Systemy/Programowanie równoległe i rozproszone

Układ równań liniowych-redukcja QR - transformacje Householdera

Autorzy
Michał Domin
Tomasz Szkaradek

1. Wstęp

Projekt dotyczy rozwiązywania układów równań liniowych przy użyciu redukcji ortogonalnej QR oraz transformacji Householdera

1.1. Wymagania

Implementacja projektu została wykonana w języku C++ w standardzie C++17 z wykorzystaniem biblioteki MPI 4.0.1, co umożliwia równoległe przetwarzanie danych.

1.2. Uruchomienie

Aby uruchomić projekt, należy przeprowadzić kilka kroków. Na początku należy sklonować repozytorium projektu przy użyciu komendy "git clone https://github.com/mdmkl6/SRIR_QR_decomposition_Project" Następnie przechodzimy do katalogu z projektem, korzystając z polecenia "cd SRIR_QR_decomposition_Project".
Plik makefile zawiera reguły:

- -compile: która pozwala skompilować projekt.
- -run: która pozwala uruchomić program.
- -clean: usuwająca pliki wynikowe projektu
- -all: uruchamiająca wszystkie powyższe po kolei

Można też podać po wpisaniu wartość "file={plik}" gdzie w miejsce {plik} wpisujemy ścieżkę do pliku z macierzą A, podstawową wartością pliku jest pik "matrix.txt" oraz wartość "outfile={plik}" gdzie jeśli zostanie podana do pliku o ścieżce podanej w miejsce {plik} zostaną zapisane wyniki.

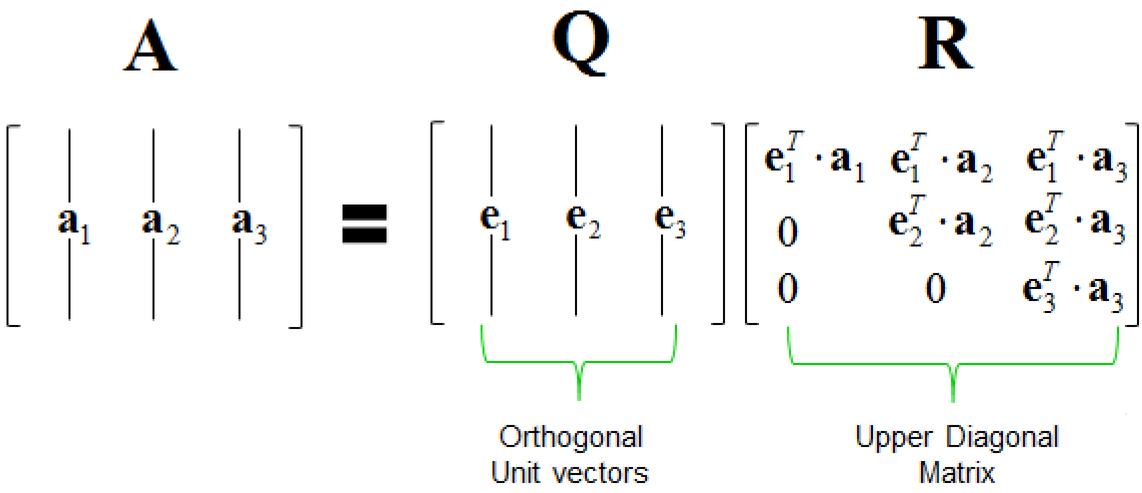
W pliku Makefile, oprócz kompilacji i uruchamiania aplikacji, można również uruchamiać testy poprzez wywołanie komendy "make compile test" lub "make run tests". Jednym z przykładów testów, które można uruchomić, jest test dokonujący wczytania zadanej macierzy z pliku, przeprowadzający dekompozycję na macierze Q i R, a następnie porównujący wczytaną macierz A z iloczynem macierzy Q i R. Jest to ważny proces testowania, który pozwala na sprawdzenie poprawności działania programu oraz wykrycie błędów.

1.3. Teoria i Algorytm

Dla danej macierzy A o wymiarach m x n, gdzie m > n, rozkład QR ma postać:

$$A = Q\begin{bmatrix} R \\ O \end{bmatrix}$$

macierz Q ma wymiar m x m, jest ortogonalna, a macierz R ma wymiar n x n i jest górnotrójkątna.



- Może być wykorzystany do rozwiązywania układów równań liniowych, problemów najmniejszych kwadratów itp.
- Podobnie jak w przypadku eliminacji Gaussa, zera są wprowadzane sukcesywnie do macierzy A, aż osiągnie ona postać górnotrójkątną. Jednak w rozkładzie QR stosuje się transformacje ortogonalne zamiast eliminacji elementarnych.
- Metody rozkładu QR:
  - transformacje Householdera (reflektory elementarne)
  - transformacje Givensa (obroty płaszczyznowe)
  - ortogonalizacja Grama-Schmidta
- Transformacja Householdera ma postać:

  $$\boldsymbol{H} = \boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}$$

  gdzie v to niezerowy wektor.
- Z definicji wynika, że $H = H^T = H^{-1}$, zatem H jest jednocześnie ortogonalna i symetryczna.
- Dla danego wektora a, wybiera się wektor v w taki sposób, aby:

$$\boldsymbol{H}\boldsymbol{a} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \boldsymbol{e}_1$$

- Podstawiając to do wzoru na H, widzimy, że możemy przyjąć:

$$\boldsymbol{v} = \boldsymbol{a} - \alpha \boldsymbol{e}_1$$

a aby zachować normę, musimy przyjąć $\alpha = \pm||a||_2$, z odpowiednim znakiem, aby uniknąć anulowania się wartości.

for $k = 1$ **to** n
$\quad \alpha_k = -\mathrm{sign}(a_{kk})\sqrt{a_{kk}^2 + \cdots + a_{mk}^2}$
$\quad \boldsymbol{v}_k = \begin{bmatrix} 0 & \cdots & 0 & a_{kk} & \cdots & a_{mk} \end{bmatrix}^T - \alpha_k \boldsymbol{e}_k$
$\quad \beta_k = \boldsymbol{v}_k^T \boldsymbol{v}_k$
$\quad$ **if** $\beta_k = 0$ **then**
$\quad\quad$ continue with next k
$\quad$ **for** $j = k$ **to** n
$\quad\quad \gamma_j = \boldsymbol{v}_k^T \boldsymbol{a}_j$
$\quad\quad \boldsymbol{a}_j = \boldsymbol{a}_j - (2\gamma_j/\beta_k)\boldsymbol{v}_k$
$\quad$ **end**
end

Rozkład QR metodą Householdera jest podobny do eliminacji Gaussa w rozkładzie LU. Tworzenie wektora Householdera v_k jest analogiczne do obliczania mnożników w eliminacji Gaussa. Kolejne aktualizacje pozostałej nieredukowanej części macierzy są również analogiczne do eliminacji Gaussa. Dlatego implementacja równoległa jest podobna do równoległej implementacji rozkładu LU, ale, z tym że wektory Householdera są transmitowane poziomo zamiast mnożników.

2. Działanie programu

Działanie programu rozpoczyna się od inicjalizacji zmiennych oraz MPI, co umożliwia korzystanie z mechanizmów równoległych. Następnie na weźle master programu przeprowadzane jest wczytanie macierzy z pliku.

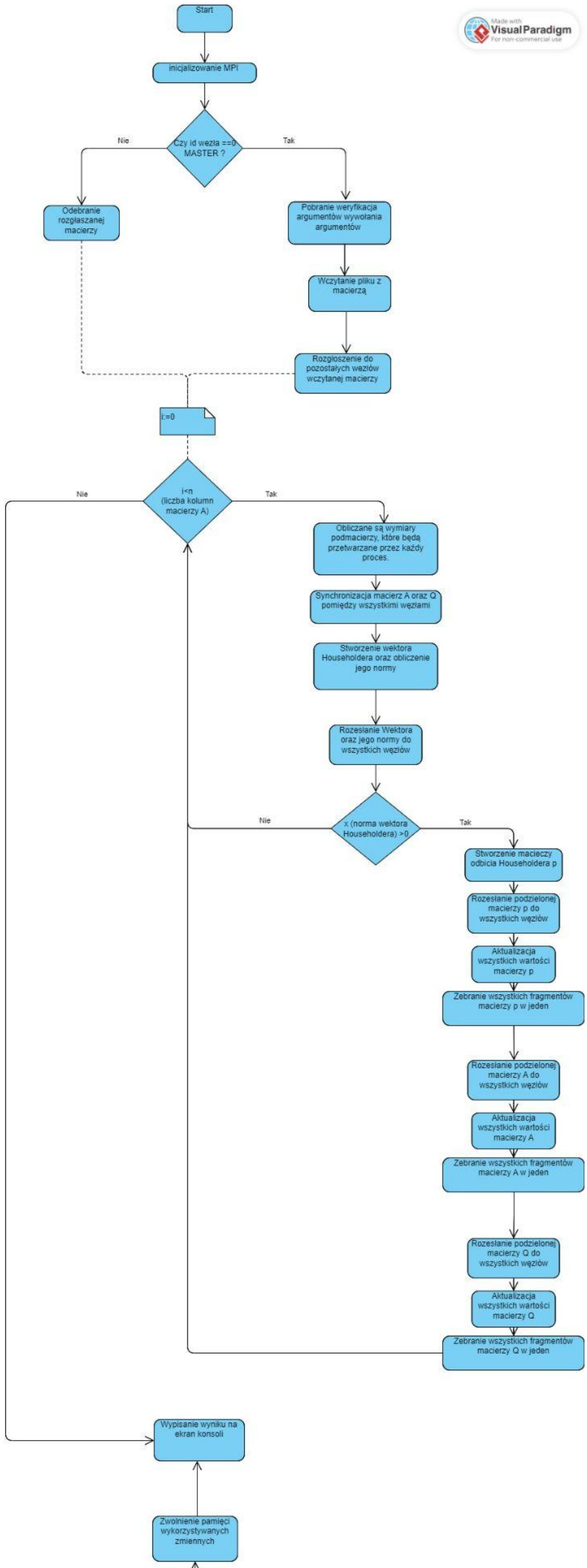
Dla każdej kolumny macierzy A, program przeprowadza szereg operacji. Na początku obliczane są wymiary podmacierzy, a następnie ich wartości są rozgłaszane do pozostałych wezłów. Następnie obliczany jest wektor Householdera oraz jego norma, a wartość tej normy jest rozgłaszana do pozostałych wezłów. Jeśli wartość normy jest większa od zera, to macierz p jest rozpraszana pomiędzy procesy metodą scatter.

```
// Metoda do rozsyłania danych macierzy pomiędzy procesami.
// sendcounts - liczba elementów, która ma zostać wysłana do każdego z procesów
// displs - przesunięcie początku danych w buforze sendbuf dla każdego procesu
// target - wskaźnik do macierzy, która otrzyma rozsyłane dane
// recvcount - liczba elementów, które zostaną odebrane przez proces
void scatter(int *sendcounts, int *displs, matrix *target, int recvcount)
{
    MPI_Scatterv(&_data[0][0], sendcounts, displs, MPI_DOUBLE, target->_data[0], recvcount, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

Każdy z procesów oblicza część macierzy mat, która zostanie użyta do skonstruowania macierzy p, a następnie funkcja gather zbiera porcje macierzy mat z powrotem do macierzy p.

```
// Metoda do zbierania danych macierzy od procesów.
// sendcounts - liczba elementów, która ma zostać wysłana przez każdy z procesów
// target - wskaźnik do macierzy, która otrzyma zebrane dane
// recvcount - liczba elementów, które zostaną odebrane przez proces
// displs - przesunięcie początku danych w buforze recvbuf dla każdego procesu
void gather(int sendcounts, matrix *target, int *recvcount, int *displs)
{
    MPI_Gatherv(&_data[0][0], sendcounts, MPI_DOUBLE, target->_data[0], recvcount, displs, MPI_DOUBLE, 0, MPI_COMM_WORLD);
}
```

Wartości macierzy A oraz Q są aktualizowane w trakcie wykonywania programu.
Na końcu działania programu, w głównym weźle, wyniki są wyświetlane, pamięć jest zwalniana, a funkcja MPI_Finalize() finalizuje środowisko MPI.


Start
inicjalizowanie MPI
Czy id wezła ==0 MASTER ?
Nie
Tak
Odebranie rozgłaszanej macierzy
Pobranie weryfikacja argumentów wywołania argumentów
Wczytanie pliku z macierzą
Rozgłoszenie do pozostałych wezłów wczytanej macierzy
i:=0
i<n (liczba kolumn macierzy A)
Nie
Tak
Obliczane są wymiary podmacierzy, które będą przetwarzane przez każdy proces.
Synchronizacja macierz A oraz Q pomiędzy wszystkimi węzłami
Stworzenie wektora Householdera oraz obliczenie jego normy
Rozesłanie Wektora oraz jego normy do wszystkich węzłów
x (norma wektora Householdera) >0
Nie
Tak
Stworzenie macieczy odbicia Householdera p
Rozesłanie podzielonej macierzy p do wszystkich węzłów
Aktualizacja wszystkich wartości macierzy p
Zebranie wszystkich fragmentów macierzy p w jeden
Rozesłanie podzielonej macierzy A do wszystkich węzłów
Aktualizacja wszystkich wartości macierzy A
Zebranie wszystkich fragmentów macierzy A w jeden
Rozesłanie podzielonej macierzy Q do wszystkich węzłów
Aktualizacja wszystkich wartości macierzy Q
Zebranie wszystkich fragmentów macierzy Q w jeden
Wypisanie wyniku na ekran konsoli
Zwolnienie pamięci wykorzystywanych zmiennych

3. Podsumowanie wyniki oraz wnioski

Program przyjmuje macierz na wejściu i zwraca dwie macierze QR, korzystając z metody Householdera oraz programowania zrównoleglonego. Oto kilka przykładowych wywołań programu:

```
9szkaradek@stud204-13:~/Desktop/Systemy równoległe i rozproszone/SRIR_QR_decomposition_Project$ make all
/opt/nfs/config/station204_name_list.sh 1 16 >nodes && \
/opt/nfs/mpich-4.0.1/bin/mpicxx QR.cpp -o QR.out
/opt/nfs/mpich-4.0.1/bin/mpiexec -f nodes -n 5 ./QR.out matrix.txt
Loaded Matrix A:
  1.00000   2.00000   3.00000
  2.00000   3.00000   4.00000
  3.00000   4.00000   5.00000

Solution
Matrix Q:
  0.26726   0.87287   0.40825
  0.53452   0.21822  -0.81650
  0.80178  -0.43644   0.40825

Matrix R:
  3.74166   5.34522   6.94879
  0.00000   0.65465   1.30931
  0.00000   0.00000  -0.00000
```

```
9szkaradek@stud204-13:~/Desktop/Systemy równoległe i rozproszone/SRIR_QR_decomposition_Project$ make all
/opt/nfs/config/station204_name_list.sh 1 16 >nodes && \
/opt/nfs/mpich-4.0.1/bin/mpicxx QR.cpp -o QR.out
/opt/nfs/mpich-4.0.1/bin/mpiexec -f nodes -n 5 ./QR.out matrix.txt
Loaded Matrix A:
  2.00000   1.00000   3.00000   7.00000
  7.00000   3.00000   1.00000   2.00000
  4.00000   2.00000   0.00000   6.00000
  4.00000   2.00000   0.00000   9.00000

Solution
Matrix Q:
  0.21693  -0.25308  -0.94281  -0.00000
  0.75926   0.65079   0.00000   0.00000
  0.43386  -0.50617   0.23570  -0.70711
  0.43386  -0.50617   0.23570   0.70711

Matrix R:
  9.21954   4.23014   1.41005   9.54494
 -0.00000  -0.32540  -0.10846  -8.06258
 -0.00000   0.00000  -2.82843  -3.06413
 -0.00000   0.00000   0.00000   2.12132
```

- Metoda Householdera jest skutecznym sposobem na dekompozycję macierzy na dwie macierze ortogonalne i trójkątną górną. Dzięki temu można łatwo rozwiązywać układy równań liniowych oraz wyznaczać wartości własne macierzy.
- Wykorzystanie programowania zrównoleglonego pozwala na przyspieszenie procesu dekompozycji macierzy, co jest szczególnie korzystne w przypadku dużych macierzy.
- Programowanie zrównoleglone wymaga jednak odpowiedniego dostosowania algorytmu do architektury procesora oraz uwzględnienia kosztów synchronizacji wątków, co może być trudne i czasochłonne.
- Warto zwrócić uwagę na optymalizację pamięciową programu, aby uniknąć przeciążenia pamięci w przypadku dużej liczby danych.
- W zastosowaniach praktycznych metoda Householdera oraz programowanie zrównoleglone są stosowane w wielu dziedzinach, takich jak przetwarzanie sygnałów, obliczenia numeryczne, czy uczenie maszynowe.
- Dostosowanie programu do specyficznych potrzeb użytkownika, takich jak zastosowanie innej metody dekompozycji macierzy, może wymagać dokładniejszej analizy algorytmu oraz jego implementacji.

4. Źródła

- **https://dsc-spidal.github.io/harp/docs/harpdaal/qr/ - Dokumentacja**
- **https://atozmath.com/example/MatrixEv.aspx?q=qrdecomphh&q1=E1 - Algorytm oraz przykłady**
- **https://github.com/mdmkl6/SRIR_QR_decomposition_Project - Repozytorium projektu**