Лабораторна робота №6

Дисципліна: Розподілені обчислення **Тема:** Паралельні алгоритми розв'язання диференціальних рівнянь в частинних похідних

1. Мета роботи

Розробка паралельної програми для чисельного розв'язання задачі Діріхле для рівняння Пуассона методом Гаусса-Зейделя з використанням технології MPI. Проведення обчислювальних експериментів та аналіз ефективності розпаралелювання.

2. Постановка задачі

Необхідно знайти функцію $u(x, y)$, що задовольняє рівняння Лапласа/Пуассона в одиничному квадраті. Використовується скінченно-різницевий метод (метод сіток). Значення у вузлі сітки на кожній ітерації перераховується за формулою:

$$u_{i,j}^k = 0.25 \cdot (u_{i-1,j}^k + u_{i+1,j}^{k-1} + u_{i,j-1}^k + u_{i,j+1}^{k-1} - h^2 f_{i,j})$$

3. Опис алгоритму розпаралелювання

Декомпозиція даних

Використано одновимірну декомпозицію (горизонтальний стрічковий поділ). Матриця розміром $N \times N$ розрізається на горизонтальні смуги, які розподіляються між процесами.

Тіньові грані (Halo Rows)

Оскільки для обчислення значення в точці (i, j) потрібні значення сусідів, на межах смуг виникає необхідність обміну даними.

- Кожен процес зберігає свою частину рядків + 2 додаткових "тіньових" рядки (верхній та нижній), які є копіями граничних рядків сусідніх процесів.
- Перед кожною ітерацією відбувається обмін цими рядками.

4. Опис програмної реалізації

У роботі розроблено дві програми мовою C++:

1. **Serial.cpp** — послідовна версія для отримання еталонного часу.
2. **Parallel.cpp** — паралельна версія з використанням MPI.

Ключові функції паралельної реалізації (Parallel.cpp):

- `DataDistribution` : Використовує `MPI_Scatter` для розсилки внутрішніх рядків матриці процесам. Окремо обробляється передача граничних умов (перший та останній рядки глобальної матриці).
- `ExchangeData` : Реалізує обмін тіньовими рядками. Використано функцію `MPI_Sendrecv` , що дозволяє уникнути взаємного блокування (deadlock) та об'єднує операції відправки та

отримання:

```
// Обмін з наступним та попереднім процесом
MPI_Sendrecv(..., NextProcNum, ..., PrevProcNum, ...);
MPI_Sendrecv(..., PrevProcNum, ..., NextProcNum, ...);
```

- `ParallelResultCalculation` : Основний цикл ітерацій. На кожному кроці викликається обмін даними, локальний перерахунок та перевірка умови зупинки.
- `MPI_Allreduce` : Використовується для збору максимальної похибки (`dmax`) з усіх процесів, щоб прийняти рішення про продовження або зупинку ітерацій глобально.

5. Результати експериментів

На основі отриманих даних (файл `screen.png`) побудовано таблицю залежності часу виконання та прискорення від розміру матриці та кількості процесорів.

Розмір матриці	Послідовний час (сек)	2 процесори (сек)	Прискорення (2)	4 процесори (сек)	Прискорення (4)	8 процесорів (сек)	Прискорення (8)
1000	0.002350	0.006209	0.37	0.006095	0.38	0.007097	0.33
2000	0.011142	0.026040	0.42	0.026910	0.41	0.025949	0.42
3000	0.023269	0.056417	0.41	0.060603	0.38	0.073260	0.31
...	...	...	...	...	...	...	...
8000	0.155384	0.522003	0.29	0.452549	0.34	0.501378	0.30
10000	0.244174	0.779762	0.31	0.708627	0.34	0.763326	0.31

Аналіз графіків та даних

1. **Малі розмірності (N < 1000):** Спостерігається сповільнення (Speedup < 1). Це пояснюється тим, що час на комунікацію між процесами (ініціалізація MPI, пересилання даних) значно перевищує час корисних обчислень.
2. **Накладні витрати:** При збільшенні кількості процесорів до 8 для задач малого та середнього розміру час виконання часто зростає порівняно з 4 процесорами. Це свідчить про те, що смуга даних стає занадто вузькою, і накладні витрати на синхронізацію домінують.
3. **Великі розмірності (N = 10000):** Навіть на великих розмірах ми не бачимо лінійного прискорення у цьому конкретному запуску. Прискорення коливається в межах 0.3 - 0.4.

Причини низького прискорення у даному експерименті:

- Ймовірно, заміри проводилися на одній фізичній машині з обмеженою кількістю ядер, або у віртуальному середовищі, де перемикання контексту процесів MPI займає багато часу.
- Алгоритм Гаусса-Зейделя має сильну залежність від даних (ітераційний процес), що вимагає синхронізації на *кожному* кроці, що є "вузьким місцем" для великої кількості процесів.

6. Висновок

У ході лабораторної роботи було реалізовано паралельний алгоритм розв'язання задачі Діріхле методом Гаусса-Зейделя.

1. Програма коректно виконує декомпозицію даних та збирає фінальний результат (перевірено на малих розмірностях матриці).
2. Використання `MPI_Sendrecv` забезпечило коректний обмін граничними умовами без виникнення deadlocks.
3. Експерименти показали, що для даного класу задач (тісний ітераційний зв'язок) ефективність розпаралелювання сильно залежить від співвідношення "обчислення/комунікації". Для отримання суттєвого прискорення необхідні дуже великі розмірності матриці, щоб завантажити процесори корисною роботою і нівелювати витрати на пересилку даних мережею.