

МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ  
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

**Лабораторна робота №3**  
“Розв’язування системи лінійних алгебраїчних рівнянь”

Виконав:  
студент групи ПМі-31  
Дудинець Олександр

Львів 2024

## Мета роботи

Метою цієї лабораторної роботи було розробити програму для розв'язання систем лінійних алгебраїчних рівнянь (СЛАР) з використанням методу Якобі як послідовно, так і паралельно. Завдання включало також обчислення часу виконання кожного алгоритму, прискорення та ефективності паралельного алгоритму для різної кількості потоків та розмірів систем.

## Опис ключових частин коду

- **Генерація матриці та вектора** (*generateDiagonallyDominantMatrix*, *generateVector*): Для забезпечення збіжності методу Якобі було реалізовано функцію *generateDiagonallyDominantMatrix*, яка створює діагонально домінуючу матрицю розміром  $n \times n$ . Функція *generateVector* генерує вектор вільних членів  $b$  розміром  $n$ .
- **Реалізація методу Якобі** (*solveJacobiSequential*, *solveJacobiParallel*):
  - Послідовна версія (*solveJacobiSequential*): Функція виконує ітеративний процес методу Якобі для розв'язання СЛАР. На кожній ітерації обчислюється нове наближення розв'язку  $x$  на основі попереднього значення.
  - Паралельна версія (*solveJacobiParallel*): Ця функція реалізує паралельне обчислення методу Якобі, розподіляючи обчислення нових значень  $x$  між кількома потоками. Рядки матриці розподіляються між потоками з урахуванням залишку, якщо кількість рядків не кратна кількості потоків.
- **Перевірка збіжності** (*areVectorsEqual*): Функція перевіряє, чи є отримані розв'язки від послідовного та паралельного алгоритмів однаковими з точністю до заданої толерантності. Це забезпечує коректність реалізації паралельного алгоритму.
- **Обчислення прискорення та ефективності** (*calculateSpeedup*, *calculateEfficiency*): Ці функції використовуються для аналізу продуктивності паралельного алгоритму порівняно з послідовним.
- **Порівняння швидкостей** (*benchmark*): Функція *benchmark* запускає послідовний та паралельний алгоритми, вимірює час їх виконання, обчислює прискорення, ефективність та перевіряє коректність розв'язків.

Програма підтримує передачу аргументів для визначення розмірності системи та кількості потоків, що дозволяє легко змінювати параметри для тестування (*Usage: <n> <threads> [runSequential] [runParallel]*). Наприклад: *./app 2500 8 1 1*.

## Аналіз результатів

Програма була протестована на системі рівнянь розміром 2500x2500.

Тестування проводилося на комп'ютері з процесором Apple M1 Pro, який має 8 високопродуктивних ядер і 2 енергоефективні ядра. Для максимального завантаження високопродуктивних ядер процесора було використано 8 потоків.

Обидва алгоритми (послідовний та паралельний) були запущені на одній і тій самій системі рівнянь, що дозволило порівняти результати та переконатися у їхній коректності. Розв'язки, отримані від обох алгоритмів, виявилися однаковими з точністю до заданої толерантності.

Отримані результати швидкості наведені нижче:

- **Послідовний алгоритм:**
  - **Час:** 219226 мс (219 с)
- **Паралельний алгоритм:**
  - **Час:** 49104 мс (49 с)
  - **Прискорення:** 4.46x
  - **Ефективність:** 55%

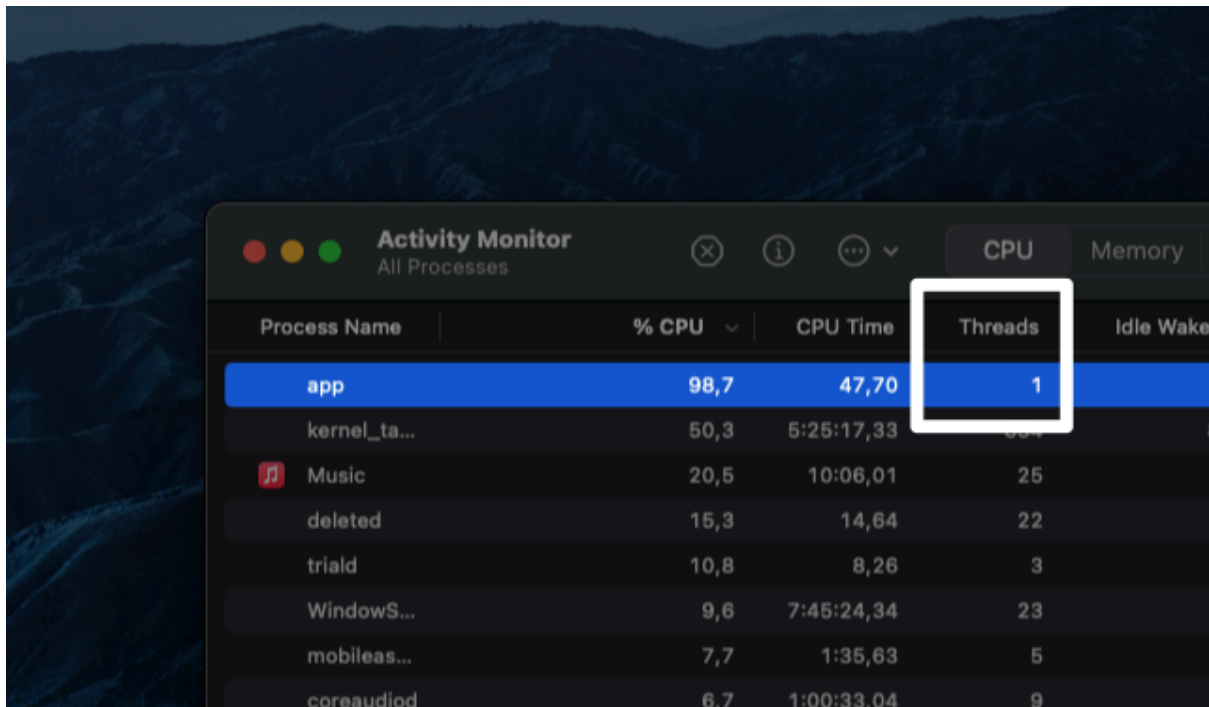
```
- Summary:  
  - System size: 2500x2500  
  - Threads: 8  
  - Sequential time: 219226ms  
  - Parallel time: 49104ms  
  - Speedup: 4.46452x  
  - Efficiency: 55%  
  - Solutions equal: Yes
```

Паралельний алгоритм демонструє значне прискорення порівняно з послідовним, особливо на великій розмірності системи рівнянь. Проте, отримане прискорення не є ідеальним (тобто не дорівнює кількості потоків) через декілька факторів:

- **Накладні витрати на управління потоками:** Створення та синхронізація потоків вимагають додаткового часу та ресурсів.
- **Обмеження пам'яті:** При великій кількості потоків можливі конфлікти при доступі до пам'яті, що може знижувати продуктивність.
- **Закон Амдала:** Відповідно до цього закону, максимальне прискорення обмежується часткою алгоритму, яка може бути паралелізована.

***Примітка:** Прискорення обчислювалося як відношення часу виконання послідовного алгоритму до часу виконання паралельного алгоритму (наприклад,  $219226 \text{ мс} / 49104 \text{ мс} = 4.46x$ ). Ефективність обчислювалась як відношення прискорення до кількості потоків (наприклад,  $4.46 / 8 \approx 55\%$ ).*

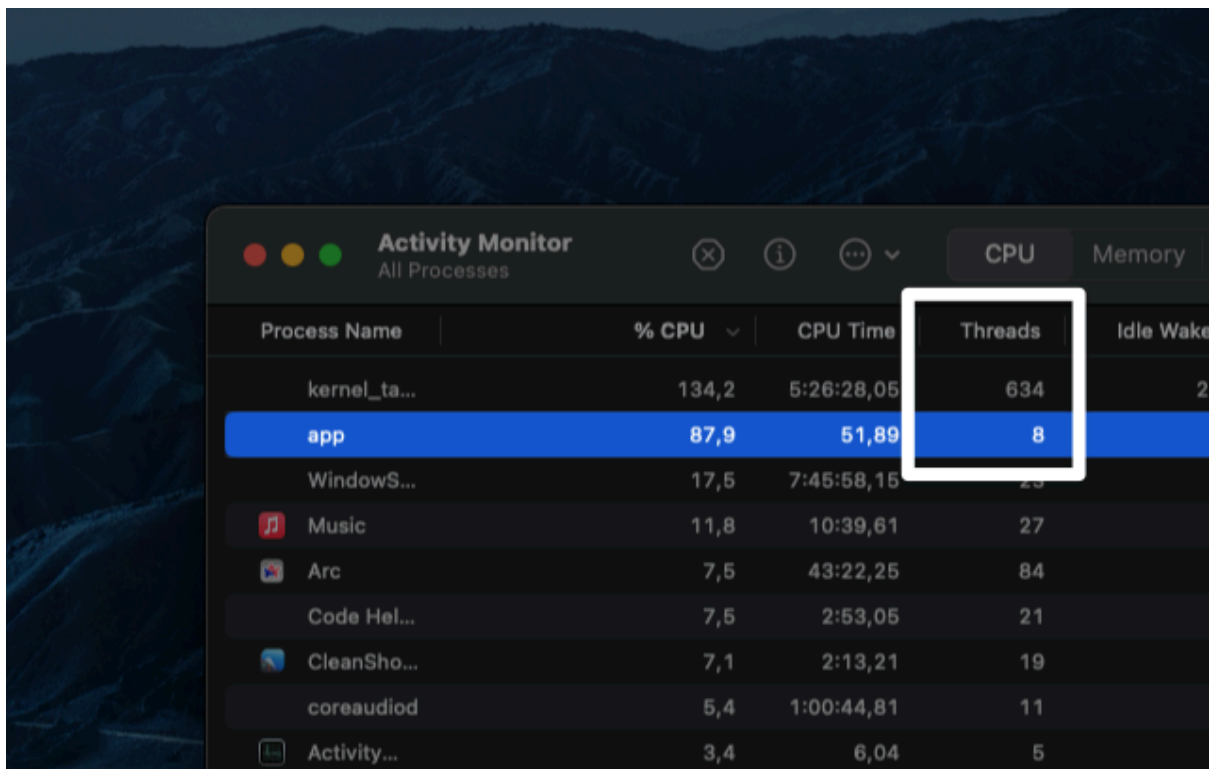
На знімках екрану нижче зображено диспетчер ресурсів комп'ютера. Видно, що послідовний режим програми використовує лише один потік:



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	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	

### Додаткові зауваження:

- **Застосування методу Якобі:** Метод Якобі є ітераційним та добре підходить для паралелізації, оскільки обчислення нового значення кожної змінної залежить тільки від значень попередньої ітерації.
- **Збіжність:** Для забезпечення збіжності методу була використана діагонально домінантна матриця, що гарантує, що ітераційний процес буде сходитися до розв'язку.
- **Можливості розширення:** Подальше покращення продуктивності може бути досягнуто шляхом використання більш ефективних механізмів синхронізації або оптимізації доступу до пам'яті.

### Випадок некратності розмірності матриці кількості потоків

Алгоритм був протестований і в умовах, коли кількість рівнянь не кратна кількості потоків. У такому випадку залишкові рядки рівномірно розподіляються між першими потоками, що забезпечує збалансоване навантаження та коректну роботу алгоритму.

### Висновок

У ході виконання цієї лабораторної роботи я поглибив свої знання в області паралельних обчислень та набув практичного досвіду в реалізації методу Якобі для розв'язання систем лінійних алгебраїчних рівнянь як послідовно, так і паралельно. Я ознайомився з особливостями багатопотокового програмування на мові C++, зокрема з використанням класу `std::thread` для створення та управління потоками.

Паралельний алгоритм розв'язання СЛАР показав суттєве прискорення на великих розмірностях систем при використанні декількох потоків. Це підтверджує доцільність використання паралельних обчислень для ресурсомістких задач.

Однак варто враховувати, що ефективність паралельних алгоритмів може знижуватися через накладні витрати на синхронізацію потоків та обмеження апаратних ресурсів. Тому при розробці паралельних алгоритмів

важливо балансувати між кількістю потоків та доступними ресурсами системи.