МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА

Лабораторна робота №9-10

“MPI кластер”

Виконав:
студент групи ПМі-31
Дудинець Олександр

Львів 2024

Мета роботи

Метою лабораторної роботи було розробити програму для обчислення добутку двох матриць з використанням послідовного та паралельного алгоритмів на кластері. Завдання включало також обчислення часу виконання кожного алгоритму, прискорення та ефективності паралельного алгоритму для різної кількості потоків та розмірів матриць. Результати, отримані на кластері, порівнювалися з результатами, отриманими на локальному комп'ютері у попередній лабораторній роботі.

Опис ключових частин коду

1. **Генерація матриць** (generateMatrix): Функція створює двовимірну динамічну матрицю розміром n x m і заповнює її випадковими числами. Ця матриця використовується як вхідні дані для алгоритмів множення.
2. **Послідовне множення матриць** (multiplyMatricesSequential): Функція виконує послідовне множення двох матриць, обчислюючи значення кожного елемента результуючої матриці через обхід по рядках і стовпцях. Цей підхід служить базою для порівняння з паралельним алгоритмом.
3. **Паралельне множення матриць** (multiplyMatricesParallel): Функція multiplyMatricesParallel реалізує паралельний алгоритм множення матриць шляхом розподілу обчислень між кількома процесами з використанням бібліотеки MPI (Message Passing Interface). Основний принцип роботи цієї функції полягає в тому, що матриця розподіляється по рядках, які обробляються незалежно один від одного.
    a. Розподіл рядків між процесами:
        i. Кожен процес отримує певну кількість рядків першої матриці для обробки. Якщо кількість рядків не кратна числу процесів, то залишкові рядки розподіляються між першими процесами, щоб усі процеси мали приблизно однакове навантаження. Це дозволяє рівномірно розподілити обчислювальні ресурси, зменшуючи час простою.
    b. Обчислення добутку для частини рядків:

i. Кожен процес обчислює добуток своїх рядків першої матриці з усіма стовпцями другої матриці. Для кожного елемента результуючої матриці обчислюється сума добутків відповідних елементів рядка з елементами стовпця.
ii. Наприклад, процес обчислює свій частковий результат, використовуючи алгоритм, аналогічний послідовному множенню, але лише для заданої підмножини рядків, що дозволяє зменшити час обчислень у кожному окремому процесі.

c. Асинхронність та оптимізація ресурсів:
i. Завдяки паралельному розподілу роботи процеси можуть виконувати обчислення одночасно, не чекаючи один на одного. Це значно підвищує швидкість виконання для великих матриць, адже замість обробки всіх рядків по черзі, кожен процес працює незалежно над своєю частиною.

4. **Збирання результатів** (gatherResults): Функція gatherResults відповідає за об’єднання часткових результатів, які кожен процес обчислив у функції multiplyMatricesParallel, в одну фінальну матрицю на основному процесі (зазвичай процес з рангом 0). Це забезпечує отримання повного результату множення матриць після завершення паралельних обчислень. Основні етапи функції включають:
a. Передача результатів від кожного процесу основному процесу:
i. Після завершення обчислень кожен процес зберігає свої результати в локальній матриці, яка містить обчислені значення для відповідних рядків. Ці часткові результати мають бути передані основному процесу для об’єднання в фінальну результуючу матрицю.
ii. Якщо процес є основним (ранг 0), він збирає результати у свою фінальну матрицю. Кожен інший процес відправляє свої часткові результати основному процесу за допомогою команди MPI_Send.
b. Збирання та об’єднання результатів:

i. Основний процес (процес 0) отримує дані від кожного процесу за допомогою команди MPI_Recv, розміщуючи їх у відповідних рядках фінальної матриці.
ii. Для кожного процесу, який надіслав свої результати, основний процес знає, які рядки він повинен отримати, та зберігає їх у фінальній матриці на відповідних позиціях.
iii. Це забезпечує коректність результату, адже всі елементи результуючої матриці об’єднані в потрібному порядку.

c. Коректність та завершення обчислень:
i. Після того, як усі процеси надіслали свої часткові результати основному процесу, обчислення можна вважати завершеними.
ii. Основний процес виводить результуючу матрицю або передає її для подальших обчислень. Важливо, що даний підхід гарантує, що всі рядки зібрані у правильному порядку.

5. **Обчислення швидкості та ефективності**: Швидкість та ефективність паралельного алгоритму обчислюються шляхом порівняння часу послідовного та паралельного виконання для кожного експерименту.

Результати експериментів

Програма була протестована на кластері з різними параметрами (кількістю потоків і розміром матриць). Нижче наведено результати тестування:

1 потік

```
- Matrix size: 100x100 to 100x100
- Threads: 1
- Sequential time: 10.00 ms
- Parallel time: 0.00 ms
- Speedup: infx
- Efficiency: inf%
- Matrices equal: Yes
```

```
- Matrix size: 1000x1000 to 1000x1000
- Threads: 1
- Sequential time: 7480.00 ms
- Parallel time: 7570.00 ms
- Speedup: 0.99x
- Efficiency: 98.81%
- Matrices equal: Yes
```

```
- Matrix size: 2000x2000 to 2000x2000
- Threads: 1
- Sequential time: 84080.00 ms
- Parallel time: 87135.00 ms
- Speedup: 0.96x
- Efficiency: 96.49%
- Matrices equal: Yes
```

4 потоки

```
- Matrix size: 100x100 to 100x100
- Threads: 4
- Sequential time: 10.00 ms
- Parallel time: 0.00 ms
- Speedup: infx
- Efficiency: inf%
- Matrices equal: Yes
```

```
- Matrix size: 1000x1000 to 1000x1000
- Threads: 4
- Sequential time: 8320.00 ms
- Parallel time: 2910.00 ms
- Speedup: 2.86x
- Efficiency: 71.48%
- Matrices equal: Yes
```

```
- Matrix size: 2000x2000 to 2000x2000
- Threads: 4
- Sequential time: 96320.00 ms
- Parallel time: 24710.00 ms
- Speedup: 3.90x
- Efficiency: 97.45%
- Matrices equal: Yes
```

12 потоків

```
- Matrix size: 100x100 to 100x100
- Threads: 12
- Sequential time: 10.00 ms
- Parallel time: 0.00 ms
- Speedup: infx
- Efficiency: inf%
- Matrices equal: Yes
```

```
- Matrix size: 1000x1000 to 1000x1000
- Threads: 12
- Sequential time: 7930.00 ms
- Parallel time: 1230.00 ms
- Speedup: 6.45x
- Efficiency: 53.73%
- Matrices equal: Yes
```

```
- Matrix size: 2000x2000 to 2000x2000
- Threads: 12
- Sequential time: 95440.00 ms
- Parallel time: 9060.00 ms
- Speedup: 10.53x
- Efficiency: 87.79%
- Matrices equal: Yes
```

```
- Matrix size: 753x924 to 924x154
- Threads: 12
- Sequential time: 620.00 ms
- Parallel time: 80.00 ms
- Speedup: 7.75x
- Efficiency: 64.58%
- Matrices equal: Yes
```

Аналіз результатів

1. **Прискорення та ефективність:** Результати показують, що паралельний алгоритм забезпечує значне прискорення для великих матриць, особливо при використанні 12 потоків. Наприклад, для матриці розміром 2000x2000 з 12 потоками спостерігалося прискорення в 10.53 рази та ефективність 87.79%. Проте, для малих матриць (100x100) паралельний алгоритм не дає значного прискорення через невеликі витрати часу на обчислення та накладні витрати на управління потоками.
2. **Ефективність при великих матрицях:** Для великих матриць (2000x2000) спостерігається висока ефективність при 4 і 12 потоках, що свідчить про вдало розроблений алгоритм для паралельного множення матриць, який ефективно використовує ресурси кластера.
3. **Низька ефективність на малих матрицях:** У випадках, коли розмірність матриць мала (100x100), накладні витрати на створення та управління потоками перевищують виграш від паралельного обчислення, що призводить до низької ефективності.
4. **Використання малої кількості потоків**: При використанні лише одного потоку (1 процес), паралельний алгоритм фактично перетворюється на послідовний, оскільки немає розподілу обчислень між процесами. Це призводить до того, що час виконання паралельного алгоритму майже дорівнює часу виконання

послідовного. Таким чином, прискорення дорівнює приблизно 1x, а ефективність становить близько 100%. Цей випадок корисний для перевірки коректності коду, але не приносить жодного виграшу у продуктивності.

5. **Матриці з розмірами, некратними кількості потоків**: Алгоритм ефективно справляється з випадками, коли розмір матриці не кратний кількості потоків. У таких ситуаціях рядки, що залишаються після рівномірного розподілу, призначаються першим процесам, що дозволяє уникнути простою і зберегти високу ефективність. Наприклад, якщо кількість рядків не ділиться на кількість потоків, залишкові рядки розподіляються між першими N потоками, що забезпечує рівномірне навантаження та мінімізує втрати продуктивності.
6. **Співпадіння результатів:** Результати множення для послідовного та паралельного алгоритмів були однаковими для всіх випадків, що підтверджує коректність реалізації паралельного алгоритму.

Порівняння з локальними результатами

У другій лабораторній роботі мною були отримані такі результати для матриць розміром 1000x2000 та 2000x1000:

```
- Summary:
   - Matrix size: 1000x2000 to 2000x1000
   - Threads: 8
   - Sequential time: 5079ms
   - Parallel time: 965ms
   - Speedup: 5.26321x
   - Efficiency: 65% (took 965ms vs 634ms ideal)
   - Matrices equal: Yes
```

Результати на кластері можна зіставити за допомогою розмірів 1000x1000 та 2000x2000, оскільки в тестах було обрано дещо інші розміри, але найближчі відповідники дають уявлення про продуктивність.

- **Прискорення та ефективність:**
    - На локальному комп’ютері з 8 потоками для матриць 1000x2000 та 2000x1000 було досягнуто прискорення 5.26x та ефективність 65%.
    - На кластері для подібних розмірів та з 12 потоками для розміру 2000x2000 було досягнуто прискорення 10.53x з ефективністю 87.79%. Це свідчить про те, що кластер забезпечує кращу продуктивність завдяки більшій кількості доступних потоків та більш оптимізованій архітектурі.
- **Час виконання:**
    - Для матриці 1000x1000 на кластері з 12 потоками час паралельного виконання становив 1230 мс, що порівняно з 965 мс на локальному комп’ютері для матриці 1000x2000 є більше.
- **Ефективність на малих матрицях:**
    - На малих матрицях (наприклад, 100x100) спостерігається значно нижча ефективність як на локальному комп’ютері, так і на кластері, оскільки накладні витрати на розподіл задач та синхронізацію потоків перевищують виграш від паралелізму.

Висновок

Під час виконання цієї лабораторної роботи я розширив свої знання в галузі паралельних обчислень та навчився застосовувати алгоритми для обчислення добутку двох матриць на кластері. Проведене тестування показало, що паралельний алгоритм ефективний для великих матриць, але для малих розмірів його використання не є доцільним через накладні витрати на управління потоками. Порівняння з результатами на локальному комп’ютері підтвердило перевагу кластерних обчислень для задач, що вимагають інтенсивних обчислень.