ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Паралельні та розподілені обчислення
ЛАБОРАТОРНА РОБОТА №1

Тема: «Розпаралелення додавання/віднімання матриць».

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

2025

Тема: Розпаралелення додавання/віднімання матриць.

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

Опис програми:

Ініціалізація матриць:

Матриці A та B (розміром n × m) заповнюються випадковими числами у діапазоні [0; 10000].

Результат додавання або віднімання зберігається у матриці C (розміром n × m).

Функції програми:

1. **fillMatrix(...)** – заповнення матриці випадковими числами:
    - Створює генератор випадкових чисел **mt19937** і рівномірний розподіл **uniform_int_distribution**;
    - Проходить по всіх елементах матриці та присвоює випадкове значення.

```
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);
        }
    }
}
```

2. **inputInt(...)** – коректне введення числових значень з перевіркою на додатність.
3. **inputMatrixParam(...)** – зчитування параметрів матриці (n, m) та кількості потоків k.
4. **addMatrices(...) та subtractMatrices(...)** – послідовне виконання додавання та віднімання матриць.
5. **addMatricesParallel(...) та subtractMatricesParallel(...)** – паралельні версії, які обробляють певний діапазон рядків матриці.
6. **parallelOperation(...)** – створює k потоків, розподіляє між ними рядки матриці і запускає обчислення:
    - Ділить матрицю на блоки рядків для кожного потоку;
    - Враховує залишок рядків (n % k), додаючи по одному рядку до перших потоків;
    - Використовує **join()**, щоб дочекатися завершення всіх потоків.

```
template<typename Func, typename ... Args>
void parallelOperation(int n, int k, Func f, Args&&... args){
    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(f, args..., startRow, endRow);
    }

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

7. **measureTime(...)** – заміряє час виконання послідовного та паралельного обчислення:
    - Для паралельного виконання може робити кілька повторів (**repeatsInternal**), щоб точніше оцінити середній час;
    - Обчислює **Speedup** = Тпосл / Тпар та **Efficiency** = Speedup / k.

```
template<typename Func1, typename Func2>
void measureTime(const vector<vector<int>>& A, const vector<vector<int>>& B, vector<vector<int>>& C, int n, int m, int k, Func1 seqFunc, Func
    const int repeatsInternal = 1000;

    auto start : time_point<...> = chrono::high_resolution_clock::now();
    seqFunc(A, B, C, n, m);
    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();
    for(int r = 0; r < repeatsInternal; r++)
        parallelOperation(n, k, parFunc, ref(A), ref(B), ref(C), m);
    end = chrono::high_resolution_clock::now();
    auto parTime : double = chrono::duration<double, milli>(end - start).count();

    cout << "\nSequential time: " << seqTime << " ms\n";
    cout << "Parallel time: " << parTime << " ms\n";
    cout << "Speedup: " << double(seqTime) / parTime << endl;
    cout << "Efficiency: " << (double(seqTime) / parTime) / k << endl;


}
```

Основна частина (main):

- Зчитуються розміри матриць та кількість потоків.
- Генерується вміст матриць A і B за допомогою **fillMatrix**.
- Користувач обирає операцію: додавання або віднімання.
- Виконується послідовне та паралельне обчислення, заміряється час виконання.
- Виводяться результати: час, speedup і ефективність.

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.**
- Перевірка введення гарантує, що користувач не введе від’ємні або некоректні значення.
- Шаблонна реалізація дозволяє одним механізмом обробляти як додавання, так і віднімання.

Результат:

```
Enter a number of rows: 5000
Enter a number of columns: 5000
Enter a number of threads: 4


1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 1

Sequential time: 277.343 ms
Parallel time: 122.138 ms
Speedup: 2.27073
Efficiency: 0.567683
```

```
1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 2

Sequential time: 269.6 ms
Parallel time: 120.819 ms
Speedup: 2.23144
Efficiency: 0.55786
```

```
Enter a number of rows: 10000
Enter a number of columns: 10000
Enter a number of threads: 8


1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 1

Sequential time: 1108.75 ms
Parallel time: 238.135 ms
Speedup: 4.65595
Efficiency: 0.581993
```

```
1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 2

Sequential time: 1064.72 ms
Parallel time: 237.609 ms
Speedup: 4.481
Efficiency: 0.560125
```

```
Enter a number of rows: 5000
Enter a number of columns: 4000
Enter a number of threads: 7


1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 1

Sequential time: 232.586 ms
Parallel time: 37.7145 ms
Speedup: 6.16702
Efficiency: 0.881003
```

```
1. Addition (A + B)
2. Subtraction (A - B)
0. Exit
Choose option: 2

Sequential time: 213.095 ms
Parallel time: 37.6659 ms
Speedup: 5.6575
Efficiency: 0.808214
```

Висновок: Розроблена програма демонструє різницю між послідовним і паралельним виконанням операцій над матрицями.

- При достатньо великих розмірах матриць використання багатопоточності значно скорочує час обчислень.
- Speedup пропорційний кількості потоків, а ефективність трохи знижується через накладні витрати на управління потоками.
- Програма коректно працює навіть у випадках, коли кількість рядків не кратна кількості потоків: перші потоки отримують по одному рядку більше, що дозволяє уникнути пропуску рядків.
- Для малих матриць паралельність може бути неефективною, бо витрати на створення потоків перевищують виграш від розпаралелювання.
- Використання шаблонної реалізації робить програму універсальною, дозволяючи виконувати як додавання, так і віднімання матриць з ефективним розподілом навантаження між потоками.