Teoria współbieżności

Laboratorium 7

Zastosowanie teorii śladów do szeregowania wątków współbieżnej eliminacji Gaussa

Sebastian Piaskowy

1. Wprowadzenie

Rozważamy problem rozwiązywania układów równań liniowych metodą Gaussa z wykorzystaniem teorii śladów do szeregowania wątków. Problem można przedstawić jako $M \times x = y$, gdzie *M* jest macierzą kwadratową o rozmiarze *N*, natomiast *x* oraz *y* są wektorami.

$$\begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Dla uproszczenia zapisu będziemy stosować poniższą notację:

$$\left[\begin{array}{ccc|c} M_{1,1} & M_{1,2} & M_{1,3} & y_1 \\ M_{2,1} & M_{2,2} & M_{2,3} & y_2 \\ M_{3,1} & M_{3,2} & M_{3,3} & y_3 \end{array}\right]$$

Elementy macierzy *M* będziemy indeksować jako $M_{i,j}$. Elementy wektora *y* będziemy indeksować jako $M_{i,N+1}$.

2. Część teoretyczna

2.1. Algorytm

Przechodzimy przez kolejne kolumny macierzy, eliminując elementy znajdujące się poniżej głównej przekątnej. Dla każdej *i*-tej kolumny iterujemy po kolejnych wierszach o indeksach *k*, które są większe niż *i*. Celem jest wyzerowanie elementu $M_{k,i}$. Aby to osiągnąć, obliczamy mnożnik, mnożymy *i*-ty wiersz i odejmujemy go od *k*-tego wiersza.

Algorytm ten można przedstawić w formie poniższego pseudokodu:

```
for i in 1 to N-1:
    for k in i+1 to N:
        m_{i,k} = M_{k,i} / M_{i,i}
        for j in i to N+1:
            p_{i,j,k} = m_{i,k} · M_{i,j}
            M_{k,j} = M_{k,j} - p_{i,j,k}
```

2.2. Niepodzielne czynności wykonywane przez algorytm

Z w.w algorytmu możemy wyróżnić 3 rodzaje działań:

1. $A_{i,k}$ - $m_{k,i} = M_{k,i} / M_{i,i}$ - wyznaczenie mnożnika dla wiersza *i* do wyzerowania wiersza *k*.
2. $B_{i,j,k}$ - $n_{k,i} = M_{i,j} \cdot m_{k,i}$ - przemnożenie *j*-tego elementu wiersza *i* przez mnożnik $m_{k,i}$ w celu odjęcia go od *k*-tego wiersza,
3. $C_{i,j,k}$ - $M_{k,j} = M_{k,j} - n_{k,i}$ - odjęcie *j*-tego elementu wiersza i od wiersza *k*, otrzymujemy nową wartość $M_{k,j}$

2.3. Alfabet w sensie teorii śladów

Alfabet w sensie teorii śladów dla tego problemu możemy zapisać jako:

$$\Sigma = \{A_{i,k}, B_{i,j,k}, C_{i,j,k} : 1 \leq i < N, i \leq j \leq N+1, i < k \leq N\}$$

Jest to zbiór wszystkich niepodzielnych czynności wykonywanych przez algorytm.

2.4. Relacja zależności i niezależności

Na relację zależności składa się kilka zbiorów. W poniższych zapisach istotne są wartości indeksów. W każdym z tych zbiorów reprezentujemy takie pary operacji, w których pierwsza operacja modyfikuje wartość odczytywaną przez drugą operację:

Czynność B korzysta z wyniku A:

$$D_1 = \{(A_{i,k}, B_{i,j,k}) \mid A_{i,k}, B_{i,j,k} \in \Sigma\}$$

Czynność C korzysta z wyniku B:

$$D_2 = \{(B_{i,j,k}, C_{i,j,k}) \mid B_{i,j,k}, C_{i,j,k} \in \Sigma\}$$

Czynność C korzysta z $M_{k,j}$, które było zmodyfikowane przez wcześniejsze operacje C dla tej komórki

$$D_3 = \{(C_{i,j,k}, C_{i',j',k'}) \mid C_{i,j,k}, C_{i',j',k'} \in \Sigma \wedge i < i' \wedge j = j' \wedge k = k'\}$$

Czynność $A_{i,k}$ korzysta z $M_{i,k}$ i $M_{i,i}$, które mogą być uprzednio zmodyfikowane przez czynności C

$$D_4 = \{(C_{i,j,k}, A_{i',k'}) \mid C_{i,j,k}, A_{i',k'} \in \Sigma \wedge j = i' \wedge (k = i' \vee k = k') \wedge i < i'\}$$

Czynność $B_{i,j,k}$ korzysta z $M_{i,j}$, które mogą być uprzednio zmodyfikowane przez czynności C

$$D_5 = \{(C_{i,j,k}, B_{i',j',k'}) \mid C_{i,j,k}, B_{i',j',k'} \in \Sigma \wedge k = i' \wedge j = j' \wedge i < i'\}$$

Relację zależności wyznaczamy wzorem:

$$D = sum((\bigcup_{i=1}^{5} D_i)^{+}) \cup I_{\Sigma}$$

Relację niezależności wyznaczamy wzorem:

$$I = \Sigma^2 - D$$

2.5. Słowo w sensie teorii śladów

W kontekście eliminacji Gaussa, słowo zawierać będzie pełen alfabet, który obejmuje wszystkie niepodzielne operacje wykonywane przez algorytm. To oznacza, że tworzenie słowa można rozpocząć od pustego słowa a następnie wykonując kolejne kroki algorytmu, dodawać na jego koniec aktualnie wykonywane niepodzielne operacje. Słowo to redukuje naszą macierz do macierzy trójkątnej górnej.

2.6. Graf zależności Diekerta

Graf składa się z wierzchołków *V* i krawędzi *E*, oznaczamy go jako *G = (V, E)*. Zbiór wierzchołków *V* jest identyczny z elementami słowa wejściowego, które w naszym przypadku odpowiadają alfabetowi, czyli $V = \Sigma$. Krawędzie reprezentują zależności, więc musimy zatem wyeliminować zależności pochodne, usuwając je ze zbioru relacji zależności, który poprzednio wyznaczyliśmy .

2.7. Ogólna postać klas Foaty

Można zauważyć, że w algorytmie (2.1) w każdej iteracji głównej pętli (for *i*) wykonujemy operacje A, B i C, które nie mogą być wykonane przed poprzednią taką iteracją zatem możemy każdą taką iterację sklasyfikować w ramach trzech klas Foaty.
Dla uproszczenia zapisu wprowadzamy pomocnicze zbiory:

$$F'_{A_i} = \{A_{i,k} \mid i < k \leq N\}$$
$$F'_{B_i} = \{B_{i,j,k} \mid i < k \leq N \ \wedge 0 < j \leq N + 1\}$$
$$F'_{C_i} = \{C_{i,j,k} \mid i < k \leq N \wedge 0 < j \leq N + 1\}$$

Tworzymy klasy Foaty grupowane po 3:

$$F_i = [F'_{A_i}]_{\equiv_I^+} \frown [F'_{B_i}]_{\equiv_I^+} \frown [F'_{C_i}]_{\equiv_I^+}$$

Ogólną postać klas Foaty możemy zatem zapisać jako:

$$[F_1]_{\equiv_I^+} \frown [F_2]_{\equiv_I^+} \frown [F_3]_{\equiv_I^+} \frown \ \dots \ \frown [F_N]_{\equiv_I^+}$$

3. Implementacja

W języku Java 17 zaimplementowałem program z algorytmem wyznaczania w.w etapów oraz współbieżny algorytmem rozwiązywania układów równań liniowych współbieżną eliminacją Gaussa. Cała logika opiera się o klasę *ConcurrentGaussElimination,* która posiada metody:
- *createModel( )* - służy do wyznaczania kolejnych etapów teoretycznych dla danego układu równań liniowych (wyznaczanie alfabetu, relacji, grafu itp.)
- *solveWithSchedulers( )* - służy do rozwiązywania danego układu równań liniowych przy pomocy współbieżnego algorytmu wykorzystującego Scheduler'y.

Program generuje graf Diekerta w formacie .dot oraz renderuje go do pliku graficznego o formacie .png przy pomocy GraphvizAPI.

Program zintegrowałem ze zmodyfikowanymi klasami *Checker* oraz *Generator* z projektu "sprawdzarki", który był dołączony przez prowadzącego w wymaganiach zadania. Klasy te pozwalają na generowanie przykładu układu równań i walidacje jego rozwiązania.

Program jako wejście przyjmuje trzy argumenty, z których trzeci jest opcjonalny:
1) nazwa pliku wejściowego z układem równań
2) nazwa pliku wyjściowego z rozwiązaniem układu równań
3) rozmiar losowego układu równań do wygenerowania

W przypadku uruchomienia programu z trzecim argumentem, układ równań zamiast zostać pobrany z pliku wejściowego zostanie wygenerowany losowo o wybranym rozmiarze i zapisze się do pliku wejściowego.

Reprezentacja układu równań w pliku:
- pierwsza linijka - rozmiar macierzy N (int),
- N kolejnych linijek - kolejne wektory macierzy (float),
- ostatnia linijka - wektor wyrazów wolnych (float).