ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

# Паралельні та розподілені обчислення
# ЛАБОРАТОРНА РОБОТА №2

## Тема: «Розпаралелення множення матриць».

Виконала:
Ст. *Пелещак Вероніка*
ПМІ-35с

2025

**Тема:** Розпаралелення множення матриць.

**Завдання:** Написати програми обчислення множення двох матриць (послідовний та паралельний алгоритми). Порахувати час роботи кожної з програм, обчислити прискорення та ефективність роботи паралельного алгоритму.

## Опис програми:

1. **Ініціалізація матриць**
    - Матриці A (розміром n × m) та B (розміром m × p) заповнюються випадковими числами у діапазоні [0; 10000].
    - Результат множення зберігається у матриці C (розміром n × p).
2. **Функції програми:**
    - fillMatrix(...) – заповнення матриці випадковими числами:
        - Створює генератор випадкових чисел **mt19937** і розподіл **uniform_int_distribution.**

```
void fillMatrix(vector<vector<int>>& mat, int n, int m){
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<> dis(0, 10000);

    for(int i = 0; i < n; ++i){
        for(int j = 0; j < m; ++j){
            mat[i][j] = dis(gen);
        }
    }
}
```

- printMatrix(...) – виведення матриці на екран (у даному випадку не використовується для великих розмірів).
- inputInt(...) – коректне введення числових значень (з перевіркою).
- multiplyMatrices(...) – послідовне множення матриць.
- multiplyMatricesParallel(...) – множення окремого діапазону рядків матриці A на B(використовується для багатопоточності).
- parallelMultiply(...) – створює k потоків, розподіляє між ними рядки матриці A та запускає обчислення:
  - Ділить матрицю A на блоки рядків, кожен з яких обробляє окремий потік;
  - Визначає **rowsPerThread** і враховує залишок рядків, якщо n%k != 0;
  - Створює k потоків і передає їм функцію **multiplyMatricesParallel** з параметрами початкового та кінцевого рядка;
  - Чекає завершення всіх потоків за допомогою **join()**.
  - Така реалізація дозволяє розподілити навантаження між кількома ядрами процесора та прискорити обчислення.

```
void parallelMultiply(int n, int k, const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int m, int p){
    vector<thread> threads;
    int rowsPerThread = n / k;
    int remainder = n % k;
    int currentRow = 0;

    for(int i = 0; i < k; ++i){
        int startRow = currentRow;
        int endRow = startRow + rowsPerThread + (i < remainder ? 1 : 0);
        currentRow = endRow;
        threads.emplace_back(multiplyMatricesParallel, ref(A), ref(B), ref(C), startRow, endRow, m, p);
    }

    for(auto& t : thread & threads) t.join();
}
```

3. **Основна частина (main):**
    - Зчитуються розміри матриць та кількість потоків k.
    - Генерується вміст матриць A та B.
    - Виконується множення:
        - спочатку послідовне, заміряється час;
        - потім паралельне, теж заміряється час.

```
auto start : time_point<...> = chrono::high_resolution_clock::now();
multiplyMatrices(A, B, C, n, m, p);
auto end : time_point<...> = chrono::high_resolution_clock::now();
auto seqTime : double = chrono::duration<double, milli>(end - start).count();

start = chrono::high_resolution_clock::now();
parallelMultiply(n, k, A, B, C, m, p);
end = chrono::high_resolution_clock::now();
auto parTime : double = chrono::duration<double, milli>(end - start).count();
```

    - Обчислюється:
        - **Speedup** = Тпосл / Тпар
        - **Efficiency** = Speedup / k

**Ключові моменти реалізації:**

- Використано **бібліотеку <thread>** для створення потоків.
- Кількість рядків рівномірно ділиться між потоками; якщо залишаються «зайві» рядки (n % k != 0), вони додаються до перших потоків.
- Для замірів часу використовується **chrono::high_resolution_clock**.

- Є **перевірка введення** (щоб не було від’ємних або некоректних значень).

**Результат:**

```
Enter a number of rows for A: 500
Enter a number of columns for A (and B): 500
Enter a number of columns for B: 500
Enter a number of threads: 2

Sequential time: 1308.84 ms
Parallel time: 669.138 ms
Speedup: 1.956
Efficiency: 0.978002
```

```
Enter a number of rows for A: 500
Enter a number of columns for A (and B): 500
Enter a number of columns for B: 500
Enter a number of threads: 4

Sequential time: 1303.32 ms
Parallel time: 340.402 ms
Speedup: 3.82876
Efficiency: 0.95719
```

```
Enter a number of rows for A: 1000
Enter a number of columns for A (and B): 1000
Enter a number of columns for B: 1000
Enter a number of threads: 8

Sequential time: 10581.6 ms
Parallel time: 1762.8 ms
Speedup: 6.00269
Efficiency: 0.750336
```

```
Enter a number of rows for A: 1000
Enter a number of columns for A (and B): 800
Enter a number of columns for B: 600
Enter a number of threads: 6

Sequential time: 5129.12 ms
Parallel time: 906.838 ms
Speedup: 5.65605
Efficiency: 0.942675
```

**Висновок:** Розроблена програма демонструє різницю між послідовним і паралельним виконанням множення матриць.

- При достатньо великих розмірах матриць використання багатопоточності значно скорочує час обчислень.

- Паралельне виконання дозволяє досягти прискорення у кілька разів.
- Ефективність залежить від кількості потоків: із зростанням k не завжди спостерігається лінійне прискорення через накладні витрати на керування потоками.
- Програма правильно працює навіть у випадках, коли кількість рядків матриці не кратна кількості потоків: перші потоки обробляють по одному рядку більше, а це дозволяє уникнути пропуску рядків та забезпечити коректний результат множення.
- Для малих розмірів матриць паралельність може навіть збільшувати час виконання, оскільки витрати на створення потоків більші за виграш від розпаралелювання.

Таким чином, паралельне множення матриць є ефективним при великих обчислювальних задачах, але потребує оптимального вибору кількості потоків, а також врахування розподілу рядків при некратності n на k.