Teoria Współbierzności - Zadanie domowe 3

Jakub Frączek

31 grudnia 2024

1 Opis zadania

Zadanie polegało na kolejno:

1. Zlokalizowaniu niepodzielnych czynności wykonywanych przez algorytm eliminacji Gaussa, nazwać je oraz zbudować alfabet w sensie teorii śladów
2. Skonstruowaniu relacji zależności dla alfabetu, opisującego algorytm eliminacji Gaussa.
3. Przedstawieniu algorytmu eliminacji Gaussa w postaci ciągu symboli alfabetu.
4. Wygenerowaniu grafu zależności Diekerta
5. Przekształceniu ciąg symboli opisującegi algorytm do postaci normalnej Foaty.
6. Zaprojektowaniu i zaimplementowaniu współbieżnego algorytmu eliminacji Gaussa.

2 Wykorzystane technologie

Do implementacji zadania został wykorzystany język Python w wersji 12.3.8 oraz moduły:

- sys
- os
- warnings
- time
- itertools
- python-graphviz 0.20.1
- numpy 1.26.4
- numba 0.60.0

3 Dane techniczne

Testy algorytmu zostały przeprowadzone na komputerze z 64 bitowym systemem Windows 11, procesorem Intel Core i5-9300H 2.40 GHz, kartą graficzną Geforce GTX 1650 oraz 32 GB pamięci RAM.

4 Realizacja zadania

4.1 Część teoretyczna

W algorytmie eliminacji Gaussa można wyróżnić trzy niepodzielne operacje:

- $A_{a,b}$ - wyznaczenie ilorazu elementów macierzy: $M[b][a]/M[a][a]$
- $B_{a,i,b}$ - wyznaczenie iloczynu: $M[a][i] * A_{a,b}$
- $C_{a,i,b}$ - wyznaczenie różnicy i przypisanie do elementu: $M[b][i] = M[b][i] - B_{a,i,b}$

4.2 Część implementacyjna dotycząca teorii śladów

Pierwszym etapem było napisanie funkcji calculate_operations, która wyznacza listę kolejnych operacji na rzędach macierzy jakie powinien wykonać algorytm Gaussa w celu otrzemania macierzy trójkątnej górnej. Wyznaczona lista po spłaszczeniu za pomocą funkcji get_alphabet zawiera unikalne symbole, które jednoznacznie wyznaczają alfabet w sensie teorii śladów oraz słowo w opisujące kroki algorytmu. Następnie wykorzystując funkcję get_dependence wyznaczam relację zależności, uwzględniając przy tym jedynie bezpośrednie zależności między operacjami. Kolejno na podstawie wyznaczej relacji i alfabetu wyznaczam graf Diekerta za pomocą funkcji diekert_graph, a następnie klasy foaty dzięki funkcji get_foaty. Na końcu dzięki bibliotece python-graphviz rysuję wyznaczony graf Diekerta.

4.3 Część implementacyjna dotycząca algorytmu Gaussa

Do implementacji współbieżnej części algorytmu skorzystałem z modułu numba pozwalającego na napisanie funkcji wykonywanych na rdzeniach CUDA. Są to:

- gaussian_elimination_A_kernel - wykonująca operację $A_{a,b}$
- gaussian_elimination_B_kernel - wykonująca operację $B_{a,i,b}$
- gaussian_elimination_C_kernel - wykonująca operację $C_{a,i,b}$

W pętli w funkcji gaussian_elimination_cuda kolejno wykonuję współbieżnie wszystkie operacje znajdujące się w jednej klasie Foaty wykorzystując wyżej wymienione funkcje. Finalnie wyliczam rozwiązanie układu już na CPU za pomocą funkcji to_identity_and_solve.

5 Wyniki

5.1 Dla przykładowego wejścia

W treści zadania podane zostało przykładowe wejście do programu:

```
3
2.0 1.0 3.0
4.0 3.0 8.0
6.0 5.0 16.0
6.0 15.0 27.0
```

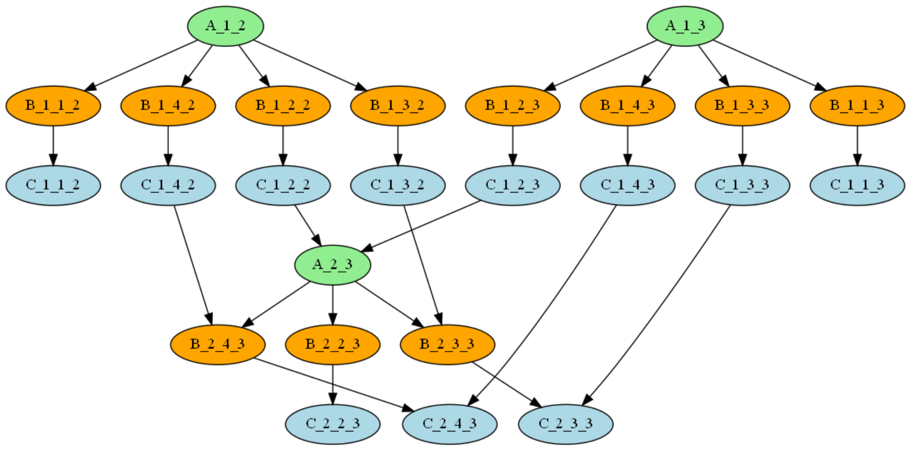
Otrzymane wyniki:

$$\begin{aligned}\Sigma = \{&A_{12}, B_{112}, C_{112}, B_{122}, C_{122}, B_{132}, C_{132}, B_{142}, C_{142}, A_{13},\\ &B_{113}, C_{113}, B_{123}, C_{123}, B_{133}, C_{133}, B_{143}, C_{143}, A_{23}, B_{223},\\ &C_{223}, B_{233}, C_{233}, B_{243}, C_{243}\}\end{aligned}$$

$$\begin{aligned}D = sym\{&(A_{12}, B_{112}), (B_{112}, C_{112}), (A_{12}, B_{122}), (B_{122}, C_{122}),\\ &(A_{12}, B_{132}), (B_{132}, C_{132}), (A_{12}, B_{142}), (B_{142}, C_{142}),\\ &(A_{13}, B_{113}), (B_{113}, C_{113}), (A_{13}, B_{123}), (B_{123}, C_{123}),\\ &(A_{13}, B_{133}), (B_{133}, C_{133}), (A_{13}, B_{143}), (B_{143}, C_{143}),\\ &(C_{123}, A_{23}), (C_{122}, A_{23}), (A_{23}, B_{223}), (B_{223}, C_{223}),\\ &(C_{132}, B_{233}), (A_{23}, B_{233}), (C_{133}, C_{233}), (B_{233}, C_{233}),\\ &(C_{142}, B_{243}), (A_{23}, B_{243}), (C_{143}, C_{243}), (B_{243}, C_{243})\}^{+} \cup I_{\Sigma}\end{aligned}$$

$$\begin{aligned}t = [w]_{\equiv_I^+} =[<&A_{12}, B_{112}, C_{112}, B_{122}, C_{122}, B_{132}, C_{132}, B_{142}, C_{142}, A_{13},\\ &B_{113}, C_{113}, B_{123}, C_{123}, B_{133}, C_{133}, B_{143}, C_{143}, A_{23}, B_{223},\\ &C_{223}, B_{233}, C_{233}, B_{243}, C_{243} >]\end{aligned}$$

$$\begin{aligned}t = [<A>]_{\equiv_I^+} = \{&[\{A_{12}, A_{13}\}]_{\equiv_I^+}\\ &\frown [\{B_{112}, B_{122}, B_{132}, B_{142}, B_{113}, B_{123}, B_{133}, B_{143}\}]_{\equiv_I^+}\\ &\frown [\{C_{112}, C_{122}, C_{132}, C_{142}, C_{113}, C_{123}, C_{133}, C_{143}\}]_{\equiv_I^+}\\ &\frown [\{A_{23}\}]_{\equiv_I^+}\\ &\frown [\{B_{223}, B_{233}, B_{243}\}]_{\equiv_I^+}\\ &\frown [\{C_{223}, C_{233}, C_{243}\}]_{\equiv_I^+}\}\end{aligned}$$



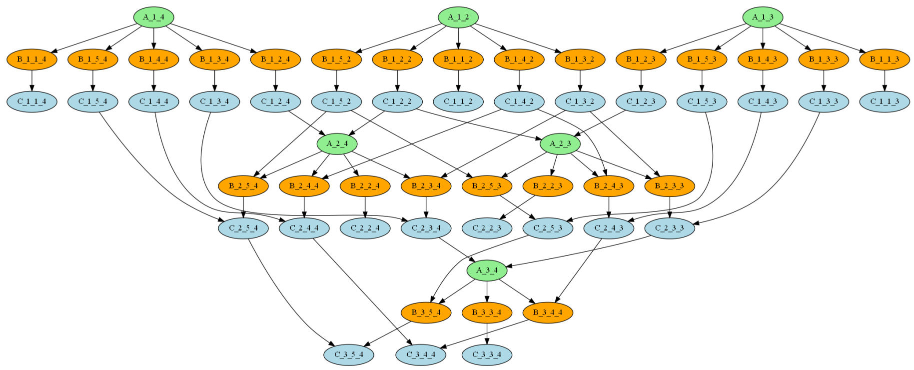
Rysunek 1: Graf Diekerta dla danego przykładu

Rozwiązanie układu równań:

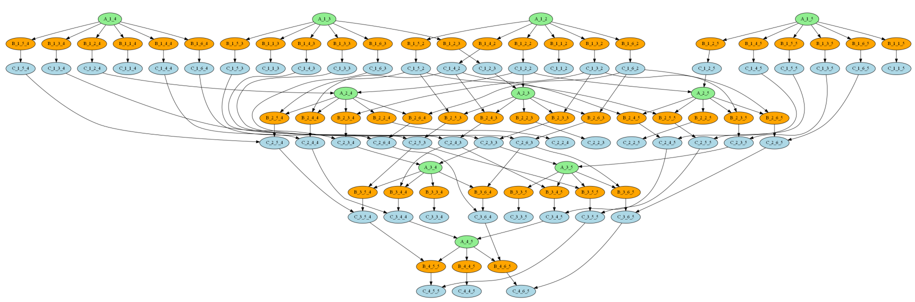
```
3
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
1.0 1.0 1.0
```

5.2 Porównanie Grafu Diekerta dla różnych rozmiarów układu

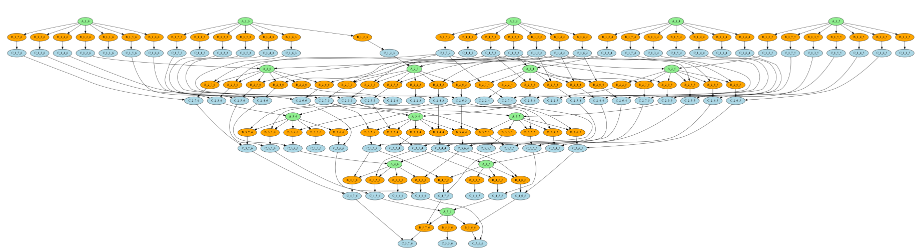
Na trzech poniższych wykresach znajduje się porównanie grafów Diekerta dla macierzy o rozmiarach kolejno 4, 5, 6. Grafy dla każdej macierzy o tych rozmiarach będą wyglądać tak samo bez względu na wartości.



Rysunek 2: Graf Diekerta dla macierzy o rozmiarze 4



Rysunek 3: Graf Diekerta dla macierzy o rozmiarze 5



Rysunek 4: Graf Diekerta dla macierzy o rozmiarze 6

5.2.1 Testy ze sprawdzarką

Przeprowadziłem testy dla różnych rozmiarów macierzy i za każdym razem otrzymany wynik był poprawny. Dodatkowo wykorzystując generator ze sprawdzarki przeprowadziłem testy czasów wykonania zaimplementowanego algorytmu Gaussa, co zostało pokazane na poniższym wykresie.

Rysunek 5: Porównanie czasów działania algorytmu Gaussa do rozmiarów macierzy

6 Wnioski

1. Zrozumienie i wykorzystanie teorii śladów pozwala na efektywne rozbicie problemu na części pierwsze, a następnie ich wykorzystanie do implementacji algorytmu współbieżnego
2. Zadanie pozwoliło mi nauczyć się i zrozumieć jak działa uruchamianie kodu na GPU.
3. Program poprawnie wyznacza obiekty matematyczne związane z teorią śladów oraz rozwiązuje układ równań.

7 Źródła

- Wykład z przedmiotu Teoria Współbieżności
- Materiały zamieszczone w sytemie UPEL
- https://developer.nvidia.com