AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Program nr 3: Rekurencyjna LU faktoryzacja

Mykola Haltiuk, Grzegorz Legęza, Wojciech Ciężobka

Analiza algorytmu

Pseudokod algorytmu rekurencyjnej LU faktoryzacji

```
def lu_decomposition(A: Matrix) -> Matrix, Matrix:

    n = size(A)     # rozmiar macierzy A

    # Warunek brzegowy - macierz L składa się z jedynki,
    # a macierz U składa się z oryginalnej wartości na przekątnej A
    if n == 1:
        return [[1]], A

    # Podział macierzy A na bloki o jednakowym rozmiarze
    [A11 A12]
    [A21 A22] = A

    # Obliczamy LU faktoryzację dla lewej górnej macierzy
    L11, U11 = lu_decomposition(A11)

    # Obliczamy lewą dolną część LU faktoryzacji
    U11_inv = inverse(U11)
    L21 = A21 * U11_inv
    U21 = zeros(m, m)  # górna trójkątna macierz w lewej dolnej części
składa się z zer
```

```
    # Obliczamy prawą górną część LU faktoryzacji
    L11_inv = inverse(L11)
    U12 = L11_inv * A12
    L12 = zeros(m, m)  # dolna trójkątna macierz w prawej górnej
części składa się z zer

    # Obliczamy prawą dolną (najtrudniejszą) część LU faktoryzacji
    S = A22 - (A21 * U11_inv) * (L11_inv * A12)
    L22, U22 = lu_decomposition(S)  # obliczamy rekurencyjnie

    # Składanie obliczonych bloków w wynikowe macierze L i U
    L = [L11, L12]
        [L21, L22]

    U = [U11, U12]
        [U21, U22]

    return L, U
```

Pseudokod rekurencyjnego algorytmu odwracania macierzy

```
def inverse(A: Matrix) -> Matrix:

    n = size(A)     # rozmiar macierzy A

    # Warunek brzegowy - trywialna odwrotność
    if n == 1:
        return 1 / A

    # Podział macierzy A na bloki o jednakowym rozmiarze
    [A11 A12]
    [A21 A22] = A

    # Zmienne pomocnicze
    A11_inv = inverse(A11)
    S22 = A22 - A21 * A11_inv * A12
    S22_inv = inverse(S22)

    # Oblicznie bloków składających się na macierz wynikową B
    B11 = A11_inv + A11_inv * A12 * S22_inv * A21 * A11_inv
    B12 = -A11_inv * A12 * S22_inv
    B21 = -S22_inv * A21 * A11_inv
    B22 = S22_inv

    # Składanie obliczonych bloków w wynikową macierz B
    B = [B11, B12]
        [B21, B22]

    return B
```

Pseudokod algorytmu rekurencyjnego mnożenia macierzy

```python
def multiply_strassen_with_classic(A: Matrix, B: Matrix, size_classic:
int = 8) -> Matrix:

    def strassen(A: Matrix, B: Matrix, n: int):

        # Warunek brzegowy - trywialne mnożenie
        if n == 1:
            return A * B

        # Warunek brzegowy - rozmiar macierzy suboptymalny aby
wywoływać rekurencyjnie
        elif n <= size_classic:
            return multiply_classic(A, B)

        # Podział macierzy A na bloki o jednakowym rozmiarze
        [A11 A12]
        [A21 A22] = A

        # Podział macierzy B na bloki o jednakowym rozmiarze
        [B11 B12]
        [B21 B22] = B

        # Wywołania rekurencyjne aby wyliczy macierze pomocnicze P_i,
i z {1, 2, ..., 7}
        P1 = strassen(A11 + A22, B11 + B22, n // 2)
        P2 = strassen(A21 + A22, B11, n // 2)
        P3 = strassen(A11, B12 - B22, n // 2)
        P4 = strassen(A22, B21 - B11, n // 2)
        P5 = strassen(A11 + A12, B22, n // 2)
        P6 = strassen(A21 - A11, B11 + B12, n // 2)
        P7 = strassen(A12 - A22, B21 + B22, n // 2)

        # Składanie macierzy wynikowej C z bloków powstałych z P_i
        C11 = P1 + P4 - P5 + P7
        C12 = P3 + P5
        C21 = P2 + P4
        C22 = P1 - P2 + P3 + P6

        C = [C11, C12]
            [C21, C22]

        return C

    return strassen(A, B, size(A))
```

FLO: liczba operacji zmiennoprzecinkowych w algorytmie

Niech:
$lu(n)$ - liczba operacji zmiennoprzecinkowych (FLO) dla rekurencyjnej LU faktoryzacji dla macierzy rozmiaru n, $inv(n)$ - liczba operacji zmiennoprzecinkowych (FLO) dla rekurencyjnego odwracania macierzy rozmiaru n,

$str(n)$ - FLO dla rekurencyjnego mnozenia macierzy algorytmem *Strassen'a*,
$sum(n)$ - FLO dla sumowania macierzy rozmiaru n.

$$\begin{cases} inv(2^k) = 2 \cdot inv(2^{k-1}) + 10 \cdot str(2^{k-1}) + 4 \cdot sum(2^{k-1}) + 1, & k \geq 2 \\ inv(2^0) = inv(1) = 1 \\ inv(2^1) = 2 \cdot inv(2^0) + 10 \cdot str(2^0) + 4 \cdot sum(2^0) + 1 = 17 \end{cases}$$

$$\begin{cases} str(2^k) = \frac{23}{5} \cdot 7^k - \frac{18}{5} \cdot 2^k, & k \geq 1 \\ str(2^0) = str(1) = 1 \end{cases}$$

$$sum(2^k) = (2^k)^2 = 4^k$$

dla $k \geq 2$:

$$\begin{aligned} inv(2^k) &= 2 \cdot inv(2^{k-1}) + 10 \cdot str(2^{k-1}) + 4 \cdot sum(2^{k-1}) + 1 = \\ &= 2 \cdot inv(2^{k-1}) + 10\left(\frac{23}{5} \cdot 7^{k-1} - \frac{18}{5} \cdot 2^{k-1}\right) + 4 \cdot 4^{k-1} + 1 = \\ &= 2 \cdot inv(2^{k-1}) + 46 \cdot 7^{k-1} + 4 \cdot 4^{k-1} - 36 \cdot 2^{k-1} + 1 \end{aligned}$$

Niech $r(k) = 46 \cdot 7^k + 4 \cdot 4^k - 36 \cdot 2^k + 1, \quad k \geq 1$

$$\begin{aligned} inv(2^k) &= 2 \cdot inv(2^{k-1}) + r(k-1) = \\ &= 2 \cdot (2 \cdot inv(2^{k-2}) + r(k-2)) + r(k-1) = \\ &= 2 \cdot (2 \cdot (2 \cdot inv(2^{k-3}) + r(k-3)) + r(k-2)) + r(k-1) = \\ &= 2 \cdot (2 \cdot (2 \cdot \ldots inv(2^1) + r(1) \ldots) + r(k-2)) + r(k-1) = \\ &= 2^{k-1} \cdot inv(1) + 2^{k-2} \cdot r(1) + 2^{k-3} \cdot r(2) + \ldots + 2^1 \cdot r(k-2) + 2^0 \cdot r(k-1) = \\ &= 17 \cdot 2^{k-1} + \sum_{i=2}^{k} 2^{k-i} \cdot r(i-1) \end{aligned}$$

Zatem ostatecznie:

$$\begin{cases} inv(2^0) = 1, \\ inv(2^1) = 17 \\ inv(2^k) = 17 \cdot 2^{k-1} + \sum_{i=2}^{k} 2^{k-1} \cdot r(i-1), & k \geq 2 \end{cases}$$

gdzie $r(i) = 46 \cdot 7^k + 4 \cdot 4^k - 36 \cdot 2^k + 1, \quad k \geq 1$

$$lu(2^0) = 1$$

$$lu(2^k) = lu(2^{k-1}) + 2inv(2^{k-1}) + 2str(2^{k-1}) + 3str(2^{k-1}) + sum(2^{k-1}) + lu(2^{k-1})$$

$$= 2lu(2^{k-1}) + 2inv(2^{k-1}) + 5str(2^{k-1}) + sum(2^{k-1})$$

Niech $q(k) = 2inv(2^k) + 5str(2^k) + sum(2^k)$, Wtedy mamy:

$$lu(2^k) = 2lu(2^{k-1}) + q(k-1)$$

$$= 2\Big(2lu(2^{k-2}) + q(k-2)\Big) + q(k-1)$$

$$= 2\Bigg(2\Big(2lu(2^{k-3}) + q(k-3)\Big) + q(k-2)\Bigg) + q(k-1)$$

$$= \dots$$

$$= 2^k + \sum_{i=1}^{k} 2^{k-i} q(i-1)$$

Jeśli rozpiszemy $q(k)$ - to dostaniemy:

$$q(k) = 2 \cdot \left(17 \cdot 2^{k-1} + \sum_{i=2}^{k} 2^{k-i} r(i-1)\right) + 5 \cdot \left(\frac{23}{5} \cdot 7^k - \frac{78}{5} \cdot 2^k\right) + 4^k$$

Eksperymenty

Eksperymenty polegają zbadaniu złożoności algorytmu rekurencyjnego odwracania macierzy poprzez:

- Pomiar złożoności czasowej (średnia artmetyczna z 15 pomiarów)
- Obliczenie liczby wykonanych operacji zmiennoprzecinkowych

Wyniki zebraliśmy dla macierzy rozmiaru $2^k \times 2^k, k \in \{1, 2, \dots, 7\}$. Ograniczenie na rozmiar wynikło z czasu wykonania algorytmu dla dużych macierzy ($k > 7$), ponieważ był on już liczony w minutach i nie było możliwe w sensownym czasie wyliczyć średniej z czasu wykonania. Wartość progowa dla mnozenia z uzyciem algorytmu *Strassena* została dobrana empirycznie i wynosi $l = 3$.

Przygotowanie danych

In [ ]:
```
from lu import lu_decomposition
from utils import *

MAX_K = 8
SEED = 420
REPS = 15
MATRICES_FILE_PATH = f"matrices_k{MAX_K}_s{SEED}.dat"
```

In [ ]:
```
matrices = generate_data(15, MAX_K, SEED, MATRICES_FILE_PATH, True)
```

```
Matrices 2x2 are ready
Matrices 4x4 are ready
Matrices 8x8 are ready
Matrices 16x16 are ready
Matrices 32x32 are ready
Matrices 64x64 are ready
Matrices 128x128 are ready
Matrices 256x256 are ready
```

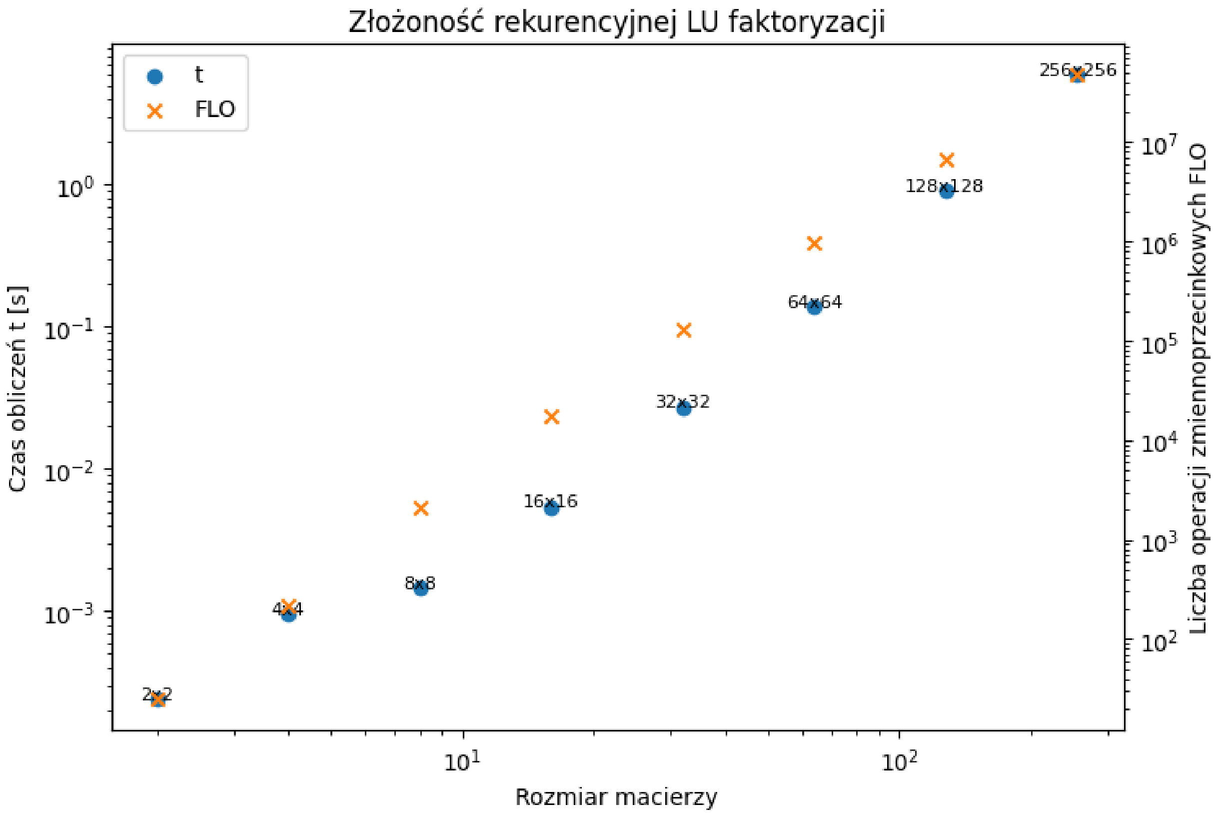
In [ ]:

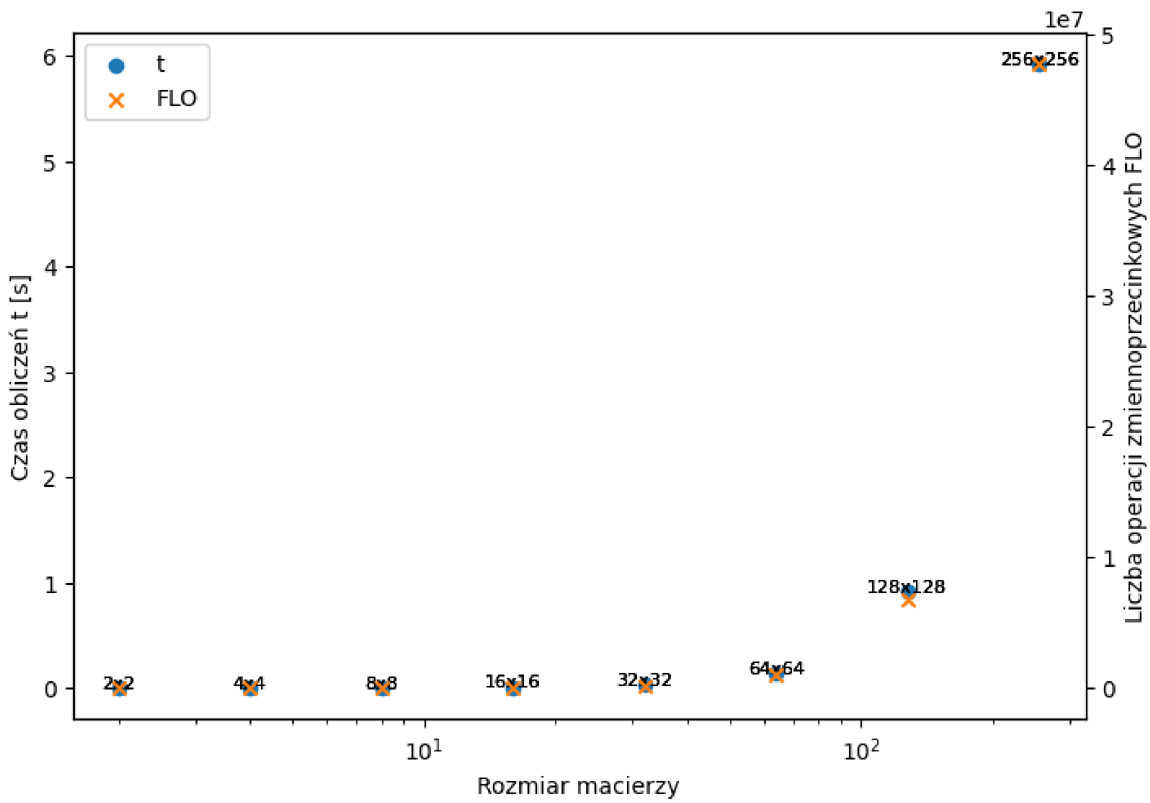
```
sizes = np.array(list(map(lambda ms: ms[0].shape[0], matrices)))
times = np.array(list(map(
    lambda set: list(map(
        lambda m: measure_exec_time(lu_decomposition, m),
        set
    )),
    matrices
)))
flos = np.array(list(map(lambda n: lu_flo(int(np.log2(n))), sizes)))
```

Wykresy

In [ ]:

```
plot_results(sizes, np.mean(times, axis=1), flos)
```





Wykres górny ma zarówno oś rozmiaru, jak i czasu i złożoności w skali logarytmicznej, przez co złożoność wykładnicza prezentuje się w tej reprezentacji liniowo. Proste aproksymujące odpowiednio zlozonosc czasową LU faktoryzacji małych ($n <= 4$) i dużych ($n >= 8$) macierzy, są nachylone pod różnymi kątami. Pokazuje to, ze faktycznie w algorytmie występuje pewna wartośc progowa warunkująca jego działanie i jest to próg $l = 3 \rightarrow n = 8$, który odpowiada za dobór algorytmu mnozenia macierzy (rekurencyjny *Strassena* lub klasyczny).

Doskonale to widać na pierwszym rysunku zaczynając od macierzy $8x8$, czyli naszego progu. Im dalej idziemy (dla większych macierzy) efekty uboczne tracą wpływ i wartość praktyczna zbliża się do teoretycznej.

Z kolei dolny wykres na którym tylko oś rozmiaru jest w skali logarytmicznej, ukazuje wykładniczy charakter złozoności algorytmu. Widac na nim, że złożoności czasowa oraz liczby operacji są silnie skolerowane, co jest zgodne z intuicją.

```
In [ ]:
```