ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

Кафедра програмування

Паралельні та розподілені обчислення

ЛАБОРАТОРНА РОБОТА №9

Тема: «Побудова методу в інтерфейсі паралельного програмування: MPI»

Виконав:

Ст. *Лук'янчук Денис*

Група ПМІ-33

2025

Тема: «Побудова методу в інтерфейсі паралельного програмування: MPI»

Мета роботи: Ознайомитися з принципами програмно-апаратної архітектури паралельних обчислень MPI, навчитися реалізовувати розподілені обчислення на кількох процесах, оцінювати прискорення та ефективність порівняно з традиційною послідовною реалізацією на CPU. Дослідити механізми передачі даних між процесами та їх вплив на продуктивність.

Теоретичний матеріал

MPI (Message Passing Interface) — це стандарт для паралельних обчислень, який дозволяє групі незалежних процесів обмінюватися даними через передачу повідомлень. На відміну від OpenCL, де обчислення виконуються на одному пристрої (CPU або GPU), MPI використовується для кластерів, де кожен процес може працювати на окремому ядрі або навіть окремому вузлі.

Основна ідея MPI — розподіл задачі між кількома процесами, кожен з яких виконує певну частину роботи, а потім результати об'єднуються у головному процесі ("root", зазвичай rank = 0).

У лабораторній реалізовано множення матриць:

$$C_{ij} = \sum_{k=1}^{N} A_{ik} \cdot B_{kj}$$

Програма містить два варіанти реалізації:

1. Послідовне множення на CPU — три вкладені цикли без паралелізму.
2. Паралельне множення з використанням MPI.

Хід роботи

Спочатку я створив дві квадратні матриці A і B розміром N×N, які наповнюються випадковими числами від 1 до 10. Розмір N задавався в коді; для експериментів використовувались значення: 1000×1000, а також інші для перевірки.

Далі я реалізував два варіанти множення:

1. Звичайний (послідовний) — через три вкладені цикли на процесорі. Це базовий варіант, щоб знати, скільки часу займає обчислення без паралелізації.

2. Паралельний варіант через MPI.

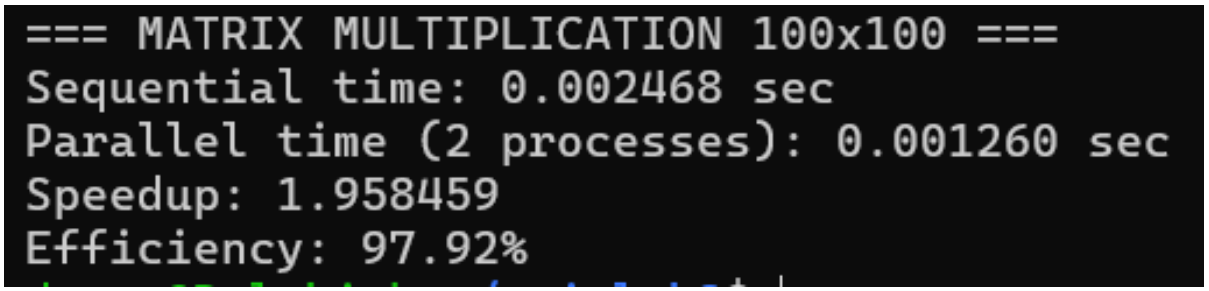
На відміну від OpenCL, де паралелізм виконує GPU, у MPI кожен процес — це окрема копія програми, яка рахує свою частину матриці.

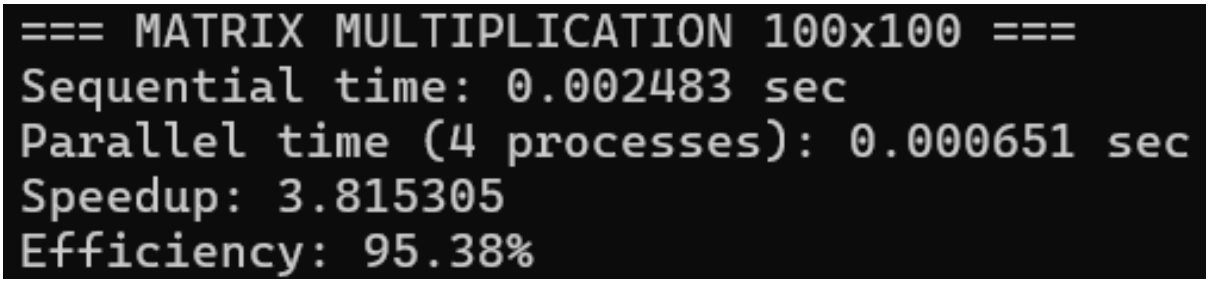
Далі для різних кількостей процесів (2, 4, 8) я вимірював час виконання та обчислював:

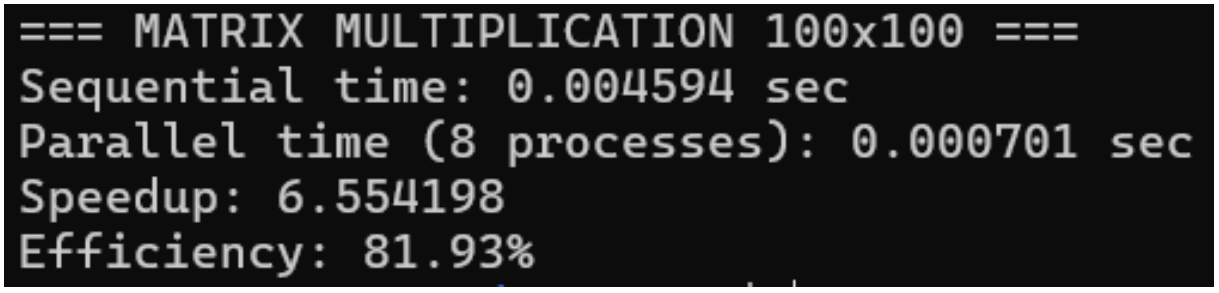
- прискорення
- ефективність

Це дозволяє оцінити, наскільки добре масштабується алгоритм при збільшенні кількості процесів.

Для початку я виконав множення **матриць розміром 100×100:**

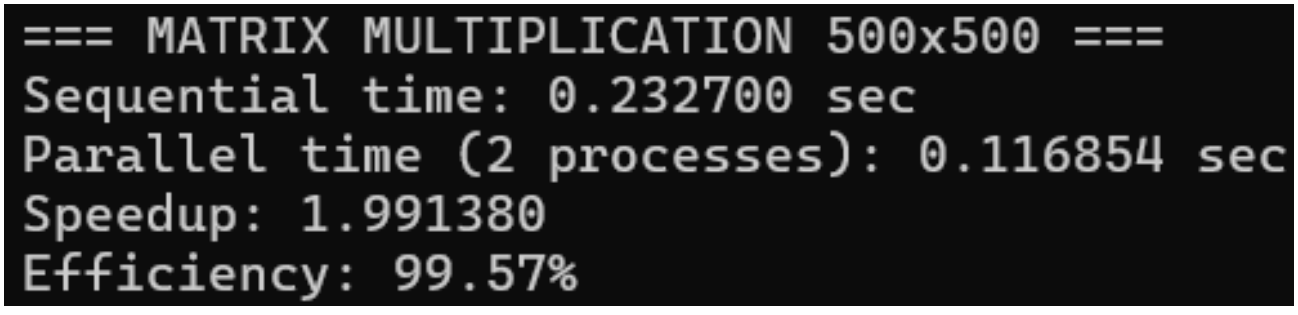






Для розміру 100×100 паралельне виконання демонструє помітне прискорення лише на невеликій кількості процесів. Проте зі збільшенням кількості процесів ефективність поступово падає через накладні витрати MPI на передачу даних та синхронізацію. Хоча прискорення є позитивним, для такого малого розміру задачі паралелізація масштабується гірше — збільшення кількості процесів не дає пропорційного виграшу в часі.

Далі я продовжив збільшувати розмірність **матриці до 500 на 500:**



```
=== MATRIX MULTIPLICATION 500x500 ===
Sequential time: 0.242175 sec
Parallel time (4 processes): 0.061673 sec
Speedup: 3.926791
Efficiency: 98.17%
```

```
=== MATRIX MULTIPLICATION 500x500 ===
Sequential time: 0.304454 sec
Parallel time (8 processes): 0.049545 sec
Speedup: 6.145042
Efficiency: 76.81%
```

Для матриці 500×500 паралельне виконання показує майже ідеальну ефективність на 2 та 4 процесах — накладні витрати мінімальні, тому ефективність майже 100%. Прискорення при збільшенні процесів збільшується. Однак при збільшенні кількості процесів до 8 ефективність суттєво падає. Це відбувається через зростання витрат на комунікацію між процесами та синхронізацію, які стають помітними при порівняно невеликому обсязі роботи на кожен процес.

Матриці 1000 на 1000:

```
=== MATRIX MULTIPLICATION 1000x1000 ===
Sequential time: 2.104642 sec
Parallel time (2 processes): 1.090464 sec
Speedup: 1.930042
Efficiency: 96.50%
```

```
=== MATRIX MULTIPLICATION 1000x1000 ===
Sequential time: 2.300762 sec
Parallel time (4 processes): 0.667646 sec
Speedup: 3.446081
Efficiency: 86.15%
```

```
=== MATRIX MULTIPLICATION 1000x1000 ===
Sequential time: 3.186529 sec
Parallel time (8 processes): 0.609071 sec
Speedup: 5.231785
Efficiency: 65.40%
```

Ситуація аналогічна попереднім прикладам. Прискорення збільшується зі збільшенням процесів, а ефективність навпаки зменшується. Але цікаво те що при матрицях 1000 на 1000 ефективність трохи гірша ніж при 500 на 500, особливо коли використовується 8 процесів.

Матриці 1500 на 1500:

```
=== MATRIX MULTIPLICATION 1500x1500 ===
Sequential time: 10.159245 sec
Parallel time (2 processes): 5.956846 sec
Speedup: 1.705474
Efficiency: 85.27%
```

```
=== MATRIX MULTIPLICATION 1500x1500 ===
Sequential time: 11.227426 sec
Parallel time (4 processes): 3.715100 sec
Speedup: 3.022106
Efficiency: 75.55%
```

```
=== MATRIX MULTIPLICATION 1500x1500 ===
Sequential time: 17.015188 sec
Parallel time (8 processes): 3.323521 sec
Speedup: 5.119628
Efficiency: 64.00%
```

Для матриці 1500×1500 паралельне множення демонструє стабільне прискорення, але ефективність поступово зменшується зі збільшенням кількості процесів. На 2 процесах прискорення є помірним через велику частку даних на кожен процес. На 4 процесах результати кращі, але ефективність падає через зростання комунікаційних витрат. При 8 процесах швидкість зростає найбільше, однак ефективність зменшується ще сильніше — накладні витрати MPI починають переважати над виграшем від додаткових процесів.

Підсумок:

- **Для малих матриць (100×100)** MPI демонструє реальне прискорення — на 2, 4 і навіть 8 процесах час виконання зменшується. Це можливо завдяки тому, що матриці маленькі та добре поміщаються в кеш, тому накладні витрати MPI майже не впливають.
- **Для матриць середнього розміру (500×500)** паралельна версія показує майже ідеальне масштабування. Ефективність наближається до 100% для 2–4 процесів, а прискорення відповідає теоретичним очікуванням. На 8 процесах ефективність падає, але прискорення все ще суттєве.

- **Для великих матриць (1000×1000 – 1500×1500)** MPI продовжує давати значний виграш — час виконання скорочується у 3–5 разів. Проте через збільшення комунікаційних обсягів ефективність поступово зменшується зі зростанням кількості процесів.

Висновок: У лабораторній роботі я навчився реалізовувати паралельне множення матриць за допомогою MPI. Експерименти показали, що MPI забезпечує реальне прискорення для всіх розмірів матриць, а найбільший виграш досягається для великих задач. На 2–4 процесах масштабування майже лінійне, тоді як при збільшенні кількості процесів ефективність зменшується через комунікаційні витрати. Загалом MPI є доцільним інструментом для прискорення обчислень при роботі з великими матрицями.