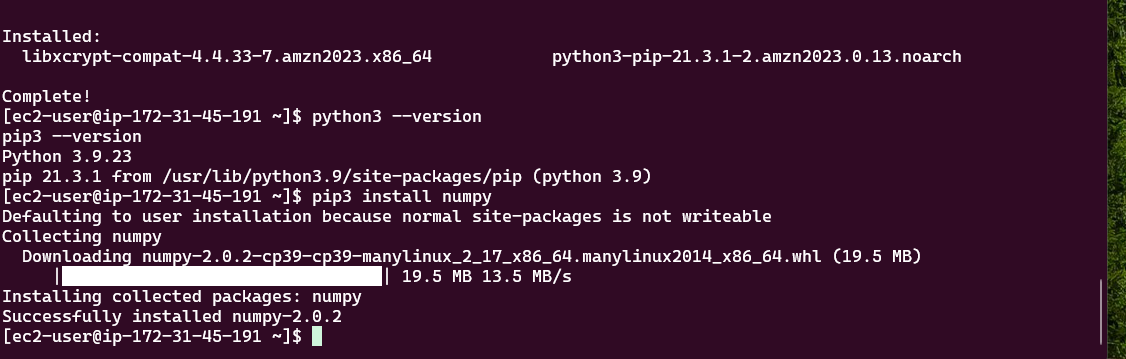
Ex\_4\_pd4660

Zadanie 4 zrobiłem w AWS dla porównania



2. Wprowadzenie do obliczeń równoległych z multiprocessing:

a. Uczycie się, jak używać biblioteki multiprocessing do prostych obliczeń równoległych.

Przykładem może być obliczenie sumy kwadratów liczb w równoległych procesach.

Porównałem sekwencyjne i równoległe obliczenia:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Wysuwa się przewaga równoległego, co byłoby bardziej widać na większych danych

#Przykładowy kod

Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Tutaj zmieniłem numer of processes do n=1 istart\_end\_pairs = [

(1, 400000)

]

Tutaj start\_end\_pairs = [

(1, 1000000),

(1000001, 2000000),

(2000001, 3000000),

(3000001, 4000000)

], czyli w porównaniu 1 proces do 4 procesów, 1 jest znacznie wolniejszy

Spróbowałem też dynamicznego dzielenia procesów:

cat <<EOF > parallel\_chunks.py

import multiprocessing

import time

def sum\_of\_squares(start, end):

return sum(x \* x for x in range(start, end + 1))

def chunk\_ranges(start, end, num\_chunks):

chunk\_size = (end - start + 1) // num\_chunks

return [(start + i \* chunk\_size, start + (i + 1) \* chunk\_size - 1) for i in range(num\_chunks - 1)] + \

[(start + (num\_chunks - 1) \* chunk\_size, end)]

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = time.time()

total\_start = 1

total\_end = 4000000

num\_processes = 4

ranges = chunk\_ranges(total\_start, total\_end, num\_processes)

pool = multiprocessing.Pool(processes=num\_processes)

results = pool.starmap(sum\_of\_squares, ranges)

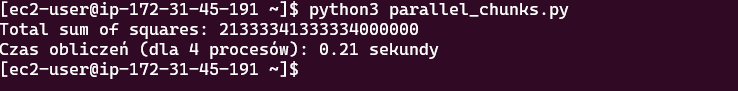
total\_sum = sum(results)

end\_time = time.time()

print(f"Total sum of squares: {total\_sum}")

print(f"Czas obliczeń (dla {num\_processes} procesów): {end\_time - start\_time:.2f} sekundy")

EOF



Zgodnie z poleceniem, potestowałem Queue i Lock:

Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez AI może być niepoprawna.

Następnie porównałem czasy (benchmark)

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Wyniki mnie nieco zaskoczyły, ale pewnie są wynikiem specyfiki danych. Dane są małe, więc czas startu procesów multi może być większy niż zysk z ich użycia (nadmierna komplikacja). Zapewne w przypadku dużych danych byłoby inaczej. A threading z tego co przeczytałem sprawdza się bardziej w zadaniach I/O, (czekających, nie liczących intensywnie) nie GPU. Zadania I/O to pobieranie danych z internetu, czytanie dużych plików z dysku itp.

Sprawdźmy zatem co będzie z dużym plikiem, wygenerowałem większy plik (1 mln liczb) i zobaczmy co się stanie:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez AI może być niepoprawna.

Niestety nadal bez zmian, pewnie potrzebowałbym większego pliku plus szybkiego dysku (a nie tego darmowego, który mam z AMS), nie mamy też żadnych transformacji, więc to sprzyja myśleniu single.

Z racji, że komentowałem tekst na bieżąco, odniosę się tylko do punktów, których nie poruszyłem.

W trakcie ćwiczeń przeszliśmy przez jednowątkowość, threading i multiprocessing. Podstawowymi zaletami obliczeń równoległych w chmurze są: skalowalność, elastyczność, szybsze przetwarzanie dużych zbiorów danych. Problemem są natomiast synchronizacja danych (komunikacja, która zabiera zasoby) i koszt pamięci (rośnie zużycie RAM przy multiprocessing), czasami multiprocessing jest zbyt skomplikowany i samo jego uruchomienie kosztuje zbyt dużo przy mniejszych danych, jeśli procesy przechodzą wąśkim gardłem (I/O), to i tak nic nie przyspieszmy.

Owarte pytania:

Google Cloud czy AWS ?

Jak najlepiej monitorować zużycie zasobów ?

Czy biblioteki Pythona dobrze działają w systemach chmurowych ?

Jak zminimalizować koszt komunikacji między procesami ?

Jak możliwie najoptymalniej dobrać model do obliczeń ?