Systemy/Programowanie równoległe i rozproszone

Układ równań liniowych-redukcja QR - transformacje Householdera

Autorzy
Michał Domin
Tomasz Szkaradek

1. Wstęp

Projekt dotyczy efektywnego rozwiązywania układów równań liniowych poprzez dekompozycję QR z użyciem transformacji Householdera. Zastosowanie programowania równoległego i rozproszonego ma na celu przyspieszenie obliczeń, co jest kluczowe w przypadku dużych zestawów danych.

1.1. Wymagania

Projekt został zaimplementowany wykorzystując język C++ w standardzie C++17, zintegrowany z biblioteką UPC++ będący interfejsem dla asynchronicznego równoległego programowania, umożliwia efektywne zarządzanie równoległością, poprzez abstrakcję takich koncepcji jak zdalne wywołania procedur, globalna przestrzeń adresowa i efektywne przesyłanie danych między wątkami. Dzięki temu możliwe jest tworzenie skalowalnych aplikacji o wysokiej wydajności, które są niezbędne w obliczeniach naukowych, inżynierii i przetwarzaniu dużych zbiorów danych.

1.2. Uruchomienie

Aby uruchomić projekt, należy podjąć kilka kroków. Na początku powinniśmy sklonować repozytorium projektu, używając komendy "git clone https://github.com/mdmkl6/SRIR_QR_decomposition_Project". Następnie, przechodzimy do katalogu projektu za pomocą polecenia "cd SRIR_QR_decomposition_Project/Project2".

Przed uruchomieniem programu, ważne jest, aby skonfigurować środowisko UPC++. Aby to zrobić, należy uruchomić następujące polecenie w wierszu poleceń: “source /opt/nfs/config/source_upcxx_2023.3.sh”
To polecenie załaduje niezbędne zmienne środowiskowe i skonfiguruje środowisko tak, aby można było korzystać z biblioteki UPC++. Ważne jest, aby to polecenie zostało wykonane w każdym nowym oknie terminala, w którym planujemy pracować z UPC++.

Po wykonaniu powyższego polecenia, jesteśmy gotowi do skompilowania i uruchomienia projektu zgodnie z instrukcjami zawartymi w pliku Makefile.

Plik Makefile zawiera następujące reguły:

- compile, która kompiluje projekt.
- run, która uruchamia program.
- clean, która usuwa pliki wynikowe projektu.
- all, która wykonuje powyższe reguły sekwencyjnie.

Możemy również podać wartości “file={plik}”, gdzie w miejsce {plik} wpisujemy ścieżkę do pliku z macierzą A (domyślnym plikiem jest "matrix.txt"), oraz “outfile={plik}”, gdzie w miejsce {plik} wpisujemy ścieżkę, do której zostaną zapisane wyniki.
W pliku Makefile, poza kompilacją i uruchamianiem aplikacji, możemy również uruchamiać testy za pomocą komendy make all-test. Wśród dostępnych reguł są:

- compile-test, która kompiluje plik testujący.
- run-test, która uruchamia plik testujący.
- clean-test, która usuwa pliki plik testujący.
- all-test, która wykonuje powyższe reguły testujące sekwencyjnie

Przykładowy test, który możemy uruchomić, wczytuje zadaną macierz z pliku, przeprowadza dekompozycję na macierze Q i R, a następnie porównuje wczytaną macierz A z iloczynem macierzy Q i R. Jest to istotny proces testowy, który pozwala na sprawdzenie poprawności działania programu oraz wykrycie ewentualnych błędów.

1.3. Teoria i Algorytm

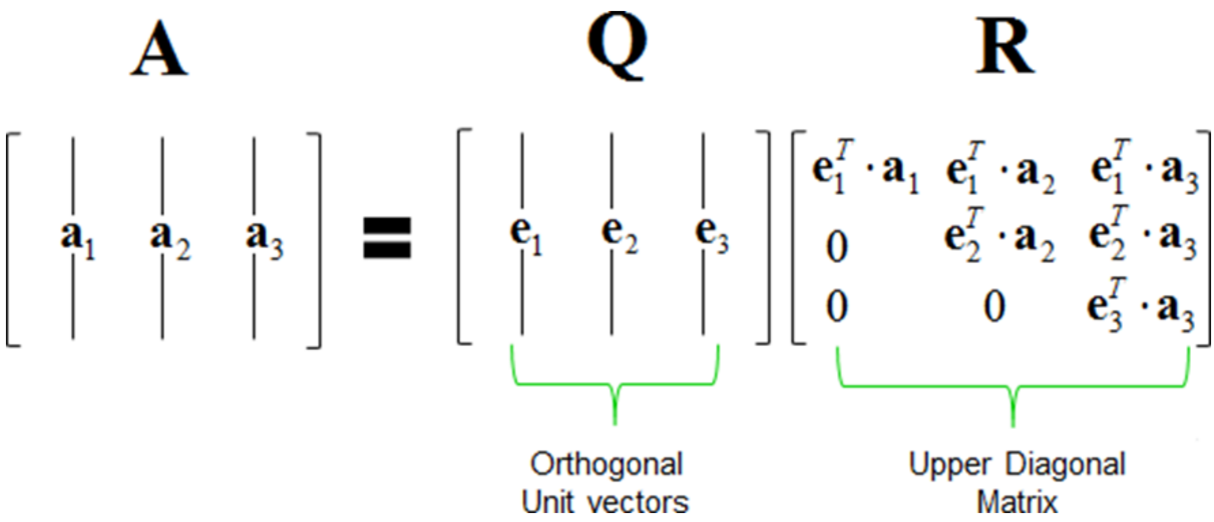
1.3.1. Podstawy Rozkładu QR

Rozkład QR macierzy A polega na przedstawieniu jej jako iloczynu macierzy ortogonalnej Q i macierzy trójkątnej górnej R, tzn A=QR. Macierz ortogonalna to macierz, której transpozycja jest równocześnie jej odwrotnością, czyli $Q^T = Q^{-1}$ Macierz trójkątna górna to macierz, w której wszystkie elementy poniżej głównej przekątnej są równe zero.

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.



1.3.2. Transformacje Householdera

Transformacja Householdera jest techniką wykorzystywaną do wprowadzenia zer w wybranych miejscach macierzy. Dla wektora x możemy znaleźć macierz Householdera H, taką że Hx jest wektorem, którego wszystkie elementy poza pierwszym są zerami. Macierz Householdera jest symetryczna i ortogonalna, a jej postać jest dana przez:

$$H = I - 2vv^T$$

gdzie v jest odpowiednio dobranym wektorem, a I jest macierzą jednostkową.

1.3.3. Rozkład QR Metodą Householdera

Aby obliczyć rozkład QR macierzy A za pomocą transformacji Householdera, stosujemy serię transformacji Householdera do macierzy A w taki sposób, aby wprowadzić zera poniżej głównej przekątnej. Po zastosowaniu n transformacji Householdera otrzymujemy macierz trójkątną górną R, a iloczyn odpowiadających macierzy Householdera daje macierz ortogonalną Q.

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.

- 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 symetryczna $H = H^T = H^{-1}$, 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 $\alpha = \pm \ ||\alpha||_2$, z odpowiednim znakiem, aby uniknąć anulowania się wartości.

for $k = 1$ **to** n

$\quad \alpha_k = -\text{sign}(a_{kk})\sqrt{a_{kk}^2 + \cdots + a_{mk}^2}$

$\quad \boldsymbol{v}_k = \begin{bmatrix} 0 & \cdots & 0 & a_{kk} & \cdots & a_{mk} \end{bmatrix}^T - \alpha_k \boldsymbol{e}_k$

$\quad \beta_k = \boldsymbol{v}_k^T \boldsymbol{v}_k$

$\quad$ **if** $\beta_k = 0$ **then**

$\qquad$ continue with next k

$\quad$ **for** $j = k$ **to** n

$\qquad \gamma_j = \boldsymbol{v}_k^T \boldsymbol{a}_j$

$\qquad \boldsymbol{a}_j = \boldsymbol{a}_j - (2\gamma_j / \beta_k)\boldsymbol{v}_k$

$\quad$ **end**

end

2. Działanie programu

Program rozpoczyna swoje działanie od inicjalizacji zmiennych oraz UPC++, co umożliwia korzystanie z mechanizmów równoległych. Na weźle głównym (master) przeprowadzane jest wczytanie macierzy z pliku, które następnie jest rozgłaszane do wszystkich procesów.

Dla każdej kolumny macierzy A, program przeprowadza szereg operacji. Na początku obliczane są wymiary podmacierzy, które mają być przetwarzane przez każdy z procesów. Następnie obliczany jest wektor Householdera oraz jego norma, a wartość tej normy jest rozgłaszana do pozostałych węzłów. Jeżeli wartość normy jest większa od zera, to macierz p jest rozpraszana pomiędzy procesy metodą `scatter`.

```cpp
void scatter(int myid, matrix *target, int recvstart, int count)
{
    upcxx::global_ptr<double> matrix;
    if(myid==0)
    {
        matrix = upcxx::new_array<double>(_numrows * _numcols);
        double *local_matrix = matrix.local();
        for (int i = 0; i < _numrows * _numcols; i++) {
            local_matrix[i] = _data[0][i];
        }
        matrix = upcxx::broadcast(matrix, 0).wait();
        for(int i = 0; i < count; i++)
            target->_data[0][i]=_data[0][recvstart+i];
    }
    else{
        matrix = upcxx::broadcast(matrix, 0).wait();
        for(int i = 0; i < count; i++)
            target->_data[0][i]=upcxx::rget(matrix+recvstart+i).wait();
    }       mdmkl6, 2 days ago • add Project2 …
}
```

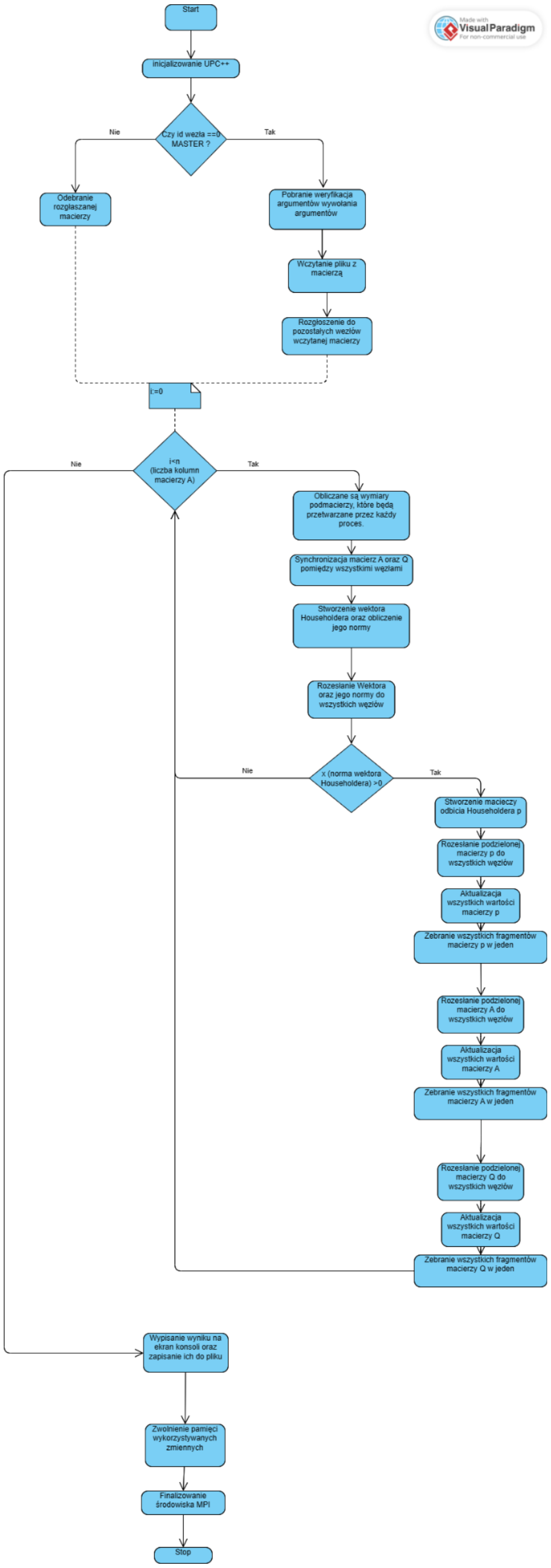
Każdy z procesów oblicza część macierzy mat, która zostanie użyta do skonstruowania macierzy p. Następnie, funkcja `gather` zbiera fragmenty macierzy mat z powrotem do macierzy p.

```cpp
void gather(int myid, matrix *target, int sendstart, int count)
{
    upcxx::global_ptr<double> matrix;
    if(myid==0)
    {
        matrix = upcxx::new_array<double>(target->_numrows * target->_numcols);
        matrix = upcxx::broadcast(matrix, 0).wait();

        int i = 0;
        for(; i < count; i++)
            target->_data[0][sendstart+i]=_data[0][i];

        upcxx::barrier();
        double *local_matrix = matrix.local();
        for(; i < target->_numrows * target->_numcols; i++)
            target->_data[0][sendstart+i]=local_matrix[i];
    }
    else{
        matrix = upcxx::broadcast(matrix, 0).wait();
        for(int i = 0; i < count; i++)
            upcxx::rput(_data[0][i], matrix+sendstart+i).wait();
        upcxx::barrier();
    }       mdmkl6, 2 days ago • add Project2 …
}
```

Wartości macierzy A oraz Q są aktualizowane w trakcie wykonywania programu. Kiedy wszystkie operacje zostaną wykonane, na głównym węźle wyniki są wyświetlane oraz zapisywane. Następnie, pamięć dynamicznie alokowana dla obiektów A, Q, p, matTmp i vec jest zwalniana, a funkcja `upcxx::finalize()` finalizuje środowisko UPC++.



3. Podsumowanie wyniki oraz wnioski

3.1. Funkcjonalność Programu

Program przyjmuje na wejściu macierz i zwraca dwie macierze: Q (ortogonalną) i R (trójkątną górną), które są wynikiem dekompozycji QR macierzy wejściowej. Program korzysta z metody Householdera do obliczenia transformacji, a także wykorzystuje programowanie równoległe do przyspieszenia obliczeń.

3.2. Przykładowe Wywołania Programu

Użytkownik może wprowadzić macierz wejściową, a program zwróci odpowiednie macierze Q i R. Przykładowe wywołania programu mogą obejmować różne rozmiary i typy macierzy.

```
9szkaradek@stud204-13:~/Desktop/Systemy_równoległe_i_rozproszone/SRIR_QR_decomposition_Project/Project2$ make all
/opt/nfs/config/station204_name_list.sh 1 16 >nodes; \
  /opt/nfs/berkeley_upcxx-2023.3.0/bin/upcxx -O2 QR.cpp utils.h -o QR.out
/opt/nfs/berkeley_upcxx-2023.3.0/bin/upcxx-run -shared-heap 256M -n 4  -ssh-servers stud204-01,stud204-02,stud204-03,stud204-0
4,stud204-05,stud204-06,stud204-07,stud204-08,stud204-09,stud204-10,stud204-11,stud204-12,stud204-13,stud204-14,stud204-15,stu
d204-16,stud204-01,stud204-02,stud204-03,stud204-04,stud204-05,stud204-06,stud204-07,stud204-08,stud204-09,stud204-10,stud204-
11,stud204-12,stud204-13,stud204-14,stud204-15,stud204-16,  QR.out ../matrix.txt ../output.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/Project2$ make all-test
g++ ../test.cpp -o ../test.out
./../test.out ../matrix.txt ../output.txt
Mean error: 7.08438e-12
Q*R matrix:
  1.00000   2.00000   3.00000
  2.00000   2.99999   4.00000
  3.00000   4.00000   5.00000
rm -f ./../test.out
```

```
9szkaradek@stud204-13:~/Desktop/Systemy_równoległe_i_rozproszone/SRIR_QR_decomposition_Project/Project2$ make all
/opt/nfs/config/station204_name_list.sh 1 16 >nodes; \
  /opt/nfs/berkeley_upcxx-2023.3.0/bin/upcxx -O2 QR.cpp utils.h -o QR.out
/opt/nfs/berkeley_upcxx-2023.3.0/bin/upcxx-run -shared-heap 256M -n 4  -ssh-servers stud204-01,stud204-02,stud204-03,stud204-04,stud204-05,stud204-06,stud204-
07,stud204-08,stud204-09,stud204-10,stud204-11,stud204-12,stud204-13,stud204-14,stud204-15,stud204-16,stud204-01,stud204-02,stud204-03,stud204-04,stud204-05,s
tud204-06,stud204-07,stud204-08,stud204-09,stud204-10,stud204-11,stud204-12,stud204-13,stud204-14,stud204-15,stud204-16,  QR.out ../matrix4.txt ../output.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  -0.25309  -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  -0.10847  -8.06258
 -0.00000  -0.00000  -2.82843  -3.06413
 -0.00000  -0.00000   0.00000  -2.12132

9szkaradek@stud204-13:~/Desktop/Systemy_równoległe_i_rozproszone/SRIR_QR_decomposition_Project/Project2$ make all-test
g++ ../test.cpp -o ../test.out
./../test.out ../matrix4.txt ../output.txt
Mean error: 7.40472e-12
Q*R matrix:
  1.99999   1.00000   3.00000   7.00000
  7.00000   3.00000   1.00000   2.00001
  4.00000   2.00000   0.00000   6.00000
  4.00000   2.00000   0.00000   9.00000
rm -f ./../test.out
```

3.3. Wnioski

- 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

- harp Documentation - QR Decomposition (dsc-spidal.github.io) - Dokumentacja
- https://atozmath.com/example/MatrixEv.aspx?q=qrdecomphh&q1=E1 - Algorytm oraz przykłady
- https://github.com/mdmkl6/SRIR_QR_decomposition_Project - Repozytorium projektu