Rachunek macierzowy

Sprawozdanie z projektu nr.1

Paweł Surdyka, Hanc Bartosz

Zadanie

1. Dla macierzy o rozmiarze mniejszym lub równym $2^l \times 2^l$ algorytm tradycyjny. Dla macierzy o rozmiarze większym od $2^l \times 2^l$ algorytm rekurencyjny Binéta.

Pseudokod – algorytm tradycyjny

```
Funkcja TraditionalMatrixMultiplication(A, B, n, Counter):
    Tworzymy macierz wynikową C o wymiarach n × n, wypełnioną zerami
    Dla i od 0 do n - 1:
        Dla j od 0 do n - 1:
            sum_val ← 0
            Dla k od 0 do n - 1:
                sum_val ← sum_val + (A[i][k] * B[k][j])   // Mnożenie i dodanie
            C[i][j] ← sum_val
    Zwróć C
```

Algorytm Bineta

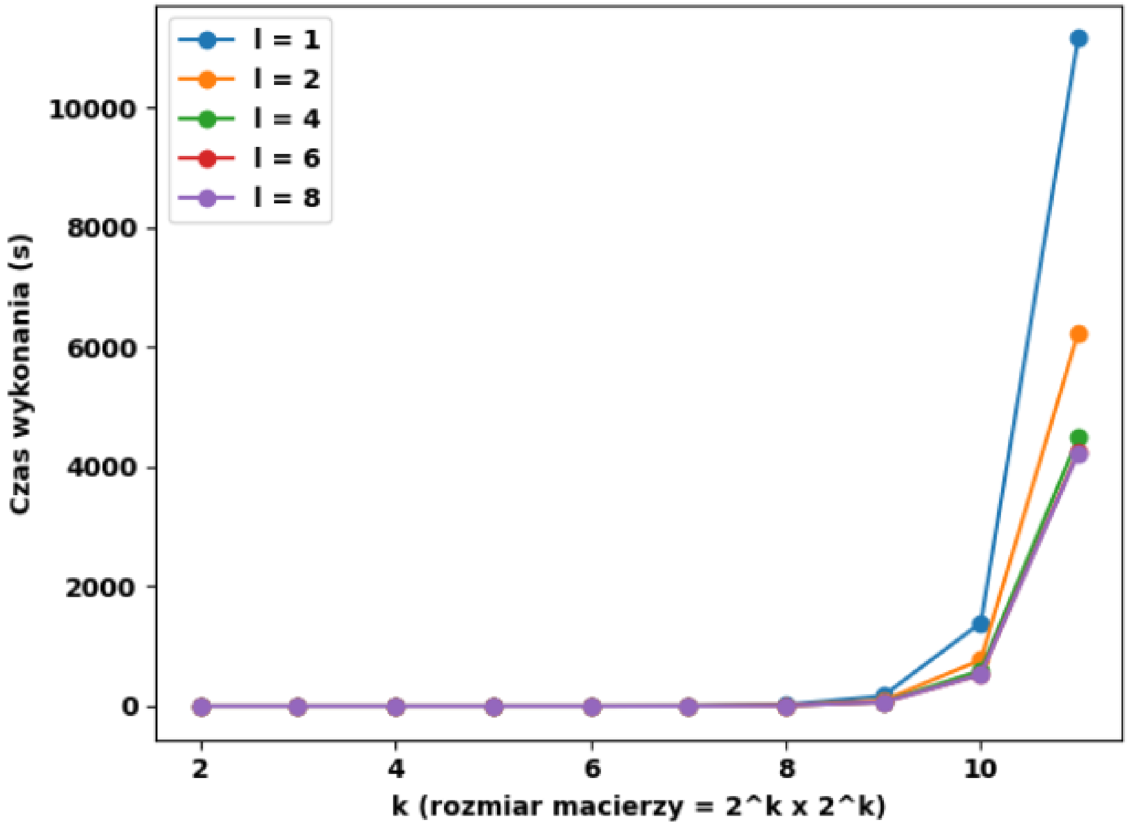
$$\text{Klasyczny (Binét):} \begin{bmatrix} (A_{11}B_{11} + A_{12}B_{21}) & (A_{11}B_{12} + A_{12}B_{22}) \\ (A_{21}B_{11} + A_{22}B_{21}) & (A_{21}B_{12} + A_{22}B_{22}) \end{bmatrix}$$

Każde mnożenie np. $A_{11} * B_{11}$ to rekurencyjne mnożenie bloków.
Koszt: 8 mnożeń $(\mathcal{O}((n/2)^3))$, 4 dodawania $(\mathcal{O}((n/2)^2))$

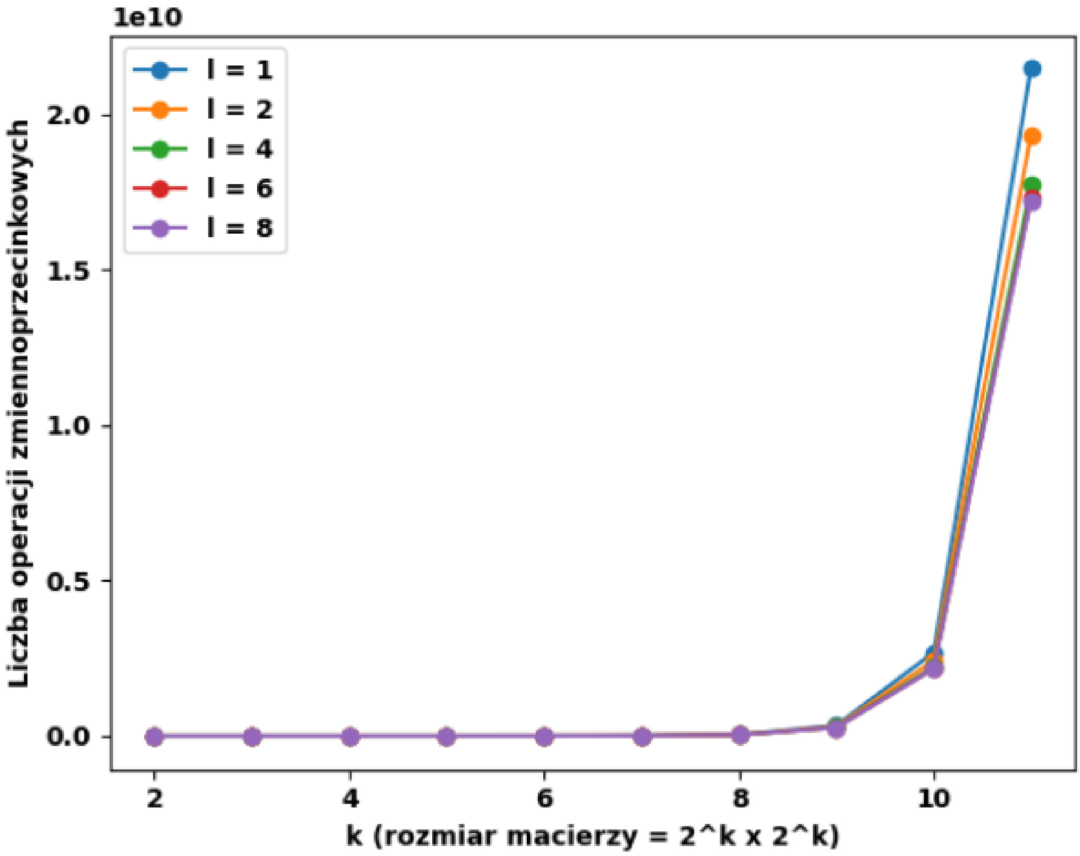
Pseudokod – algorytm Bineta

```
Funkcja RecursiveMatrixMultiplication(A, B, l, Counter):
    n ← długość A
    Jeżeli n = 1:
        Zwróć [[A[0][0] * B[0][0]]]

    Jeżeli n ≤ l:  // Gdy macierz jest mała, używamy tradycyjnego mnożenia
        Zwróć TraditionalMatrixMultiplication(A, B, Counter)

    W przecwinym wypadku rekurencyjnie liczymy podmacierze
        (A11, A12, A21, A22) ← PodzielMacierz(A)
        (B11, B12, B21, B22) ← PodzielMacierz(B)

        M1 ← RecursiveMatrixMultiplication(A11, B11, l, Counter)
        M2 ← RecursiveMatrixMultiplication(A12, B21, l, Counter)
        M3 ← RecursiveMatrixMultiplication(A11, B12, l, Counter)
        M4 ← RecursiveMatrixMultiplication(A12, B22, l, Counter)
        M5 ← RecursiveMatrixMultiplication(A21, B11, l, Counter)
        M6 ← RecursiveMatrixMultiplication(A22, B21, l, Counter)
        M7 ← RecursiveMatrixMultiplication(A21, B12, l, Counter)
        M8 ← RecursiveMatrixMultiplication(A22, B22, l, Counter)

        C11 ← AddMatrix(M1, M2, Counter)
        C12 ← AddMatrix(M3, M4, Counter)
        C21 ← AddMatrix(M5, M6, Counter)
        C22 ← AddMatrix(M7, M8, Counter)

    Zwróć PolaczMacierze(C11, C12, C21, C22)
```

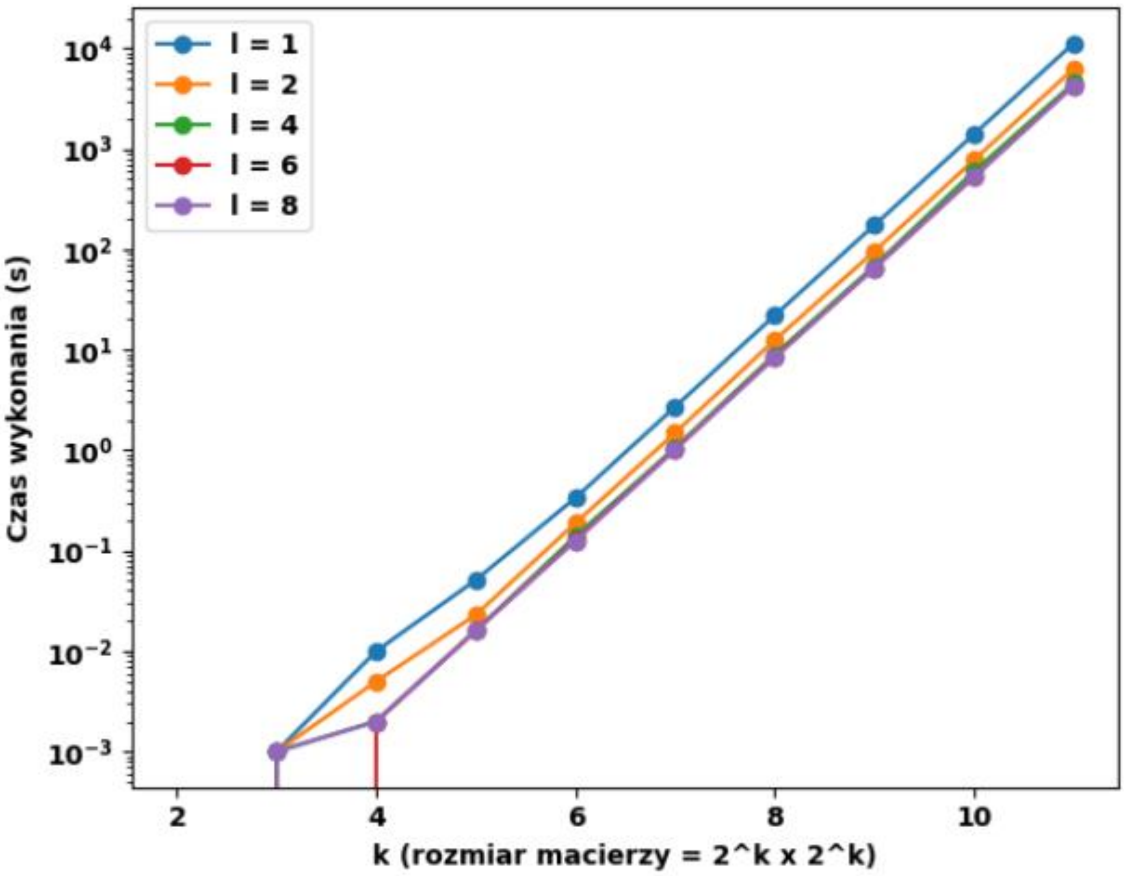
Wykresy



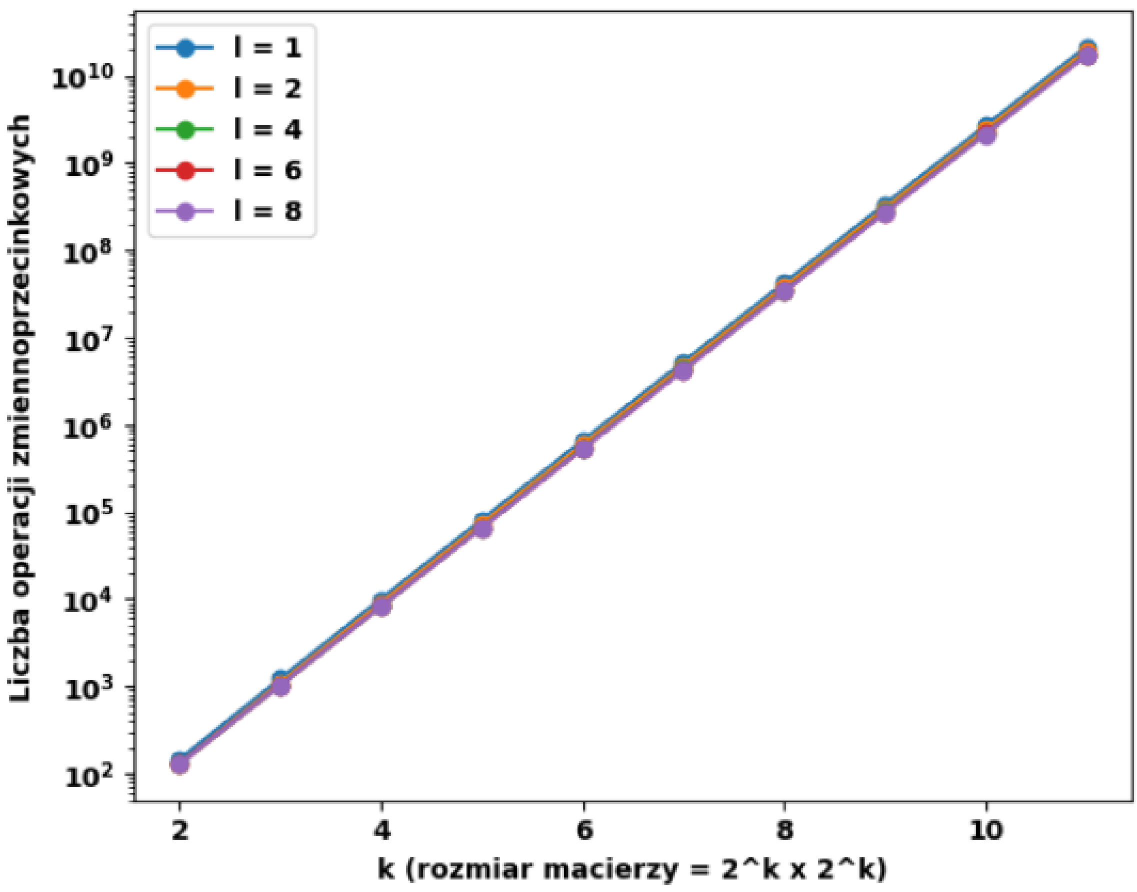
Wykres 1: Czas działania algorytmu Bineta dla różnych l



Wykres 2: Liczba operacji zmiennoprzecinkowych dla różnych l



Wykres 3: Czas działania algorytmu Bineta dla różnych l (skala logarytmiczna)



Wykres 4: Liczba operacji zmiennoprzecinkowych dla różnych l (skala logarytmiczna)

Wnioski

Wyższe l zmniejsza czas działa i ilość operacji zmiennoprzecinkowych, ponieważ wcześniej przechodzimy na mnożenie tradycyjne, które dla małych macierzy jest bardziej efektywne niż rekurencyjna dekompozycja.

Dla małych wartości l algorytm wykonuje więcej poziomów rekurencji, co zwiększa liczbę wywołań i operacji dodawania, powodując wydłużenie czasu działania.

Zależność pomiędzy rozmiarem macierzy k a czasem wykonania jest wykładnicza.