Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського» Факультет інформатики та обчислювальної техніки

*Кафедра інформатики та програмної інженерії*

Звіт до лабораторної роботи №6

з курсу

“Технології паралельних обчислень”

# **Виконав:**

Студент гр. ІТ-02

Терешкович М.О.

Київ – 2023

**Тема:** Розробка паралельного алгоритму множення матриць з використанням МРІ-методів обміну повідомленнями «один-до-одного» та дослідження його ефективності

A picture containing text, font, screenshot

Description automatically generated**Завдання:**

**Хід роботи:**

**Почнемо з допоміжних методів:**

**A picture containing text, screenshot, font

Description automatically generated**

Даний метод відповідає за заповнення матриці рандомними числами з заданого діапазона.

Далі йде метод Start(), який запускає експеримент та викликається із Main(), який також виконує обчислення і вимірює час, необхідний для кожного обчислення, використовуючи MPI (Message Passing Interface - інтерфейс передачі повідомлень) для паралельної обробки.

1. Метод Start приймає два параметри: rank і size, які представляють ранг і розмір MPI-процесу.
2. Якщо ранг дорівнює 0, він виводить повідомлення із зазначенням кількості процесорів.
3. Потім він ініціалізує цілочисельну змінну experimentAmount зі значенням 2. Ця змінна представляє кількість разів, яку буде виконано обчислення.
4. Створює список ArrayList з іменем massive і до нього додаються три цілих числа (500, 1000 і 2000). Ці числа представляють розміри матриць, для яких будуть виконуватися обчислення.
5. Далі йде цикл, який перебирає елементи масиву massive. Усередині циклу генеруються дві випадкові матриці (A і B) за допомогою методу RandomMatrix. Розмір цих матриць визначається поточним елементом n у циклі.
6. Створюється новий масив типу double C розміром n\*n. У цьому масиві буде зберігатись результат обчислення.
7. Запускається секундомір (Stopwatch s) для вимірювання часу, що пройшов за час обчислення. Він входить в інший цикл, який обчислює експериментальний час.
8. Усередині цього циклу він викликає обидва методи Blocking і NonBlocking, передаючи ранг, розмір, n, A, B і C як параметри.
9. Після завершення внутрішнього циклу викликається MPI.COMM\_WORLD.Barrier(). Це точка синхронізації, якої повинні досягти всі MPI-процеси перед тим, як продовжити роботу.
10. Якщо ранг дорівнює 0, то обчислюється середній час, необхідний для обчислення, шляхом ділення часу, що пройшов з секундоміра, на experimentAmount. Потім він виводить цей середній час разом з розміром поточної матриці «n».

A picture containing text, screenshot, software, multimedia software

Description automatically generated

Як бачимо основні параметри експеременту будуть матриці різної розірності (500, 1000, 2000).

A picture containing text, screenshot, font

Description automatically generated

Це головний метод є точкою входу в програму. Він викликає MPI.Init(args) для ініціалізації середовища MPI і передачі будь-яких аргументів командного рядка. Він використовує MPI.COMM\_WORLD.Rank() для отримання рангу поточного MPI-процесу і присвоює його змінній rank. Також використовує MPI.COMM\_WORLD.Size() для отримання загальної кількості MPI-процесів і присвоює її змінній size. Далі викликає метод Start, який передає змінні rank та size як параметри. Після повернення методу Start викликається метод MPI.Finalize() для завершення роботи середовища MPI та очищення ресурсів.

Таким чином, головний метод ініціалізує середовище MPI, отримує ранг і розмір MPI-процесу, викликає метод Start для виконання обчислень, а потім завершує роботу MPI-середовища. Така структура дозволяє декільком MPI-процесам виконувати метод Start одночасно у паралельному середовищі.

Тепер перейдемо до основних методів блокуючого та не блокуючого обміну повідомленнями.

A picture containing text, screenshot, software

Description automatically generated

Цей метод відповідає за множення матриці з використанням блокування у паралельному середовищі MPI. Розберемо як працює код:

1. Метод Blocking отримує декілька параметрів: rank (ранг MPI-процесу), size (загальна кількість MPI-процесів), n (розмір матриць) та масиви A, B і C, що представляють вхідні та вихідні матриці.
2. Він обчислює кількість рядків на процес шляхом ділення n на розмір і присвоює результат змінній rowsPerProcess.
3. Якщо ранг дорівнює 0 «Майстер процес» , він надсилає шматки матриці A іншим процесам. Для надсилання даних використовується метод MPI.COMM\_WORLD.Send. Відправка виконується у циклі, де кожен процес (за винятком Майстер процесу) отримує частину матриці A.
4. Якщо ранг не дорівнює 0, то він отримує частину матриці A, надіслану кореневим процесом за допомогою методу MPI.COMM\_WORLD.Recv.
5. Після розподілу матриці A виконується цикл, який ітераційно перебирає кількість процесів за розміром. На кожній ітерації процес з початковим рангом надсилає всю свою матрицю B наступному процесу за допомогою методу MPI.COMM\_WORLD.Send. Наступний процес отримує матрицю B від поточного процесу за допомогою MPI.COMM\_WORLD.Recv.
6. Після передачі матриці B відбувається бар'єрна точка синхронізації за допомогою MPI.COMM\_WORLD.Barrier(). Це гарантує, що всі процеси досягли цієї точки, перш ніж продовжити роботу.
7. Множення матриці виконується шляхом ітерації над рядками, призначеними поточному процесу (на основі рангу), і обчислення відповідних елементів матриці C. Кожен процес виконує свою частину множення.
8. Після завершення множення матриці, якщо ранг дорівнює рангу Майстер процесора «0» то, він отримує частини матриці C від інших процесів і збирає результати. Якщо ранг не дорівнює «0», він надсилає свою частину матриці C кореневому процесу.

В результаті ми маємо метод, який виконує множення матриць за допомогою блокування у паралельному середовищі MPI. Він розподіляє вхідні матриці A і B між різними процесами, виконує множення і збирає результати. Метод використовує примітиви зв'язку MPI (Send та Recv) та точки синхронізації (Barrier) для координації паралельних обчислень.

A picture containing text, screenshot

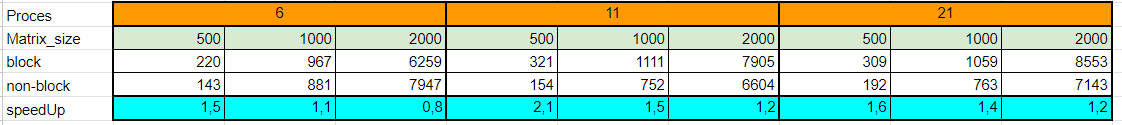
Description automatically generatedДалі йде метод з назвою NonBlocking, який виконує множення матриць з використанням неблокуючого підходу у паралельному середовищі MPI. Розберемо код покроково:

1. Метод NonBlocking отримує всі ті ж парматри що і блокуючий метод до цього.
2. Далі він обчислює кількість рядків на процес шляхом ділення n на розмір і присвоює результат змінній rowsPerProcess.
3. Оголошує масив об'єктів типу Request з іменем reqs довжиною 5. Цей масив буде використовуватися для зберігання неблокуючих комунікаційних запитів.
4. Якщо ранг дорівнює «0», він надсилає фрагменти матриці A іншим процесам, використовуючи неблокуючий зв'язок. Він використовує метод MPI.COMM\_WORLD.Isend для ініціювання операції неблокуючого надсилання і присвоює об'єкт запиту reqs[0].
5. Якщо ранг не дорівнює «0», він отримує частину матриці A, надіслану кореневим процесом за допомогою неблокуючого зв'язку. Він використовує метод MPI.COMM\_WORLD.Irecv для ініціювання неблокуючої операції отримання і чекає на завершення операції за допомогою методу Wait на об'єкті запиту.
6. Після розподілу матриці A відбувається цикл, який ітераційно перебирає задану кількість процесів. На кожній ітерації процес з початковим рангом надсилає всю свою матрицю B наступному процесу, використовуючи неблокуючий зв'язок Isend. Наступний процес отримує матрицю B від поточного процесу за допомогою неблокуючого зв'язку Irecv і чекає на завершення операції отримання за допомогою методу Wait.
7. Після передачі матриці B у цьому коді відсутня бар'єрна точка синхронізації(на відміну від попереднього метода)
8. Множення матриці виконується шляхом ітерації над рядками, призначеними поточному процесу (на основі рангу) та обчислення відповідних елементів матриці C. Кожен процес виконує свою частину множення.
9. Після завершення множення матриці, якщо ранг дорівнює «0», він отримує частини матриці C від інших процесів і збирає результати, використовуючи блокування зв'язку Recv. Якщо ранг не дорівнює «0», він надсилає свою частину матриці C кореневому процесу, використовуючи неблокуючий зв'язок Isend.

Як результат, метод NonBlocking виконує множення матриць з використанням неблокуючого підходу у паралельному середовищі MPI. Він розподіляє вхідні матриці A і B між різними процесами, виконує множення і збирає результати. Метод використовує неблокуючі комунікаційні примітиви MPI (Isend та Irecv) для покращення продуктивності за рахунок перекриття комунікації та обчислень.

Приклади роботи програми:  
A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

Рисунок 1 – Тестування швидкості множення матриць з блокуючими та неблокуючими методами обміну повідомленнями

Всі заміри були виконані в мілісекундах. Як бачимо не блокуючий метод виявився швидшим, так як йому не потрібно очікувати на інші «повідомлення». Найбільше пришвидшення було отримане при розмірі матриці 500\*500 та 11 процесорами, а саме 2,1.

**Висновок:**

У ході виконання лабораторної роботи ми ознайомились з методами блокуючого та неблокуючого обміну повідомленнями типу point-to-point за допомогою стандарту MPI. Було реалізовано алгоритми паралельного множення матриць з використанням розподілених обчислень в MPI для обох методів обміну повідомленнями.

Проведено дослідження ефективності розподіленого обчислення алгоритму множення матриць при збільшенні розміру матриць та при збільшенні кількості процесів, на яких здійснюється запуск програми. В результаті порівняння ефективності, можна зробити висновок, що

* Використання неблокуючих методів обміну повідомленнями забезпечує кращу швидкість роботи програми;
* Швидкодія програми залежить від кількості «процесорів/вузлів» та їхньої кратності розміру матриці.
* Кількість вузлів треба підбирати досить прискіпливо, оскільки занадто мала чи занадто велика кількість, може призвести до зменшення швидкості роботи програми;