Sprawozdanie

Obliczenia równoległe i rozproszone – Laboratorium nr 1

Prowadzący: mgr inż. Michał Zimoń

Wykonawcy: Mateusz Gajda, Kuba Kaczmarski, Radosław Relidzyński

Grupa: WCY20IJ1S1

# 1. Wstęp

**Cel zadania:**

Wykorzystanie znajomości w zakresie obliczeń równoległych w realizacji przykładowego problemu.

**Opis danych:**

Dane stanowią kolejne liczby całkowite z zadeklarowanego zakresu poprzez pierwszą liczbę i ilość liczb w zbiorze.

**Oczekiwany rezultat:**

Algorytm, który działa równolegle, przetworzy dane znacznie szybciej niż ten, który działa sekwencyjnie.

# 2. Metodologia

**Narzędzia i technologie:**

Język Python, z wykorzystaniem biblioteki multiprocessing.

**Opis algorytmu sekwencyjnego:**

Wywołanie funkcji przechodzącej po każdej liczbie z zakresu i sprawdzającej czy jest podzielna przez dany dzielnik. Zlicza wszystkie przypadki, gdzie liczba pasuje do zakresu.

**Opis algorytmu równoległego:**

Zbiór liczb do sprawdzenia jest dzielony równo pomiędzy daną liczbę procesów, każda dostaje zakres, dla którego ma zliczyć ilość liczb podzielnych. Na końcu zbiór wyników jest sumowany do jednej wartości, który jest wynikiem całej operacji dla całego zbioru.

# 3. Implementacja

**Kod:**

import time  
import multiprocessing  
  
  
def count\_divisible(divisor, start, end):  
 count = 0  
 for i in range(start, end + 1):  
 if not i % divisor:  
 count += 1  
 return count  
  
  
def count\_serial(divisor, start, n):  
 return count\_divisible(divisor, start + 1, start + n)  
  
  
def count\_parallel(divisor, start, n, num\_processes):  
 # specify size of data for one process  
 chunk\_size = n // num\_processes  
 # create a pool of processes to spread calculation between them  
 pool = multiprocessing.Pool(processes=num\_processes)  
 # create list of numbers ranges to specify for every process  
 ranges = [(divisor, (start + i \* chunk\_size + 1), (start + (i + 1) \* chunk\_size)) for i in range(num\_processes)]  
 # for every process run function, giving ranges list for each process  
 results = pool.starmap(count\_divisible, ranges)  
 # summarize processes efforts to one output  
 result = sum(results)  
 return result  
  
  
def run\_calculations(divisor, start, nx):  
  
 start\_time = time.time()  
 result = count\_serial(divisor, start, nx)  
 end\_time = time.time()  
 print(f"serial n={nx} result={result} time={end\_time-start\_time}")  
  
 for num\_processes in [2, 4]: # different amount of processes  
 start\_time = time.time()  
 result = count\_parallel(divisor, start, nx, num\_processes)  
 end\_time = time.time()  
 print(f"parallel{num\_processes} n={nx} result={result} time={end\_time-start\_time}")  
  
  
def main():  
  
 # input  
 start = 1000  
 n = [10 \*\* i for i in range(5, 9)] # different numbers amount  
 divisors = [2, 3, 5]  
  
 x = 1  
 for divisor in divisors:  
 for nx in n:  
 print(f"\nAttempt {x} n={nx} divisor={divisor}")  
 run\_calculations(divisor, start, nx)  
 nx = int(nx/10)  
 x += 1  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Struktura kodu:**

* main()– zawiera dane wejściowe i wywołuje funkcje dla różnych wartości dzielnika i zakresu liczb
* run\_calculations(divisor, start, nx)– uruchamia obliczanie sekwencyjne oraz równoległe dla różnych ilości procesów. Dodatkowo mierzy czas działania każdego wywołania funkcji
* count\_parallel(divisor, start, n, num\_processes) - tworzy pulę procesów, dzieli między nimi zadanie obliczeniowe, uruchamia te procesy, sumuje wyniki działań procesów i zwraca wynik końcowy
* count\_serial(divisor, start, n) – wywołuje funkcję zliczania liczb podzielnych przez dany dzielnik dla pełnego zakresu
* count\_divisible(divisor, start, end) – oblicza ilość liczb podzielnych przez dany dzielnik w danym zakresie od start do end, wywoływane zarówno przez obliczanie sekwencyjne jak i równoległe

Wykorzystane techniki optymalizacji: Dzielenie zadania obliczeniowego między procesy. W obliczaniu równoległym każdy proces dostaje część zadania do obliczenia dla siebie, a gdy wszystkie skończą swoje działanie ich wyniki są zbierane do jednej postaci. Procesy te działają równolegle, przez co wykonują równocześnie wiele obliczeń na raz co sprzyja zmniejszaniu czasu wykonania algorytmu.

Problemy napotkane podczas implementacji:

* Problem przygotowania i przekazywania danych – dzieląc zakres między procesy należy zadbać o to, żeby dwa procesy nie robiły obliczeń dla tej samej wartości oraz, żeby nie było wartości, która nie została wliczona
* Problem ilości operacji – dla zbyt małych ilości operacji czas wykonania obliczeń z wykorzystaniem obliczania równoległego był wielokrotnie większy. Dopiero przy odpowiednio dużej liczbie operacji obliczanie równoległe wygrywa czasowo z obliczaniem sekwencyjnym, szczególnie przy większej liczbie procesów.

# 4. Wyniki

**Porównanie czasów wykonania:**

Zestawienie czasów w tabelach:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Dzielnik | 2 |  |
| n | Ts | Tp2 | Tp4 |
| 10^5 | 0,0040 | 0,1365 | 0,1340 |
| 10^6 | 0,0480 | 0,1430 | 0,1765 |
| 10^7 | 0,4700 | 0,3640 | 0,2360 |
| 10^8 | 4,7008 | 2,7340 | 1,5920 |
|  |  |  |  |
|  |  |  |  |
|  | Dzielnik | 3 |  |
| n | Ts | Tp2 | Tp4 |
| 10^5 | 0,0050 | 0,1220 | 0,1360 |
| 10^6 | 0,0490 | 0,1360 | 0,1690 |
| 10^7 | 0,4810 | 0,3971 | 0,3260 |
| 10^8 | 4,5131 | 2,5397 | 1,4970 |
|  |  |  |  |
|  |  |  |  |
|  | Dzielnik | 5 |  |
| n | Ts | Tp2 | Tp4 |
| 10^5 | 0,0050 | 0,1180 | 0,1550 |
| 10^6 | 0,0530 | 0,1355 | 0,1445 |
| 10^7 | 0,4940 | 0,3501 | 0,2700 |
| 10^8 | 4,8788 | 3,5997 | 1,5978 |

Zestawienie czasów na wykresach:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Legenda:

* n – wielkość zakresu
* Ts – czas dla algorytmu sekwencyjnego
* Tp2 – czas dla algorytmu równoległego z wykorzystaniem 2 procesów
* Tp4 – czas dla algorytmu równoległego z wykorzystaniem 4 procesów

**Analiza przyspieszenia:**

Wzór na przyspieszenie:

Gdzie:

* p – liczba procesorów
* T1 – czas wykonania algorytmu sekwencyjnego
* Tp – czas wykonania algorytmu równoległego przy użyciu p procesorów

Wzór na efektywność:

gdzie:

* p – liczba procesorów
* Sp – przypieszenie dla p procesorów

Wartości przyspieszenia i efektywności dla mierzonych wartości:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Dzielnik | 2 |  |  |  |  |  |
| n | Ts | Tp2 | **S2** | **E2** | Tp4 | **S4** | **E4** |
| 10^5 | 0,0040 | 0,1365 | **0,0293** | **0,0146** | 0,1340 | **0,0298** | **0,0075** |
| 10^6 | 0,0480 | 0,1430 | **0,3358** | **0,1679** | 0,1765 | **0,2721** | **0,0680** |
| 10^7 | 0,4700 | 0,3640 | **1,2914** | **0,6457** | 0,2360 | **1,9918** | **0,4979** |
| 10^8 | 4,7008 | 2,7340 | **1,7194** | **0,8597** | 1,5920 | **2,9528** | **0,7382** |
|  |  |  |  |  |  |  |  |
|  | Dzielnik | 3 |  |  |  |  |  |
| n | Ts | Tp2 | **S2** | **E2** | Tp4 | **S4** | **E4** |
| 10^5 | 0,0050 | 0,1220 | **0,0410** | **0,0205** | 0,1360 | **0,0368** | **0,0092** |
| 10^6 | 0,0490 | 0,1360 | **0,3603** | **0,1801** | 0,1690 | **0,2899** | **0,0725** |
| 10^7 | 0,4810 | 0,3971 | **1,2114** | **0,6057** | 0,3260 | **1,4755** | **0,3689** |
| 10^8 | 4,5131 | 2,5397 | **1,7770** | **0,8885** | 1,4970 | **3,0147** | **0,7537** |
|  |  |  |  |  |  |  |  |
|  | Dzielnik | 5 |  |  |  |  |  |
| n | Ts | Tp2 | **S2** | **E2** | Tp4 | **S4** | **E4** |
| 10^5 | 0,0050 | 0,1180 | **0,0424** | **0,0212** | 0,1550 | **0,0323** | **0,0081** |
| 10^6 | 0,0530 | 0,1355 | **0,3911** | **0,1955** | 0,1445 | **0,3667** | **0,0917** |
| 10^7 | 0,4940 | 0,3501 | **1,4110** | **0,7055** | 0,2700 | **1,8298** | **0,4575** |
| 10^8 | 4,8788 | 3,5997 | **1,3553** | **0,6777** | 1,5978 | **3,0534** | **0,7633** |

**Wnioski z analizy wyników:**

Poprzez analizę wykresów można zobaczyć, że dla odpowiednio małych ilości obliczeń algorytm sekwencyjny wykonuje się szybciej. Natomiast dla odpowiednio dużych ilości obliczeń wygrywa algorytm równoległy. Dodatkowo algorytm działa skuteczniej dla większej ilości procesów.

Wnioski te potwierdzają obliczone wartości przyspieszenia i efektywności, przyspieszenie większe od 1 oznacza, że algorytm równoległy działa szybciej.

Jeśli chodzi o efektywność, to pomimo tego, że algorytm dla 4 procesów działa szybciej, to procesy te były znacznie mniej efektywne niż dla 2 procesów.

Wnioski te wskazują na to, że algorytm równoległy działa zgodnie z oczekiwaniami.

# 5. Dyskusja

**Możliwe optymalizacje:**

Dla lepszej optymalizacji należałoby wprowadzić rozróżnienie w ramach którego można dobierać odpowiedni algorytm do skali problemu wraz z odpowiednią liczbą procesów dla algorytmu równoległego.

Co do samego algorytmu zliczającego liczby podzielne przez dany dzielnik, mając kolejne liczby całkowite można zastosować wzór do obliczenia ilości liczb w zakresie bez sprawdzania każdej po kolei (na podstawie wiedzy o ciągach liczbowych).

**Ograniczenia metody:**

Ograniczeniem jest ilość obliczeń, dla odpowiednio małych algorytm równoległy działa nieefektywnie i można go zastąpić algorytmem sekwencyjnym, który lepiej się sprawdzi w takiej sytuacji.

# 6. Wnioski końcowe

**Podsumowanie:**

W ramach ćwiczenia udało się stworzyć algorytm rozwiązujący problem zliczania liczb podzielnych przez dany dzielnik w danym zakresie. Zastosowane różne próby pokazały kiedy szybszy jest algorytm równoległy, a kiedy sekwencyjny oraz jak ilość procesów w algorytmie równoległym wpływa na ich efektywność.

**Sugestie na przyszłość:**

Można zastosować więcej prób w celu dokładniejszego zaobserwowania skuteczności zastosowanego algorytmu do danej ilości obliczeń do wykonania.