

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

**Лабораторна робота №2**  
“Розпаралелення множення матриць”

Виконав:  
студент групи ПМі-31  
Дудинець Олександр

Львів 2024

## Мета роботи

Метою лабораторної роботи було розробити програму для обчислення добутку двох матриць з використанням послідовного та паралельного алгоритмів. Завдання включало також обчислення часу виконання кожного алгоритму, прискорення та ефективності паралельного алгоритму для різної кількості потоків та розмірів матриць.

## Опис ключових частин коду

- **Генерація матриць** (*generateMatrix*): Функція створює двовимірну динамічну матрицю розміром  $n \times m$  і заповнює її випадковими числами в діапазоні від 0 до 99. Використовується для створення вхідних даних для алгоритмів множення матриць.
- **Множення рядка на стовпчик** (*multiplyRowByColumn*): Функція обчислює добуток рядка першої матриці на стовпчик другої матриці. Вона використовується для обчислення елементів результуючої матриці при множенні.
- **Послідовне обчислення матриць** (*multiplyMatricesSequential*): Функція виконує обчислення добутку двох матриць послідовно, рядок за рядком, використовуючи функцію *multiplyRowByColumn* для розрахунку значень елементів результуючої матриці.
- **Паралельне обчислення матриць** (*multiplyMatricesParallel*): Функція виконує обчислення добутку двох матриць за допомогою декількох потоків. Рядки матриці розподіляються між потоками з урахуванням залишку у разі, якщо кількість рядків не кратна кількості потоків. Кожен потік обчислює результати для своєї підмножини рядків.
- **Порівняння матриць** (*areMatricesEqual*): Функція перевіряє, чи є дві матриці ідентичними за значеннями. Використовується для перевірки коректності результатів паралельних обчислень шляхом порівняння з результатами послідовних обчислень.

Програма підтримує передачу аргументів для визначення розмірності матриць та кількості потоків, що дозволяє легко змінювати параметри для тестування (*Usage: <n> <m> <l> <threads> [runSequential] [runParallel]*). Наприклад: *./app 2000 3000 2000 8 0 1*.

## Аналіз результатів

Програма була протестована на матрицях розміром 1000x2000 та 2000x1000.

Тестування проводилося на комп'ютері з процесором на архітектурі ARM (Apple M1 Pro), який має 8 високопродуктивних ядер і 2 енергоефективні ядра. Було вирішено використовувати 8 потоків для максимального завантаження високопродуктивних ядер процесора.

Обидва алгоритми (послідовний та паралельний) були запущені на однакових матрицях, що дозволило порівняти результати та переконатися у їхній коректності. Результати множення для обох варіантів алгоритмів були однаковими.

Отримані результати швидкості наведені нижче:

- **Послідовний алгоритм:**
  - **Час:** 5079 *мс*
- **Паралельний алгоритм:**
  - **Час:** 965 *мс*
  - **Прискорення:** 5.26x
  - **Ефективність:** 65%

```
- Summary:  
  - Matrix size: 1000x2000 to 2000x1000  
  - Threads: 8  
  - Sequential time: 5079ms  
  - Parallel time: 965ms  
  - Speedup: 5.26321x  
  - Efficiency: 65% (took 965ms vs 634ms ideal)  
  - Matrices equal: Yes
```

Паралельний алгоритм демонструє значні переваги для великих матриць, таких як 1000x2000 та 2000x1000. Проте, при використанні 8 потоків ми не отримали 8-кратного прискорення, оскільки ідеальне прискорення рідко досягається в реальних умовах.

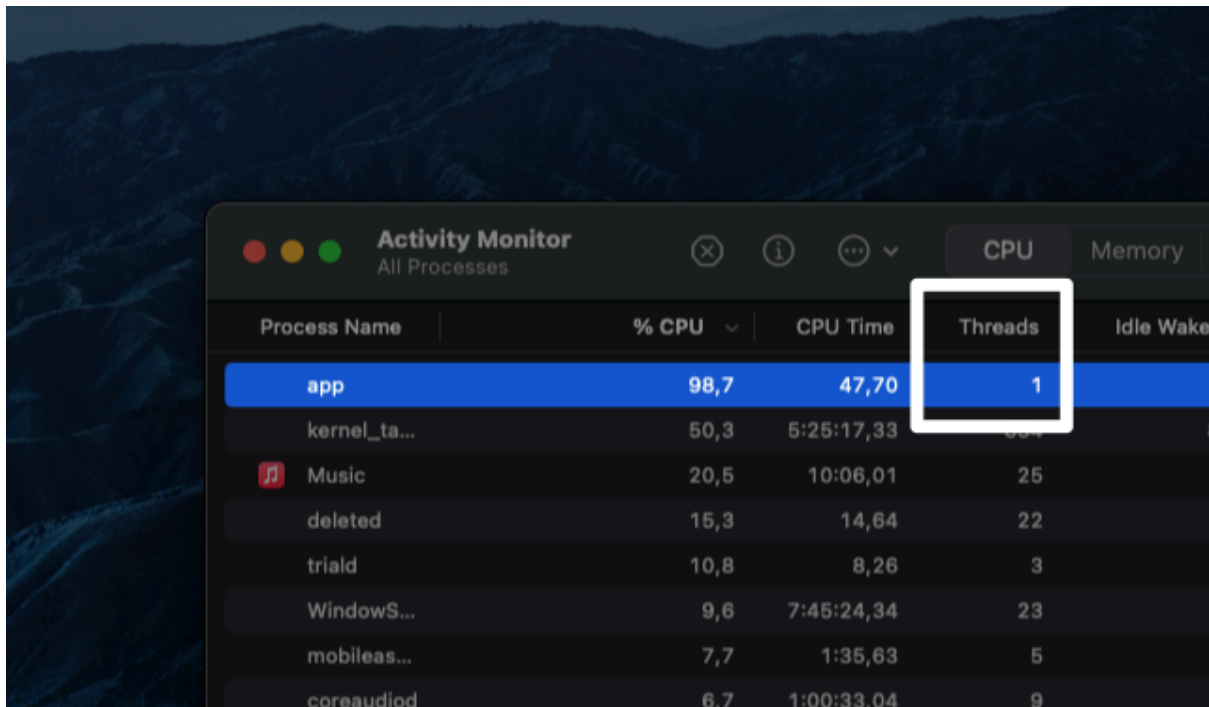
Основними причинами цього є накладні витрати, пов'язані з управлінням потоками, синхронізацією та розподілом задач між ними. Наприклад, для створення потоків, обміну даними між ними та координації їх роботи витрачається додатковий час і ресурси процесора.

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, 5079 мс / 965 мс = 5.26x). Ефективність обчислювалась як відношення прискорення до кількості потоків (наприклад, 5.26 / 8  $\approx$  65%).

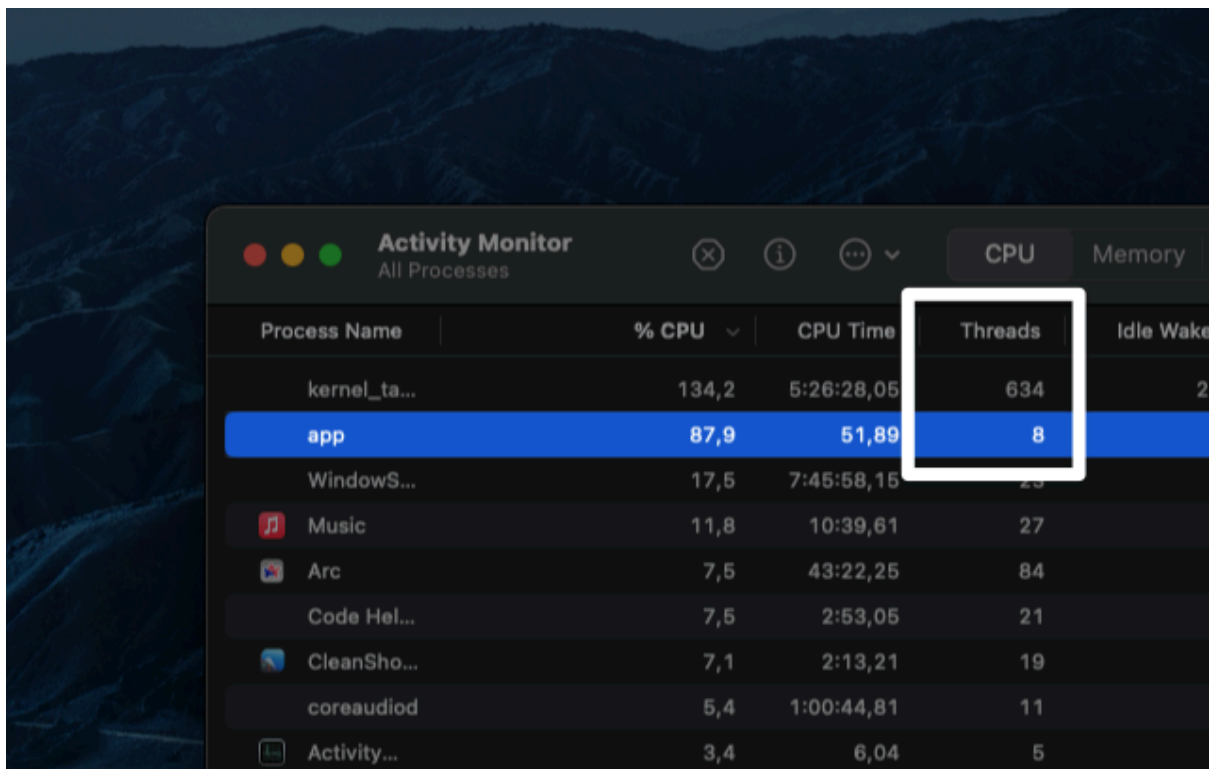
На знімках екрану нижче зображено диспетчер ресурсів комп'ютера. Видно, що послідовний режим програми використовує лише один потік:



The screenshot shows the macOS Activity Monitor window with the 'CPU' tab selected. The 'app' process is highlighted in blue, and its 'Threads' column value is 1, which is circled in white. Other processes listed include 'kernel\_ta...', 'Music', 'deleted', 'triald', 'WindowS...', 'mobileas...', and 'coreaudiod'.

Process Name	% CPU	CPU Time	Threads	Idle Wake
app	98,7	47,70	1	
kernel_ta...	50,3	5:25:17,33	634	2
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 'app' process is highlighted in blue, and its 'Threads' column value is 8, which is circled in white. Other processes listed include 'kernel\_ta...', 'WindowS...', 'Music', 'Arc', 'Code Hel...', 'CleanSho...', 'coreaudiod', and 'Activity...'.

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	23	
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 працює коректно у випадках, коли розмірність матриці не кратна кількості потоків. У такому випадку залишкові рядки розподіляються між першими  $N$  потоками (де  $N$  - залишок від поділу), що забезпечує рівномірне навантаження.

## **Висновок**

Під час виконання цієї лабораторної роботи я поглибив знання в галузі паралельних обчислень і навчився реалізовувати алгоритми для обчислення добутку двох матриць як послідовним, так і паралельним способом. Я ознайомився з принципами багатопотокового програмування на мові C++, зокрема з використанням класу `thread` для створення паралельних потоків та механізмом синхронізації потоків.

Паралельний алгоритм обчислення матриць продемонстрував значне прискорення на великих розмірностях при використанні достатньої кількості потоків. Я зробив висновок, що використання паралельних обчислень доцільне при великих розмірностях матриць, проте для менших матриць ефективність може бути нижчою через витрати ресурсів комп'ютера на створення потоків.