

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Лабораторна робота №5
“Алгоритм Флойда”

Виконав:
студент групи ПМі-31
Дудинець Олександр

Львів 2024

Мета роботи

Метою цієї лабораторної роботи було розробити програму для знаходження найкоротших шляхів між усіма парами вершин в орієнтованому зваженому графі за допомогою алгоритму Флойда-Воршалла. Завдання включало реалізацію як послідовного, так і паралельного варіантів алгоритму. Також потрібно було обчислити час виконання кожного алгоритму, визначити прискорення та ефективність паралельного алгоритму для різної кількості потоків та розмірів графів.

Опис ключових частин коду

1. Генерація графу (*generateGraph*):

- a. **Призначення:** Створює випадковий орієнтований зважений граф у вигляді матриці суміжності.
- b. **Реалізація:** Для кожної пари вершин (i, j) :
 - i. Якщо $i == j$, встановлюється вага 0.
 - ii. З ймовірністю 20% між вершинами створюється ребро з випадковою вагою від 1 до 10.
 - iii. В іншому випадку вага встановлюється як *INF* (нескінченність), що означає відсутність прямого ребра.

2. Послідовна реалізація алгоритму Флойда-Воршалла (*computeFloydSequential*):

- a. **Призначення:** Знаходить найкоротші шляхи між усіма парами вершин у графі.
- b. **Реалізація:**
 - i. Створюється матриця відстаней *dist*, ініціалізована значеннями з матриці графу.
 - ii. Три вкладені цикли по k, i, j оновлюють $dist[i][j]$, якщо знайдено коротший шлях через вершину k .

3. Паралельна реалізація алгоритму Флойда-Воршалла (*computeFloydParallel*):

- a. **Призначення:** Прискорює виконання алгоритму шляхом паралельної обробки.
- b. **Реалізація:**
 - i. Зовнішній цикл по k залишається послідовним.

- ii. На кожній ітерації k рядки матриці $dist$ розподіляються між потоками.
- iii. Кожен потік обробляє свій підмножину рядків, виконуючи оновлення $dist[i][j]$ незалежно.

4. Перевірка еквівалентності результатів (*areMatricesEqual*):

- a. **Призначення:** Перевіряє, чи матриці відстаней, отримані послідовним та паралельним алгоритмами, є ідентичними.
- b. **Реалізація:** Порівнює відповідні елементи двох матриць $dist$ по всіх індексах i та j .

5. Обчислення прискорення та ефективності:

- a. **Функція *calculateSpeedup*:** Обчислює прискорення як відношення часу виконання послідовного алгоритму до часу паралельного.
- b. **Функція *calculateEfficiency*:** Обчислює ефективність як відношення прискорення до кількості потоків.

6. Аналіз (*benchmark*):

- a. **Призначення:** Запускає обидва алгоритми, вимірює час їх виконання та аналізує результати.
- b. **Реалізація:**
 - i. Генерує граф заданого розміру.
 - ii. Виконує послідовний та/або паралельний алгоритми залежно від параметрів.
 - iii. Вимірює час виконання кожного алгоритму.
 - iv. Виводить результати та обчислює прискорення й ефективність.

7. Параметри командного рядка:

- a. **Формат:** $\langle n \rangle \langle threads \rangle \langle a \rangle \langle b \rangle [runSequential] [runParallel]$
- b. **Пояснення:**
 - i. n — кількість вершин у графі.
 - ii. $threads$ — кількість потоків для паралельного алгоритму.
 - iii. a, b — індекси вершин для знаходження найкоротшого шляху між ними.
 - iv. *runSequential* — прапорець для запуску послідовного алгоритму (1 — запустити, 0 — пропустити).
 - v. *runParallel* — прапорець для запуску паралельного алгоритму.

Аналіз результатів

Програма була протестована на графі з 2000 вершинами з використанням 8 потоків. Тестування проводилося на комп'ютері з процесором Apple M1 Pro (8 високопродуктивних ядер і 2 енергоефективні ядра).

Обидва алгоритми (послідовний та паралельний) були запуснені на однакових графах, що дозволило порівняти результати та переконатися у їхній коректності. Результати обчислення найкоротшого шляху для обох варіантів алгоритмів були однаковими.

Отримані результати швидкості наведені нижче:

- **Послідовний алгоритм:**
 - **Час:** 25673 мс (26.6 с)
- **Паралельний алгоритм:**
 - **Час:** 8399 мс (8.3 с)
 - **Прискорення:** 3.05x
 - **Ефективність:** 38%

```
- Summary:  
  - Nodes: 2000  
  - Threads: 8  
  - Sequential time: 25673ms  
  - Parallel time: 8399ms  
  - Speedup: 3.05667x  
  - Efficiency: 38% (took 8399ms vs 3209ms ideal)  
  - Shortest paths equal: Yes
```

Паралельний алгоритм демонструє значні переваги для великих графів. Проте, при використанні 8 потоків ми не отримали 8-кратного прискорення, оскільки ідеальне прискорення рідко досягається в реальних умовах.

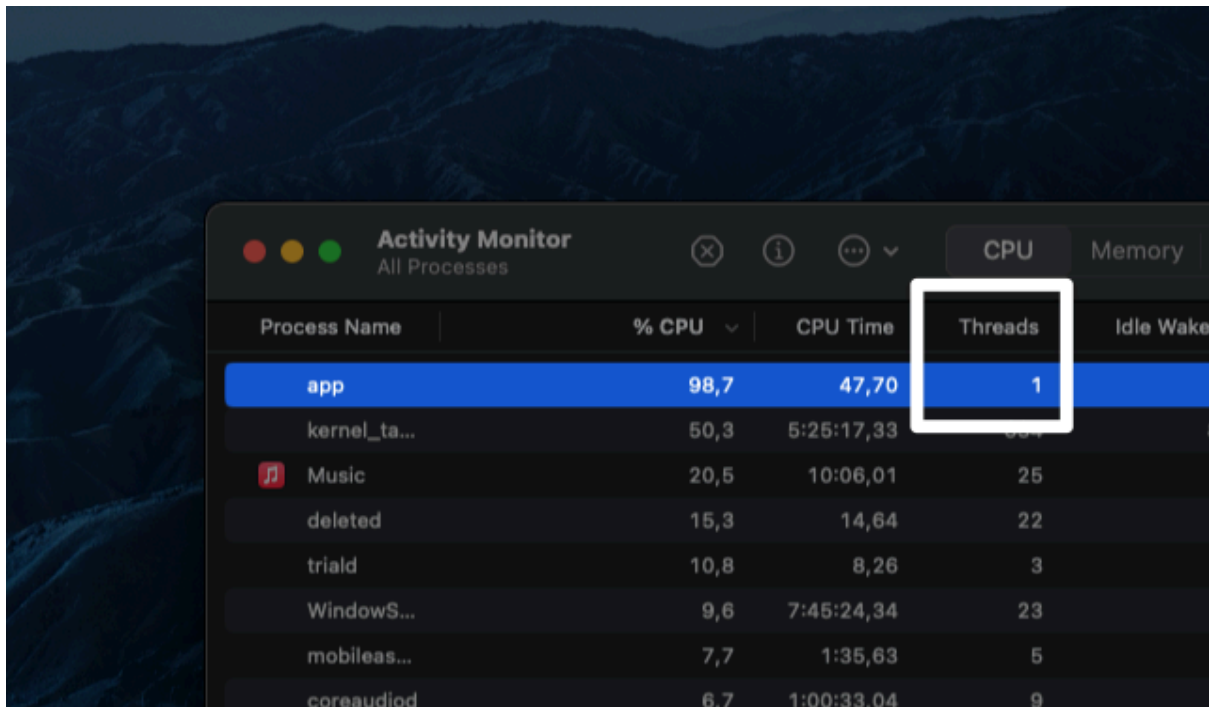
Основними причинами цього є накладні витрати, пов'язані з управлінням потоками, синхронізацією та розподілом задач між ними. Наприклад, для створення потоків, обміну даними між ними та координації їх роботи витрачається додатковий час і ресурси процесора.

Крім того, не всі операції можуть бути виконані паралельно з однаковою ефективністю, і частина алгоритму залишається послідовною, що обмежує максимальне прискорення, досяжне за законом Амдала.

Також обмеження наявних апаратних ресурсів, таких як кеш-пам'ять і смуга пропускання пам'яті, можуть призводити до зниження продуктивності при збільшенні кількості потоків.

Примітка: Прискорення обчислювалося як відношення часу виконання послідовного алгоритму до часу виконання паралельного алгоритму (наприклад, $25673 \text{ мс} / 8399 \text{ мс} = 3.05x$). Ефективність обчислювалась як відношення прискорення до кількості потоків (наприклад, $3.05 / 8 \approx 38\%$).

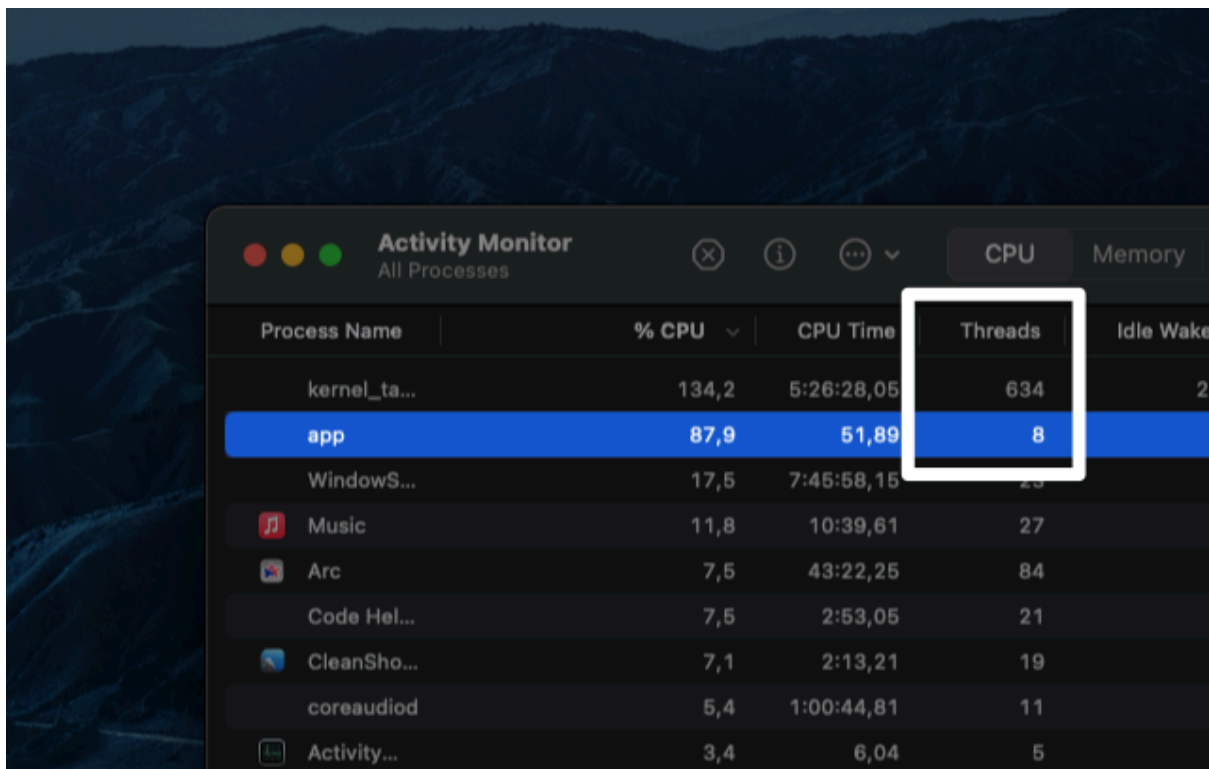
На знімках екрану нижче зображено диспетчер ресурсів комп'ютера. Видно, що послідовний режим програми використовує лише один потік:



The screenshot shows the macOS Activity Monitor window with the 'CPU' tab selected. The 'Threads' column is highlighted with a white box, showing the value '1' for the 'app' process.

Process Name	% CPU	CPU Time	Threads	Idle Wake
app	98,7	47,70	1	
kernel_ta...	50,3	5:25:17,33		
Music	20,5	10:06,01	25	
deleted	15,3	14,64	22	
triald	10,8	8,26	3	
WindowS...	9,6	7:45:24,34	23	
mobileas...	7,7	1:35,63	5	
coreaudiod	6,7	1:00:33,04	9	

Тоді як паралельний – 8:



The screenshot shows the macOS Activity Monitor window with the 'CPU' tab selected. The 'Threads' column is highlighted with a white box, showing the value '8' for the 'app' process.

Process Name	% CPU	CPU Time	Threads	Idle Wake
kernel_ta...	134,2	5:26:28,05	634	2
app	87,9	51,89	8	
WindowS...	17,5	7:45:58,15	28	
Music	11,8	10:39,61	27	
Arc	7,5	43:22,25	84	
Code Hel...	7,5	2:53,05	21	
CleanSho...	7,1	2:13,21	19	
coreaudiod	5,4	1:00:44,81	11	
Activity...	3,4	6,04	5	

Випадок некратності розмірності матриці кількості потоків

Алгоритм був протестований у випадках, коли кількість вершин не кратна кількості потоків. Залишкові рядки були рівномірно розподілені між потоками. Це забезпечило збалансоване навантаження та ефективне використання ресурсів без втрати коректності.

Висновок

У цій лабораторній роботі був реалізовано як послідовний, так і паралельний варіанти алгоритму Флойда-Воршалла для знаходження найкоротших шляхів у графі. Паралельний алгоритм продемонстрував значне прискорення на великих графах, що підтверджує доцільність використання паралельних обчислень для задач такого типу.

Під час роботи я поглибив свої знання з паралельного програмування на мові C++, зокрема з використанням бібліотеки *std::thread*. Я також ознайомився з особливостями паралелізації алгоритмів, які мають послідовні залежності.

Однак ефективність паралельного алгоритму була нижчою за очікувану через накладні витрати на управління потоками та обмеження, пов'язані з послідовними частинами алгоритму. У майбутньому можна розглянути можливість оптимізації коду шляхом зменшення накладних витрат, наприклад, використовуючи пул потоків (замість створення та приєднання потоків на кожній ітерації можна створити потоки один раз і використовувати їх протягом усього алгоритму) або інші методи управління потоками.