Raport
PYTHON02

Jan Kwinta

28 grudnia 2025

Wstęp

Celem ćwiczenia jest analiza wydajności różnych podejść do implementacji mnożenia macierzy w Pythonie. Mnożenie macierzy zostało wybrane ze względu na to, że jest to wymagający algorytm. W tym raporcie porównywane podejścia do jego implementacji to:

- Czysta implementacja w pythonie: funkcja z trzema zagnieżdżonymi pętlami for, operująca na listach list (macierz jako lista wierszy),
- NumPy: implementacja wykorzystująca gotowe struktury i funkcje biblioteki numpy, np operator @,
- Cython: funkcja napisana w pliku .pyx, z typowanymi zmiennymi skompilowana do kodu w C, używana jako moduł w pythonie,
- Zewnętrzna funkcja w C: czyste C skompilowane do współdzielonej biblioteki wywoływana z Pythona za pomocą ctypes.

Do raportu dołączony jest kod python02.py, oraz implementacje mnożenia macierzy w cythonie (matmul_cy.pyx) i w C (matmul.c). Zadaniem skryptu python02.py jest wygenerowanie dwóch całkiem dużych macierzy zapełnionych losowymi liczbami całkowitymi oraz pomnożenie ich przez siebie na te cztery różne sposoby. Mnożąc mierzy czas tych podejść oraz upewnia się, że wszystkie wyniki są identyczne.

Wyniki

Przykładowy wynik skryptu python02:

```
Python: 238.3237s
Numpy: 2.8895s | Correct: True
Cython: 1.9583s | Correct: True
External C: 1.9164s | Correct: True
```

Powyższy wynik został osiągnięty przy mnożeniu dwóch macierzy o wymiarach rzędu tysiąc na kilka tysięcy.

Jak widać różnica w czasie wykonania między podejściami jest ogromna. Na mojej maszynie już przy macierzach o wymiarach 500x500 czas wykonania mnożenia przy użyciu NumPy, Cythona i C jest sto razy szybszy od podejścia czysto pythonowskiego. Przy większych rozmiarach, od 1000x1000, widać też różnicę między stosowaniem NumPy a Cythonem/C: mniej więcej 30-40% przyspieszenia. Czasy wykonania Cythona i C są podobne niezależnie od wymiarów macierzy.

Dlaczego tak się dzieje? Takie różnice w czasach wykonania wynikają z kilku rzeczy: Każda liczba w Pythonie to obiekt. Trzymanie macierzy jako listy wierszy i iterowanie po nich sprawia, że każde dodawanie kosztuje dużo (pobranie wskaźnika, rozpakowanie, wyciągnięcie wartości, obliczenia, zapakowanie w nowy obiekt).

W języku C (biblioteka NumPy jest "pod spodem" pisana też w C) operacje dzieją się na niższym poziomie: dane leżą obok siebie w pamięci nieopakowane - to sprawia, że bez rozpakowania można podać je na procesor, a także można korzystać z cache'owania.

Przy wielu obliczeniach NumPy przegrywa z czystym C z kilku powodów: np jest pisany uniwersalnie i bezpiecznie, a moja implementacja pomija testy typów oraz sprawdzanie i komunikowanie błędów, które oczywiście jest wymagane w publicznej bibliotece.

Cython i sprawdzanie błędów

Wyłączenie sprawdzania przez Cythona błędów wyjścia poza tablicę oraz *wraparound'u* czyli użycia indeksu [-1] w tablicach sprawia, że zagnieżdżone liczenie w środku pętli może odbywać się w całości w czystym C, bez interakcji z Pythonem, co zwiększa wydajność wygenerowanego kodu: osiągnięte zostało "*C-speed*" czyli kod generowany przez cythona jest na poziomie wydajnościowym czystego C.

```
Generated by Cython 3.2.3

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython
generated for it.

Raw output: matmul_cy.c

+01: import numpy as np
 02: import cython
 03:
+04: @cython.boundscheck(False)
 05: @cython.wraparound(False)
 06: def mult(int[:, :] A, int[:, :] B):
+07:     cdef int rows_A = A.shape[0]
+08:     cdef int cols_A = A.shape[1]
+09:     cdef int rows_B = B.shape[0]
+10:     cdef int cols_B = B.shape[1]
 11:
+12:     cdef int[:, :] C = np.zeros((rows_A, cols_B), dtype=np.int32)
 13:
 14:     cdef int i, j, k
 15:     cdef int temp_sum
 16:
+17:     for i in range(rows_A):
+18:         for j in range(cols_B):
+19:             temp_sum = 0
+20:             for k in range(cols_A):
+21:                 temp_sum += A[i, k] * B[k, j]
+22:             C[i, j] = temp_sum
 23:
+24:     return np.asarray(C)
```

Zawartość matmul_cy.html wygenerowana przez cython -a. W środku pętli brak żółtych linii.