Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Комп’ютерний практикум №2**

з дисципліни «Технології паралельних обчислень»

Тема: «Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності»

|  |  |
| --- | --- |
| **Виконав:**  студент групи ІТ-04  Коновальчук Андрій Володимирович  Дата здачі \_\_\_\_\_\_\_\_  Захищено з балом \_\_\_\_\_\_\_ | **Перевірила**:  асистент кафедри ІПІ  Дифучина Олександра Юріївна |

Київ 2023

**Хід роботи:**

***Завдання 1:*** Реалізуйте стрічковий алгоритм множення матриць. Результат множення записуйте в об’єкт класу Result.

*Опис пункту та алгоритму:*

Основна ідея алгоритму полягає в тому, що матриці розбиваються на рядки та стовпці, і обчислення результатів проводяться поелементно вздовж цих рядків та стовпців.

Принцип роботи стрічкового алгоритму полягає в тому, що матриці розбиваються на стрічки, а потім виконується наступний алгоритм:

1. Вибирається одна стрічка з першої матриці та всі стрічки з другої матриці.
2. Кожна пара стрічок обчислюється як добуток їх елементів та сумується.
3. Отримана сума стає одним з елементів результатної матриці.
4. Процес повторюється для кожної стрічки першої матриці та кожної стрічки другої матриці.
5. Отримані результати збираються в кінцеву матрицю.

Принцип роботи паралельного стрічкового алгоритму полягає в тому, що розбиття матриць на стрічки проводиться паралельно на кількох потоках. Кожен потік отримує свою стрічку та обчислює результати добутків своєї частини зі стрічками іншої матриці

*Фрагмент коду алгоритму:*

Алгоритм складається з двох класів: один містить метод що повертає об’єкт Result, інший розширює клас Thread. В методі першого класу ми ділимо матрицю на “стрічки”, а потім “стрічки” та їх положення разом з результатною матрицею та другою матрицею відправляємо як параметри в конструктор класу StripedThread, де відбуваються подальше множення. Об’єкти класу StripedThread додаємо в список потоків та join-имо.

Інша версія цього методу замість списку потоків використовує інтерфейс ExecutorService де ми створюємо пул потоків і використовуємо метод execute.

В середині методу multiplyMatrix також фіксуємо час роботи алгоритму.



Рисунок 1 – Стрічковий алгоритм. Метод множення матриць

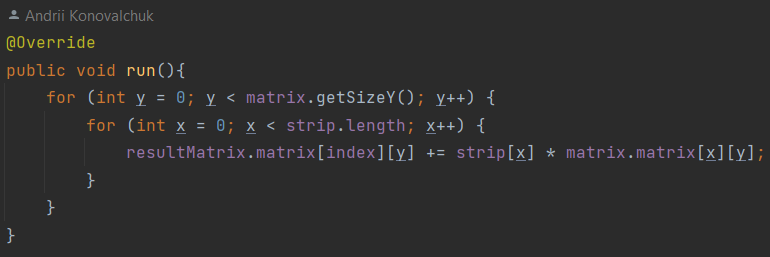


Рисунок 2 – Стрічковий алгоритм. Метод run класу що розширює клас Thread

*Спостереження, результати роботи алгоритму, висновки:*

Запустимо алгоритм з матрицею розмірами 1000 на 1000 з випадково згенерованими числами та порівняємо швидкість роботи зі звичайним множенням матриць. В даному випадку бачимо збільшення швидкості роботи алгоритму на 75%.

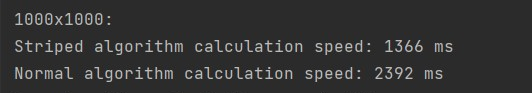


Рисунок 3 – Швидкість множення стрічковим алгоритмом

***Завдання 2:*** Реалізуйте алгоритм Фокса множення матриць.

*Опис пункту та алгоритму:*

Алгоритм Фокса базується на ідеї поділу вхідних матриць на підматриці та виконання множення цих підматриць паралельним способом. Алгоритм розбиває також матрицю на сітку потоків, причому кожен потік відповідає за обчислення підматриці результуючої матриці.

Алгоритм роботи наступний:

1. Вхідні дані: дві матриці A і B розміром n x n.
2. Розбиваємо матриці на квадратні підматриці розміром n/p x n/p, де p – кількість потоків у кожному вимірі сітки потоків.
3. Розподіляємо підматриці A і B між процесорами в сітці процесорів.
4. Виконуємо p ітерації наступних кроків:
5. Кожен процесор множить свої підматриці A і B і накопичує результат у підматриці результуючої матриці C.
6. Зсуваємо підматриці A і B циклічно вліво і вгору відповідно, щоб кожен процесор отримував нову підматрицю A і B для наступної ітерації.
7. Збираємо підматриці матриці C із потоків і об’єднуємо їх, щоб сформувати результуючу матрицю.

*Фрагмент коду алгоритму:*

Спершу треба підібрати таку кількість потоків, котра розділить сітку потоків на рівну кількість елементів. Для цього знаходимо найближче до доступних процесів кратне число до довжини сторони матриці (кількість елементів по Х або У). “Величиною кроку” буде частка розміру сторони і кількості потоків. Створюємо пул потоків та матриці, котрі будуть “картами кроків” для поділу заданих матриць на блоки. Обходимо ці карти-матриці та вирізаємо блоки, враховуючи циклічні та відповідні зміщення по осям Х та У. Створюємо об’єкт класу що розширює клас Thread, конструктор якого приймає блоки (підматриці), індекси у вигляді елементів матриці кроків та результуючу матрицю. В методі run виконується перемноження елементів блоків і запис у результуючу матрицю.



Рисунок 4 – Алгоритм Фокса. Метод множення матриць

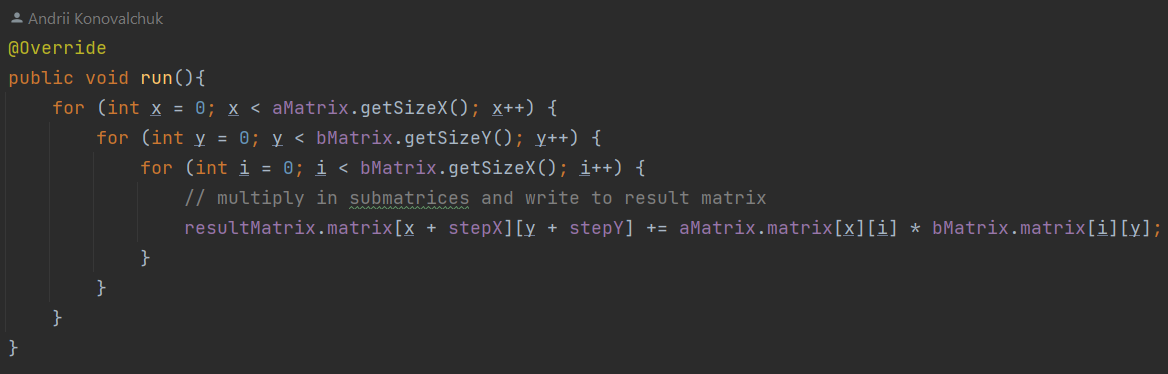


Рисунок 5 – Алгоритм Фокса. Метод run класу що розширює клас Thread

*Спостереження, результати роботи алгоритму, висновки:*

Запустимо алгоритм з матрицею розмірами 1000 на 1000 з випадково згенерованими числами та порівняємо швидкість роботи зі звичайним множенням матриць. В даному випадку бачимо значне збільшення швидкості роботи алгоритму на 440%.

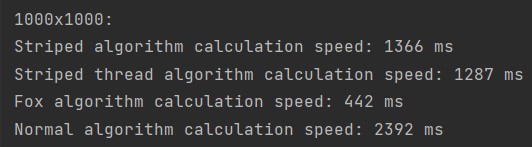


Рисунок 6 – Швидкість множення алгоритмом Фокса

***Завдання 3:*** Виконайте експерименти, варіюючи розмірність матриць, які перемножуються, для обох алгоритмів, та реєструючи час виконання алгоритму. Порівняйте результати дослідження ефективності обох алгоритмів

*Опис пункту:*

Для виконання експерименту зробимо сет розмірів матриць: 10, 100, 250, 500, 1000, 1500, 2000. Та запустимо по черзі три алгоритми: стрічковий, Фокса та звичайний алгоритм множення матриць. Виводимо час затрачиний на обчислення.

*Результати роботи:*

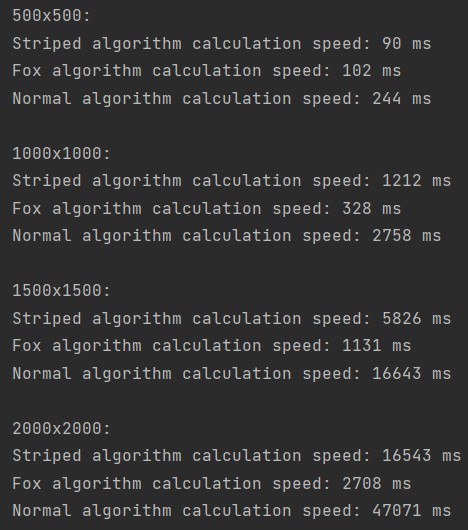
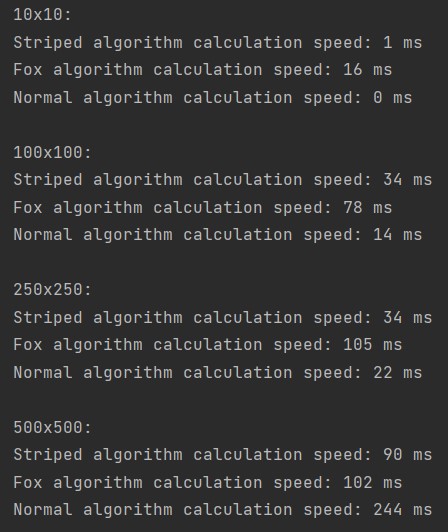


Рисунок 7 – Швидкість множення матриць різних величин стрічковим алгоритмом, алгоритмом Фокса та звичайним множенням матриць

*Спостереження та висновки:*

Глянувши на приблизні графіки, помічаємо різницю у швидкості обчислення алгоритмів. Відчутна різниця між ними при достатньо великих величинах матриць, і ця різниця доволі швидко зростає.

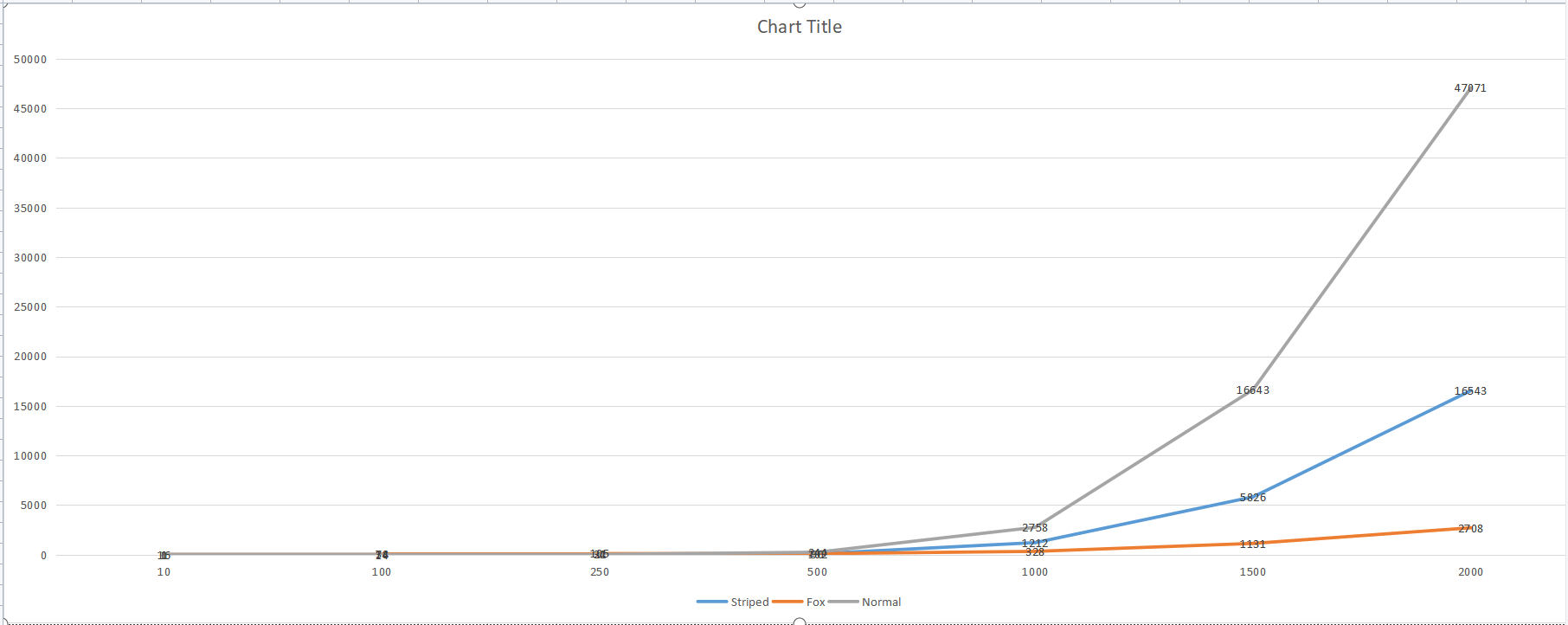


Рисунок 8 – Графік різниці швидкості при різних величинах матриці

***Завдання 4:*** Виконайте експерименти, варіюючи кількість потоків, що використовується для паралельного множення матриць, та реєструючи час виконання. Порівняйте результати дослідження ефективності обох алгоритмів.

*Опис пункту:*

Для виконання експерименту зробимо сет кількості потоків: 5, 10, 25, 50, 75, 100. Та запустимо по черзі стрічковий алгоритм та алгоритм Фокса. Матриці сталого розміру – 1000 на 1000.

Стрічковий алгоритм модифікований під це завдання. Замість списку потоків було використано інтерфейс ExecutorService де ми створюємо пул із заданим (фіксованим) значенням потоків.

*Результати роботи:*

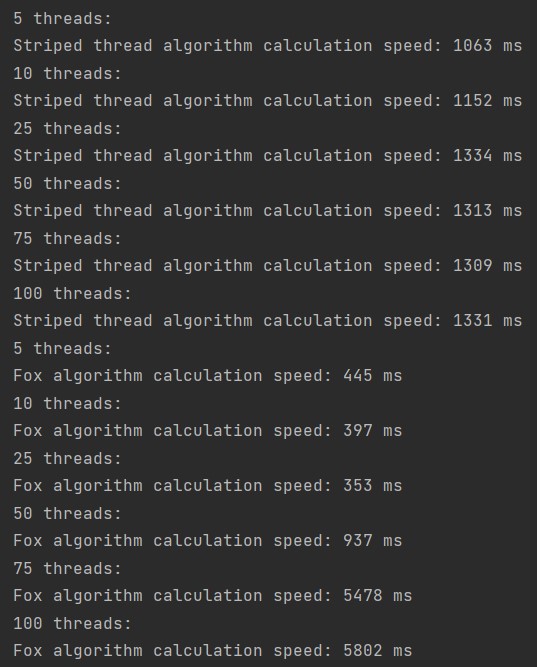


Рисунок 9 – Швидкість множення матриць різної кількості потоків стрічковим алгоритмом та алгоритмом Фокса

*Спостереження та висновки:*

Глянувши на приблизні графіки, помічаємо різницю у швидкості обчислення алгоритмів. Бачимо невелику зміну у швидкості у стрічковому алгоритмі, і вихід на “плато”. У алгоритму Фокса це спершу збільшення швидкості, а потім різкий спад швидкості.

Пояснити це можна тим, що стрічковий алгоритм ділить матриці на однакові стрічки, а алгоритм Фокса ділить матрицю на блоки в залежності від кількості потоків. Тож в один момент становиться забагато блоків і забагато операцій, і алгоритм починає обчислювати повільніше.

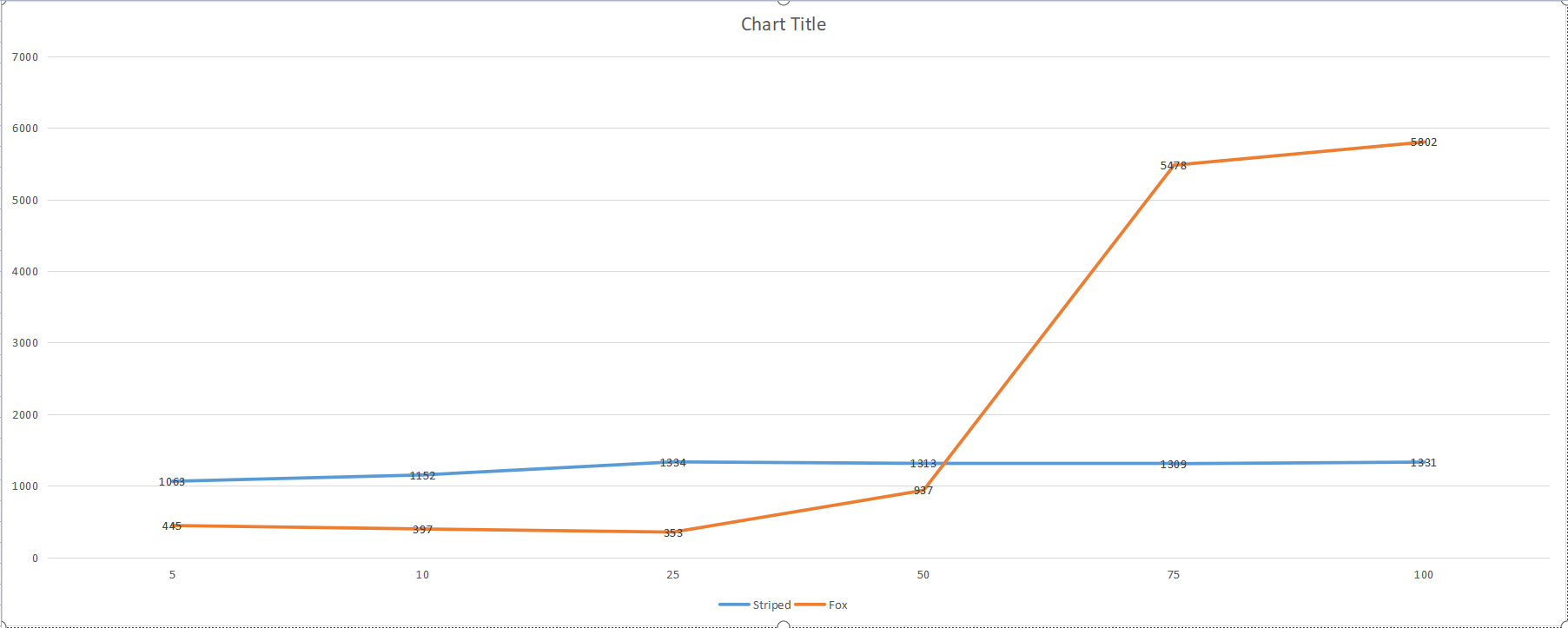


Рисунок 10 – Графік різниці швидкості при різній кількості потоків

**Висновок:**

В ході виконання практикуму було розроблено, продемонстровано, описано та пояснено стрічковий алгоритм множення матриць та алгоритм Фокса множення матриць. Було проведено есперименти над матрицями різних величин та з різною кількістю потоків. Результатами експериментів є демонстрація роботи алгоритмів, візуалізація паралелізму, а також вирішення поставлених проблем та пришвидшення роботи програм та алгоритмів шляхом паралельного програмування.

У звіті наявні скріншоти з прикладами результатів роботи коду, графіки, а також опис роботи коду та пояснення результатів. Вперше використано такі класи, інтерфейси, методи та ключові слова: ExecutorService, Runtime, execute, shutdown, awaitTermination.

**Лістинг коду у вигляді GitHub-репозиторію:**

<https://github.com/m4cy43/parallel_programming/tree/master/lab2>