Spis treści

1. Opis projektu – Obliczanie wartości własnych korzystając z faktoryzacji QR

Żeby obliczyć (rzeczywiste) wartości własne macierzy *A* wygodnie użyć jej faktoryzacji QR, tzn. reprezentacji:

$$A = QR$$

gdzie Q jest macierzą ortogonalną (tzn. taką że $Q^T\mathrm{Q} = \mathrm{I}$), zaś R jest macierzą górną trójkątną. Algorytm jest następujący:

$$A^{(1)} = A$$

$$A^{(k)} = Q_k R_k$$

$$A^{(k+1)} = R_k Q_k$$

Można pokazać, że w przypadku gdy macierz A ma na wartości własnych o różnych modułach:

$$a_{ii}^k \xrightarrow[k\to\infty]{} \lambda_i$$

$$a_{ij}^k \xrightarrow[k\to\infty]{} 0, i \neq j$$

Faktoryzacja QR macierzy A o postaci $A = QR$, gdzie Q jest macierzą ortogonalną (tzn. taką, że $Q^T Q = I$), zaś macierz R jest macierzą górną trójkątną, może być wyznaczona przy użyciu algorytmu ortogonalizacji Grama-Schmidta.

2. Opis algorytmu

2.1 Faktoryzacja QR

```
void qr_factorization_seq(int size, double **matrix, double **q, double **r)
{
    for (int k = 0; k < size; k++)
    {
        double r_sum = 0;
        for (int i = 0; i < size; i++)
        {
            r_sum += matrix[i][k] * matrix[i][k];
        }

        r_sum = sqrt(r_sum);
        r[k][k] = r_sum;

        for (int i = 0; i < size; i++)
        {
            q[i][k] = matrix[i][k] / r_sum;
        }

        for (int j = k + 1; j < size; j++)
        {
            r_sum = 0;
            for (int i = 0; i < size; i++)
            {
                r_sum += q[i][k] * matrix[i][j];
            }

            r[k][j] = r_sum;

            for (int i = 0; i < size; i++)
            {
                matrix[i][j] = matrix[i][j] - r[k][j] * q[i][k];
            }
        }
    }
}
```

2.2 Wyznaczanie wartości własnych przy pomocy faktoryzacji QR

```c
double *qr_algorithm(int size, double **matrix, int parallel, int number_of_iterations)
{
    double **result = create_matrix(size);
    copy_matrix(size, matrix, result);
    double **q = create_matrix(size);
    double **r = create_matrix(size);

    double *eigenvalues = malloc(sizeof(double) * size);

    for (int i = 0; i < number_of_iterations; i++)
    {
        qr_factorization_seq(size, result, q, r);
        /* Ak+1 = Rk Qk */
        mul_matrix(size, r, q, result);
    }

    for (int i = 0; i < size; i++)
        eigenvalues[i] = result[i][i];

    free_matrix(size, result);
    free_matrix(size, q);
    free_matrix(size, r);

    return eigenvalues;
}
```

- macierz zbiega do macierzy trójkątnej górnej, a wartości własne znajdują się na przekątnej.

3. Podejście równoległe

Wszystkie operacje kolumnowe w faktoryzacji QR są niezależne, więc możemy wykonywać je równolegle.

3.1 Równoległa Faktoryzacja QR z wykorzystaniem OpenMP

```
void qr_factorization_parallel(int size, double **matrix, double **q, double **r)
{

    double r_sum;

    int i;
    int j;
    int k;

    for (k = 0; k < size; k++)
    {
        r_sum = 0;
        #pragma omp parallel for private(i) shared(matrix, size) reduction(+:r_sum)
        for (i = 0; i < size; i++)
        {
            r_sum += matrix[i][k] * matrix[i][k];
        }

        r_sum = sqrt(r_sum);
        r[k][k] = r_sum;

        #pragma omp parallel for private(i) shared(k, r, matrix, q, size)
        for (i = 0; i < size; i++)
        {
            q[i][k] = matrix[i][k] / r[k][k];
        }

        #pragma omp parallel for private(j, r_sum, i) shared(k, r, q, matrix, size)
        for (j = k + 1; j < size; j++)
        {
            r_sum = 0;
            for (i = 0; i < size; i++)
            {
                r_sum += q[i][k] * matrix[i][j];
            }

            r[k][j] = r_sum;

            for (i = 0; i < size; i++)
            {
                matrix[i][j] = matrix[i][j] - r[k][j] * q[i][k];
            }
        }
    }
}
```

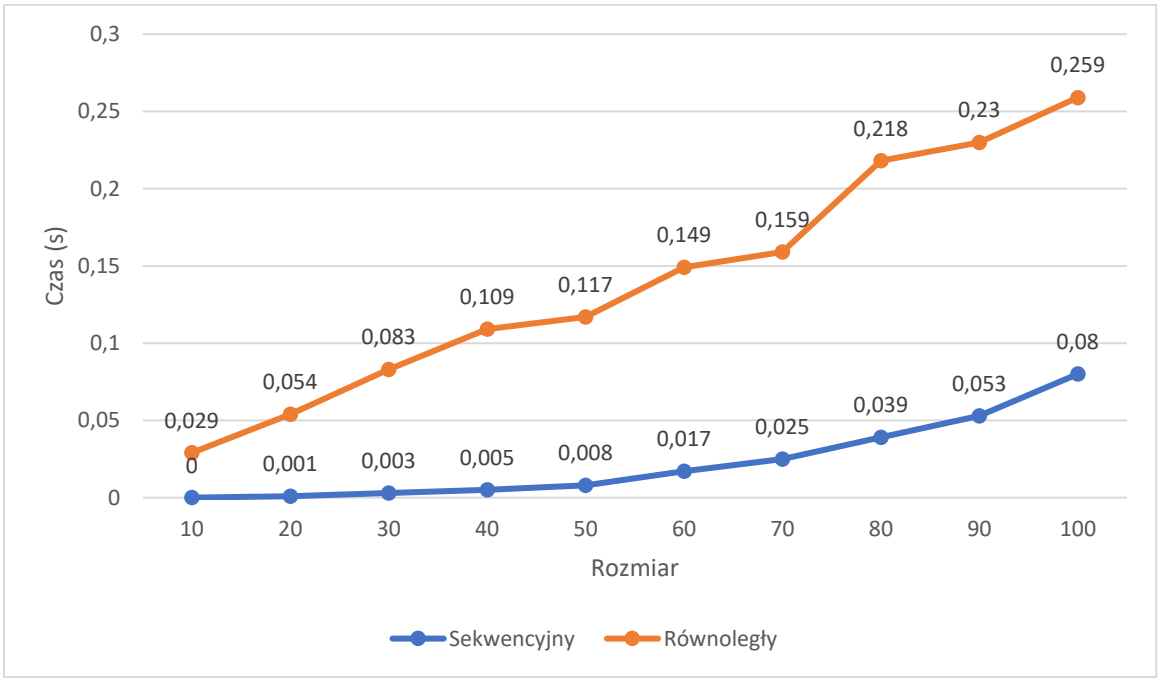
3.2 Równoległe wyznaczanie wartości własnych z wykorzystanie OpenMP

```c
double *qr_algorithm(int size, double **matrix, int parallel, int number_of_iterations)
{
    double **result = create_matrix(size);
    copy_matrix(size, matrix, result);
    double **q = create_matrix(size);
    double **r = create_matrix(size);

    double *eigenvalues = malloc(sizeof(double) * size);

    for (int i = 0; i < number_of_iterations; i++)
    {
        qr_factorization_parallel(size, result, q, r);
        /* Ak+1 = Rk Qk */
        mul_matrix_parallel(size, r, q, result);
    }

    for (int i = 0; i < size; i++)
        eigenvalues[i] = result[i][i];

    free_matrix(size, result);
    free_matrix(size, q);
    free_matrix(size, r);

    return eigenvalues;
}
```

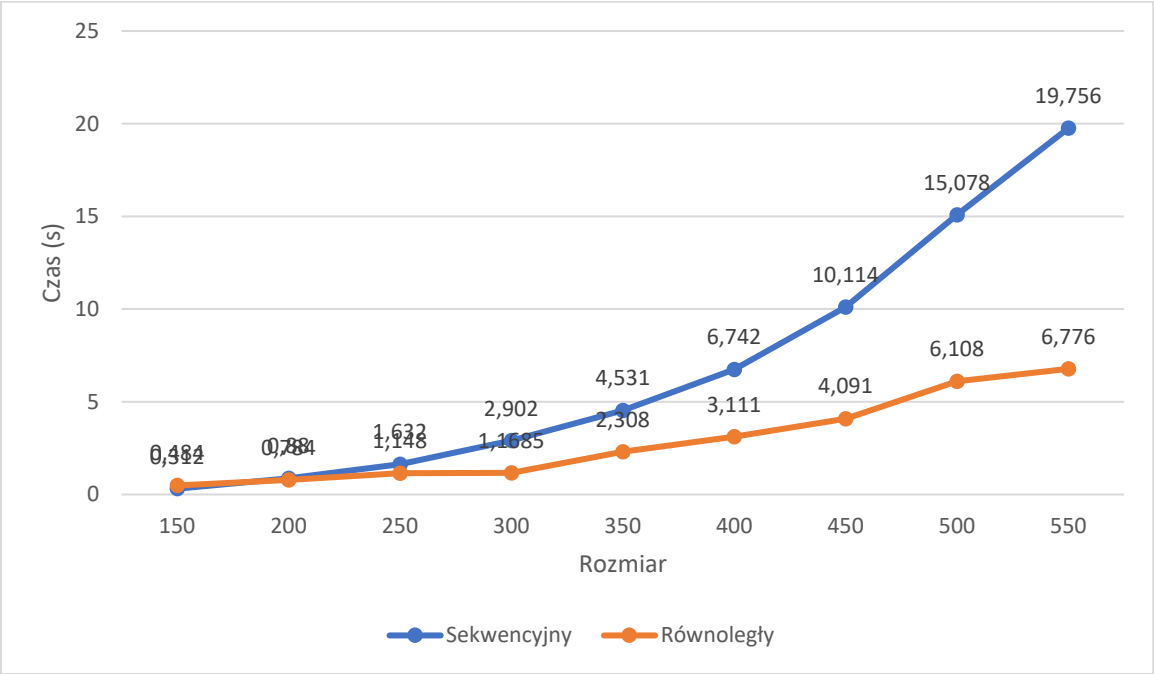
4. Wydajność algorytmu

Testy były wykonywane na komputerze z 4 rdzeniami procesora dla 10 iteracji.

4.1 Małe macierze



4.2 Duże macierze



5. Wnioski

Algorytm równoległy dla małych macierzy jest wolniejszy dla małych macierzy z powodu synchronizacji wątków. Zyski z algorytmu równoległego są zauważalne przy rozmiarze macierzy ok. 200 na 200.