniknąć powtarzania tych samych operacji kilkukrotnie. Dzięki tym zmianom kod został poprawiony i działa zgodnie z założeniami. Kod implementuje metodę Monte Carlo do przybliżenia wartości liczby pi, wykorzystując wieloprocesowość w celu przyspieszenia obliczeń. Generuje losowe punkty w kwadracie jednostkowym, sprawdza, ile z nich znajduje się wewnątrz koła, i na tej podstawie podaje pi. Całość jest podzielona na równoległe zadania, które są przetwarzane przez wiele procesów, a wyniki są sumowane. Tak uzyskujemy końcowe przybliżenie liczby pi.