IV zestaw zadań - Mnożenie macierzy hierarchicznych

Kacper Kozubowski, Mateusz Podmokły
III rok Informatyka WI

11 grudnia 2024

1 Treść zadania

Należy wygenerować macierz o rozmiarze $2^{3k} = 2^k \cdot 2^k \cdot 2^k$ dla $k \in \{2, 3, 4\}$ o strukturze opisującej topologię trójwymiarowej siatki zbudowanej z elementów sześciennych. Następnie wykonać polecenia:

1. użyć rekurencyjną procedurę kompresji macierzy z Zadania 3,
2. narysować macierz skompresowaną używając funkcji z Zadania 3,
3. przemnożyć macierz skompresowaną przez wektor,
4. przemnożyć macierz skompresowaną przez samą siebie.

2 Specyfikacja użytego środowiska

Specyfikacja:

- Środowisko: Jupyter Notebook,
- Język programowania: Python,
- System operacyjny: Microsoft Windows 11,
- Architektura systemu: x64.

3 Działanie algorytmów

3.1 Wykorzystane biblioteki

W realizacji rozwiązania wykorzystane zostały następujące biblioteki:

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

3.2 Kod funkcji

W zadaniu wykorzystane zostały funkcje `truncated_SVD`, `compress_matrix`, `create_tree` oraz klasa `Node` z Zadania 3.

```python
def matrix_vector_mult(node, x, tmin, tmax, smin, smax):
    if not node.children:
        if node.rank == 0:
            return np.zeros(tmax - tmin, dtype=float)

        U = np.array(node.U)
        V = np.array(node.V)
        x_sub = x[smin:smax]

        if node.size == (1, 1):
            return node.singular_values * x_sub

        tmp = V @ x_sub
        y = U @ tmp
        return y

    tmid = (tmin + tmax) // 2
    smid = (smin + smax) // 2
    y_tl = matrix_vector_mult(node.children[0], x, tmin, tmid, smin, smid)
    y_tr = matrix_vector_mult(node.children[1], x, tmin, tmid, smid, smax)
    y_bl = matrix_vector_mult(node.children[2], x, tmid, tmax, smin, smid)
    y_br = matrix_vector_mult(node.children[3], x, tmid, tmax, smid, smax)

    top = y_tl + y_tr
    bottom = y_bl + y_br
    return np.concatenate([top, bottom])
```

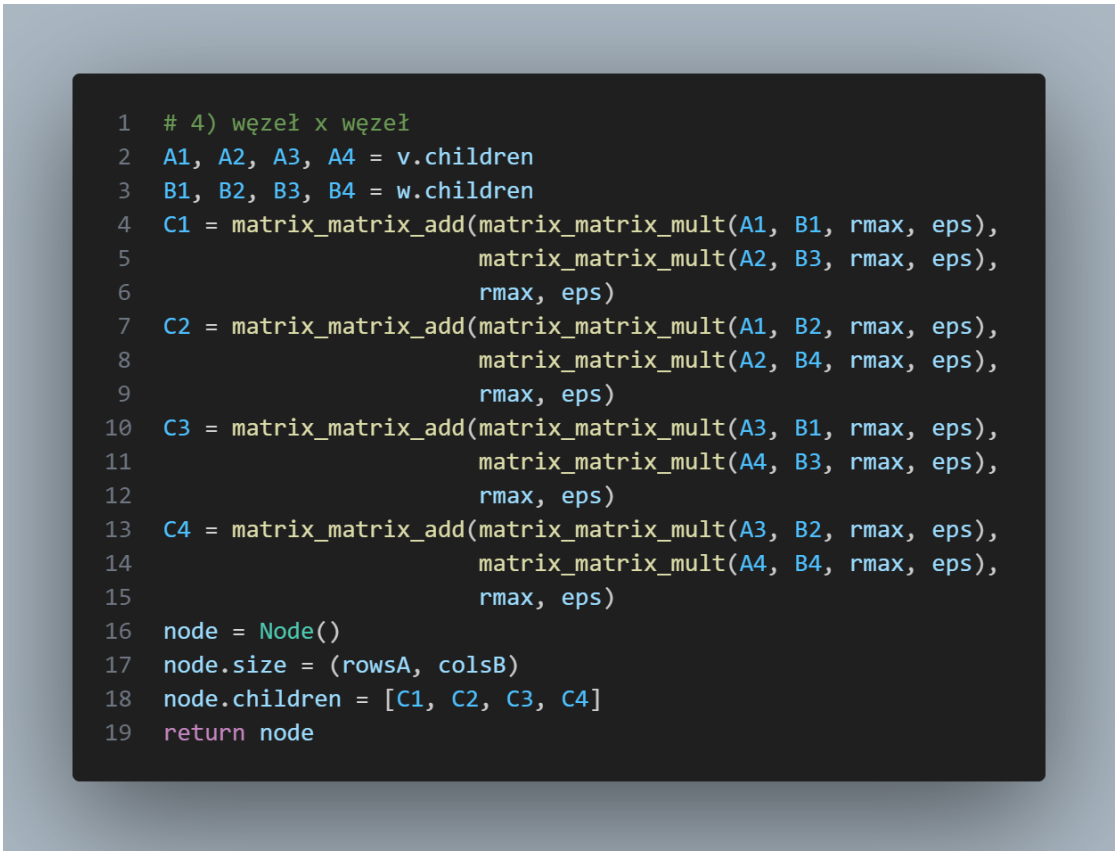
Rysunek 1: Mnożenie macierzy przez wektor.

```python
def matrix_matrix_mult(v: Node, w: Node, rmax=10, eps=1e-8) -> Node:
    if v.rank == 0 and not v.children:
        result = Node()
        result.size = (v.size[0], w.size[1])
        result.rank = 0
        return result
    if w.rank == 0 and not w.children:
        result = Node()
        result.size = (v.size[0], w.size[1])
        result.rank = 0
        return result

    rowsA, colsA = v.size
    rowsB, colsB = w.size
    if colsA != rowsB:
        raise ValueError("Niezgodne rozmiary macierzy do mnożenia!")

    # 1) liść x liść
    if not v.children and not w.children:
        if v.rank == 0 or w.rank == 0:
            node = Node()
            node.size = (rowsA, colsB)
            node.rank = 0
            return node

        Uv = np.array(v.U)
        Vv = np.array(v.V)
        Uw = np.array(w.U)
        Vw = np.array(w.V)
        block = Uv @ (Vv @ Uw) @ Vw

        rank_new, U_r, V_r = truncated_SVD_block(block, rmax, eps)
        node = Node()
        node.size = (rowsA, colsB)
        node.rank = rank_new
        if rank_new > 0:
            node.U = U_r
            node.V = V_r
        return node
```

Rysunek 2: Mnożenie dwóch macierzy część 1.

```
# 2) liść x węzeł
if not v.children and w.children:
    A1 ,A2, A3, A4 = split_leaf_into_4_subleafs(v)
    B1, B2, B3, B4 = w.children
    C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
                           matrix_matrix_mult(A2, B3, rmax, eps),
                           rmax, eps)
    C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
                           matrix_matrix_mult(A2, B4, rmax, eps),
                           rmax, eps)
    C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
                           matrix_matrix_mult(A4, B3, rmax, eps),
                           rmax, eps)
    C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
                           matrix_matrix_mult(A4, B4, rmax, eps),
                           rmax, eps)
    node = Node()
    node.size = (rowsA, colsB)
    node.children = [C1, C2, C3, C4]
    return node

# 3) węzeł x liść
if v.children and not w.children:
    B1, B2, B3, B4 = split_leaf_into_4_subleafs(w)
    A1, A2, A3, A4 = v.children
    C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
                           matrix_matrix_mult(A2, B3, rmax, eps),
                           rmax, eps)
    C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
                           matrix_matrix_mult(A2, B4, rmax, eps),
                           rmax, eps)
    C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
                           matrix_matrix_mult(A4, B3, rmax, eps),
                           rmax, eps)
    C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
                           matrix_matrix_mult(A4, B4, rmax, eps),
                           rmax, eps)
    node = Node()
    node.size = (rowsA, colsB)
    node.children = [C1, C2, C3, C4]
    return node
```

Rysunek 3: Mnożenie dwóch macierzy część 2.



```
# 4) węzeł x węzeł
A1, A2, A3, A4 = v.children
B1, B2, B3, B4 = w.children
C1 = matrix_matrix_add(matrix_matrix_mult(A1, B1, rmax, eps),
                       matrix_matrix_mult(A2, B3, rmax, eps),
                       rmax, eps)
C2 = matrix_matrix_add(matrix_matrix_mult(A1, B2, rmax, eps),
                       matrix_matrix_mult(A2, B4, rmax, eps),
                       rmax, eps)
C3 = matrix_matrix_add(matrix_matrix_mult(A3, B1, rmax, eps),
                       matrix_matrix_mult(A4, B3, rmax, eps),
                       rmax, eps)
C4 = matrix_matrix_add(matrix_matrix_mult(A3, B2, rmax, eps),
                       matrix_matrix_mult(A4, B4, rmax, eps),
                       rmax, eps)
node = Node()
node.size = (rowsA, colsB)
node.children = [C1, C2, C3, C4]
return node
```

Rysunek 4: Mnożenie dwóch macierzy część 3.

Funkcja pomocnicza do mnożenia macierzy:

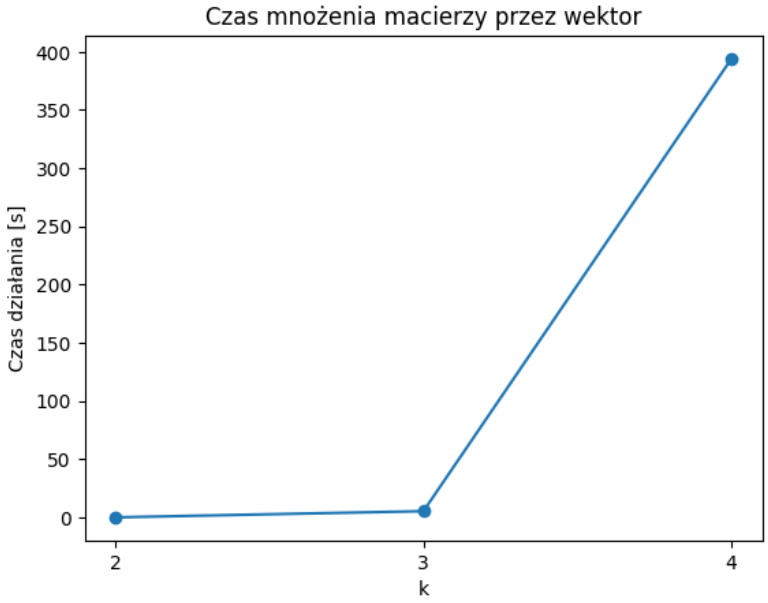
```python
def matrix_matrix_add(v: Node, w: Node, rmax=10, eps=1e-8) -> Node:
    if v.rank==0 and not v.children: return copy_node(w)
    if w.rank==0 and not w.children: return copy_node(v)
    if v.size != w.size:
        raise ValueError("Niezgodne rozmiary macierzy w add!")

    # 1) liść + liść
    if not v.children and not w.children:
        if v.rank == 0 and w.rank == 0:
            node = Node()
            node.size = v.size
            node.rank = 0
            return node

        if v.rank == 0 and w.rank > 0:
            return copy_node(w)
        if w.rank == 0 and v.rank > 0:
            return copy_node(v)
        return add_leaf_leaf(v, w, rmax, eps)

    # 2) węzeł + węzeł
    if v.children and w.children:
        node = Node()
        node.size = v.size
        A1, A2, A3, A4 = v.children
        B1, B2, B3, B4 = w.children

        C1 = matrix_matrix_add(A1, B1, rmax, eps)
        C2 = matrix_matrix_add(A2, B2, rmax, eps)
        C3 = matrix_matrix_add(A3, B3, rmax, eps)
        C4 = matrix_matrix_add(A4, B4, rmax, eps)
        node.children = [C1,C2,C3,C4]
        return node
```

Rysunek 5: Dodawanie część 1.

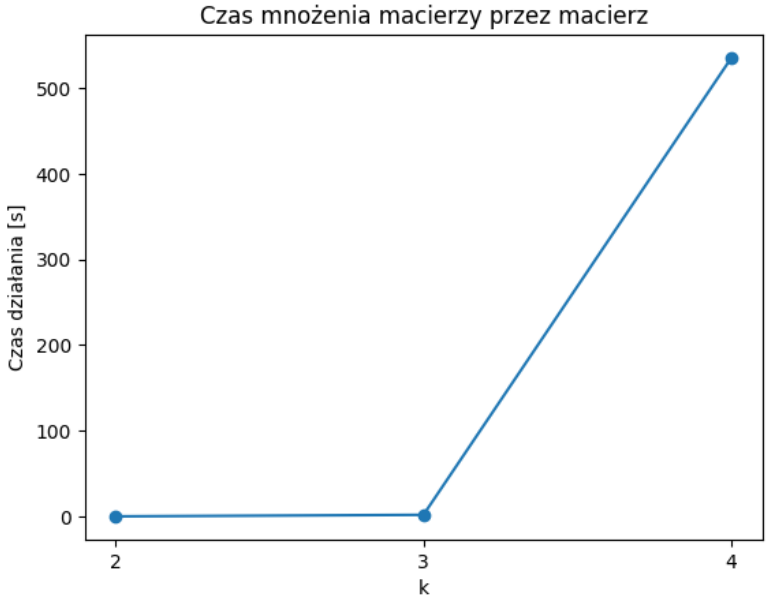
```
# 3) liść + węzeł
if not v.children and w.children:
    sub_v = split_leaf_into_4_subleafs(v)
    node = Node()
    node.size = v.size
    node.children = [matrix_matrix_add(sub_v[0], w.children[0], rmax, eps),
                     matrix_matrix_add(sub_v[1], w.children[1], rmax, eps),
                     matrix_matrix_add(sub_v[2], w.children[2], rmax, eps),
                     matrix_matrix_add(sub_v[3], w.children[3], rmax, eps)]
    return node

# 4) węzeł + liść
if v.children and not w.children:
    sub_w = split_leaf_into_4_subleafs(w)
    node = Node()
    node.size = v.size
    node.children = [matrix_matrix_add(v.children[0], sub_w[0], rmax, eps),
                     matrix_matrix_add(v.children[1], sub_w[1], rmax, eps),
                     matrix_matrix_add(v.children[2], sub_w[2], rmax, eps),
                     matrix_matrix_add(v.children[3], sub_w[3], rmax, eps)]
    return node

raise RuntimeError("Nieobsłużona kombinacja w matrix_matrix_add.")
```

Rysunek 6: Dodawanie część 2.

4 Czas działania



Rysunek 7: Wykres czasu działania.



Rysunek 8: Wykres czasu działania.

5 Oszacowanie złożoności obliczeniowej

Do wyników pomiarów, za pomocą funkcji `curve_fit` z biblioteki `scipy.optimize`, dopasowana została krzywa o równaniu

$$y = \alpha N^{\beta}$$

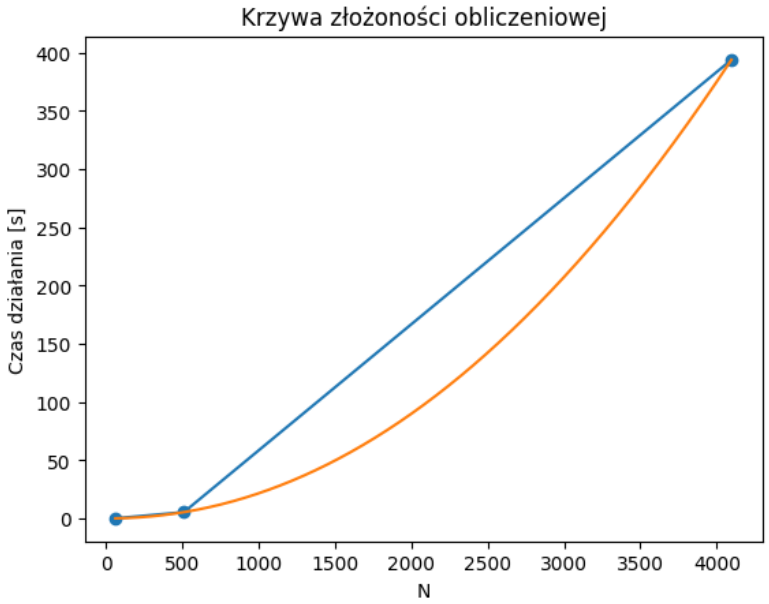
5.1 Mnożenie macierzy i wektora

Dopasowana krzywa:

$$y = 1,41 \cdot 10^{-5} \cdot N^{2,06}$$

Zatem oszacowana złożoność obliczeniowa wynosi:

$$O(n) = n^{2,06}$$



Rysunek 9: Krzywa na wykresie.

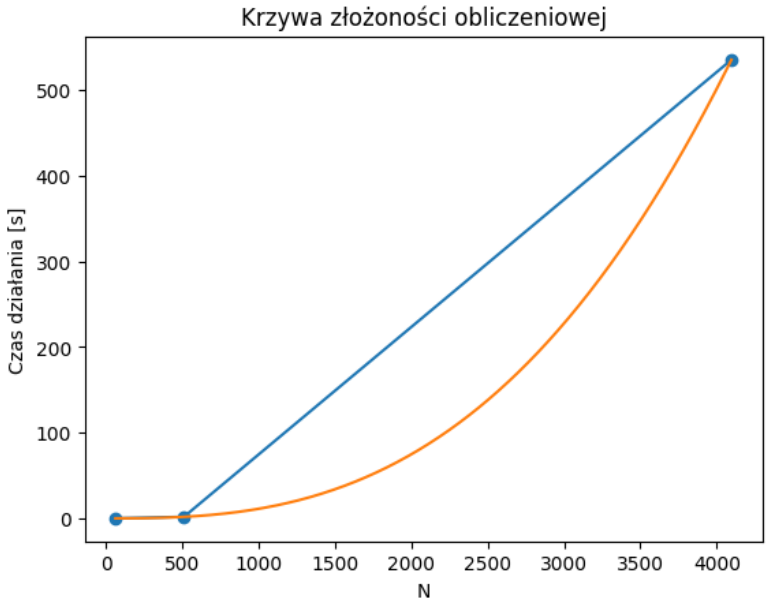
5.2 Mnożenie dwóch macierzy

Dopasowana krzywa:

$$y = 6,4 \cdot 10^{-8} \cdot N^{2,75}$$

Zatem oszacowana złożoność obliczeniowa wynosi:

$$O(n) = n^{2,75}$$



Rysunek 10: Krzywa na wykresie.

6