НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Звіт до комп’ютерного практикуму з дисципліни “Теорія паралельних обчислень”

**Виконав**

**Студенти групи ІТ-02**

**Терешкович М.О.**

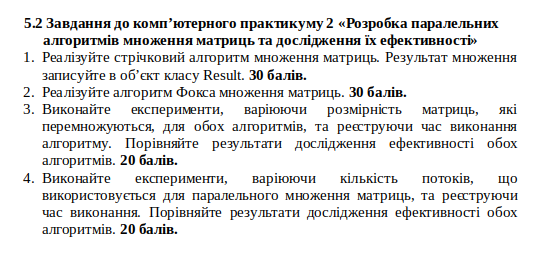
**Перевірив:**

Київ – 2022

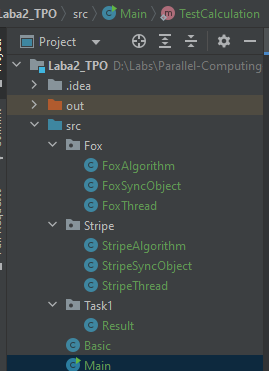
**Комп‘ютерний практикум No 2**

**Розробка паралельних алгоритмів множення матриць та дослідження їх ефективності**

ЗАВДАННЯ



**ВИКОНАННЯ**

Почнемо зі структури проекту:

Основний файл – Main в якому выдбувається запуск всіх алгоритмів, створення матриць та підрахунок часу кожного з виконань. Присутній клас Result (потрібен для виконання 1 завдання) – в який записується результати алгоритмів. Також присутні:

* базовий Basic клас – в ньому знаходиться код для базового алгоритму множення матриць.
* Пакет Fox – в якому присутній алгоритм Фокса, сам потік та синхронізований об’єкт для обміну інформацією між потоками.
* Пакет Stripe – в якому присутній Стрічковий алгоритм, сам потік та синхронізований об’єкт для обміну інформацією між потоками.

А тепер поговоримо трохи про теорію, що б на далі не зупинятися на ній.

*Фокс:*

Теорія мультипоточного алгоритму множення матриць Fox полягає в розбитті вихідних матриць A та B на блоки та розподілі їх між паралельними обчислювальними вузлами. Кожен вузол обчислює свою частину результату шляхом множення блоків матриць A та B, які були розподілені між вузлами.

Алгоритм Fox розбиває матриці A та B на блоки розміром ***n/p***, де ***n*** - розмір матриці, а ***p*** - кількість паралельних обчислювальних вузлів. При цьому, кожен блок матриці A множиться на відповідний блок матриці B з допомогою алгоритму множення внутрішніх блоків.

Для того, щоб зібрати результат, кожен вузол повинен відправити свою частину до головного вузла, який збирає частини та обчислює фінальний результат.

*Стрічковий:*

Мультипоточний стрічковий алгоритм множення матриць - це алгоритм, який використовує кілька потоків для розпаралелювання обчислень та прискорення процесу множення матриць.

Мультипоточний стрічковий алгоритм множення матриць розбиває вихідні матриці на стрічки і використовує кілька потоків для паралельного множення кожної стрічки матриці A на матрицю B. Цей алгоритм дозволяє зменшити час обчислень шляхом розподілу навантаження між потоками.

\*Ця теорія дуже загальна тому в ній пропущені точні деталі розпаралелення алгоритму. Йдемо далі. Спочатку код тих методів і файлів, які мають другорядне відношення до завдання.\*

## Файл Result:

Text

Description automatically generated

Це клас Result, який містить конструктор і приватне поле result типу int[][]. Конструктор приймає масив int[][], який встановлюється як значення поля result.

Цей клас використовується для зберігання результату множення матриць, який представлений у вигляді двовимірного масиву int[][].

## Файл Basic:

Text

Description automatically generated

Це клас Basic, який містить статичний метод multiply. Цей метод виконує звичайне множення двох матриць matrixA і matrixB, переданих у вигляді двовимірних масивів int[][]. Результат множення повертається у вигляді об'єкту класу Result.

У першу чергу, обчислюються розміри матриць: кількість рядків і стовпців у матриці matrixA і кількість стовпців у матриці matrixB. Далі створюється масив result, розмірністю rowsA x colsB, який буде містити результат множення матриць.

Потім за допомогою вкладених циклів обходяться всі елементи матриць і здійснюється операція множення та додавання. Кожен елемент результуючої матриці обчислюється, як сума добутків елементів відповідних рядка і стовпців вихідних матриць. У кінці метод повертає об'єкт класу Result.

## Файл Main:

Text

Description automatically generated

Це статичний метод setUpRandomMatrix, який створює двовимірний масив int[][] розміром m x n і заповнює його випадковими цілими числами в діапазоні від 1 до 9.У кінці метод повертає заповнений масив matrix.

Text

Description automatically generated

Це приватний статичний метод timer, який приймає два аргументи: об'єкт Runnable (інтерфейс для представлення потоку) та назвуалгоритму - algorithhm. Метод використовується для вимірювання часу виконання обчислювальних задач і виведення результату на екран.

Text

Description automatically generated

Це статичний метод TestCalculation, який запускає наші основні алгоритми та тестує швидкість їх швидкість роботи (Basic, Stripe та Fox) для різних розмірів матриць та різної кількості потоків.

Метод містить два масиви sizes та numThreads, що містять розмір матриць та кількість потоків, які будуть використовуватися для кожного тесту.

Далі метод проходить циклом по всіх елементах масиву sizes, виводячи на екран розмір поточної матриці. Для кожного розміру матриці виконується вкладений цикл по елементах масиву numThreads, виводячи на екран кількість потоків, які будуть використовуватися для кожного тесту.

Для кожного тесту створюються дві випадкові матриці з розміром size, які передаються до трьох методів множення матриць: Basic.multiply, StripeAlgorithm.multiply та FoxAlgorithm.multiply.

Для кожного методу множення викликається метод timer, щоб виміряти час виконання операції.

Тепер перейдемо до коду самих алгоритмів. Базовий ми вже розглянули, тоді почнемо зі стрічкового – так як на мою думку він набагато легший в реалізації ніж алгоритм Фокса, хоча ідеї досить схожі проте формули, які використовуються різні.

## Файл StripeAlgorithm:

Цей клас відповідає за Стрічковий алгоритм множення матриць multiply(), що отримує на вхід два двовимірних цілочисельних масиви (matrixA і matrixB) та кількість потоків numOfThreads і повертає об'єкт Result. Метод обчислює матричне множення матриць A та B з використанням багатопотокового підходу.

1. Для початку нам треба перевірити чи взагалі матриці будуть перемножуватися, а для цього вони повинні мати однакову розмірність. Кількість рядків і стовпців у кожній вхідній матриці визначається за допомогою оператора length. Якщо кількість стовпців у матриці A не дорівнює кількості рядків у матриціB, генерується виключення IllegalArgumentException.
2. Далі створюється порожня результуюча матриця з розмірами основних матриць numOfRowsA та numOfColsB.
3. Створюється синхронізований об’єкт для обміну інформації між потоками StripeSyncObject.
4. Створюється масив об'єктів StripeThread(потоки) довжиною в задану кількість numOfThreads.
5. Потім ми ініціалізуємо два списки listOfRows та listOfIndexes. Вони будуть використовуватися для розбиття рядків матриці A на набори рядків, які будуть оброблятися кожним потоком.
6. Цикл for проходиться по кожному рядку матриці A, додає поточний рядок до listOfRows і додає його індекс до listOfIndexes. Якщо кількість доданих рядків дорівнює rowsPerThread, поточні listOfRows і listOfIndexes передаються як аргументи новому потоку (об'єкту класу) StripeThread. Конструктор StripeThread приймає ці аргументи, а також syncObject, threadIndex і result.
7. start() викликається для кожного об'єкту StripeThread у масиві, щоб почати його виконання.
8. join() викликається для кожного об'єкту StripeThread, щоб дочекатися його завершення.

Створюється новий об'єкт Result з обчисленою матрицею результатів.

Таким чином, метод multiply() розбиває рядки матриціA на підмножини, які призначаються різним потокам для паралельної обробки. Спільний стан між потоками зберігається в об'єкті StripeSyncObject. Метод повертає результат перемноження матриць A та B як новий об'єкт Result.

## Файл StripeSyncObject:

A screen shot of a computer screen

Description automatically generated with medium confidence

Це основний клас StripeSyncObject і він створений для обміну даними між потоками при множенні. Клас включає в себе два методи getAndBlockColumn() та unblockColumn(), які також синхронізовані за допомогою synchronized.

Конструктор класу отримує на вхід масив matrix. Він ініціалізує список ArrayList з назвою columns, який зберігає стовпці матриці. Цикл for проходить по кожному стовпчику матриці і додає його до списку стовпчиків.

Перший метод getAndBlockColumn() отримує на вхід індекс і перевіряє, чи по цьому індексу вже є у списку blockedColumns. Якщо так, то метод чекає, поки інший потік не викличе unblockColumn() для того самого індексу, перш ніж продовжити роботу. Метод wait() знімає блокування з об'єкта і дозволяє іншим потокам отримати його. Якщо індексу немає у списку blockedColumns, він додає його і повертає стовпець, пов'язаний з цим індексом, зі списку columns.

Другий метод unblockColumn() отримує на вхід теж індекс, видаляє по ньому зі списку blockedColumns колонку і викликає notifyAll(), щоб розбудити всі потоки, що очікують.

Таким чином, клас StripeSyncObject використовується для синхронізації доступу до спільного списку стовпців між декількома потоками.

## A screenshot of a computer program Description automatically generated with medium confidenceФайл StripeThread:

Клас StripeThread є підкласом класу Thread і використовується для паралельного виконання множення частини двох матриць і відповідає за роботу з потоками також.

Клас має конструктор, який отримує параметри rowsOfMatrixA, rowsNumber, syncObj, currentColumnIndex і result та ініціалізує відповідні змінні екземплярів.

Клас також перевизначає метод run(), успадкований від класу Thread. Метод run() використовує цикл для перебору кожного рядка у списку rowsOfMatrixA, який є підмножиною рядків матриці A, за які відповідає цей потік.

1. Усередині циклу запускається ще один цикл, який обчислює точковий добуток поточного рядка матриці A на один стовпець матриці B. Це робиться для певної кількості стовпців, яка визначається змінною columnsCount.
2. Метод syncObj.getAndBlockColumn(currentColumnIndex) викликається для отримання блокування поточного стовпця матриці B, за який відповідає даний потік.
3. Точковий добуток поточного рядка матриці A на поточний стовпець матриці B обчислюється шляхом перебору елементів стовпця матриці B і множення їх на відповідні елементи рядка матриці A, а потім додавання добутків.
4. Результат точкового добутку зберігається у масиві результатів.
5. Метод syncObj.unblockColumn(prevRightRowIndex) викликається для зняття блокування з попереднього стовпця матриці B, за який відповідав цей потік.
6. Змінна currentColumnIndex оновлюється і вказує на наступний стовпець матриці B, за який відповідає цей потік.

Цикл продовжується до тих пір, поки для поточного рядка матриці A не будуть обчислені точкові добутки для всіх стовпців, за які відповідає даний потік, а зовнішній цикл продовжується до тих пір, поки не будуть обчислені добутків всіх рядків, за які відповідає даний потік.

Як результат маємо обчислення підмножини добутків точок між рядками матриці A та стовпцями матриці B використовуючи синхронізований об’єкт, а саме спільний StripeSyncObject для синхронізації доступу до стовпців матриці B між декількома потоками.

Тепер перейдемо до коду, який відповідає за Фокса.

## Файл FoxAlgorithm:

В цьому файлі у нас присутні два основні методи – це multiply() і divideMatrix().

Почнемо з методу divideMatrix(), який відповідає за те, щоб матриця ділилася на певні блоки для подальшого множення.

A picture containing text, screenshot, font

Description automatically generated

Цей метод приймає двовимірну квадратну матрицю і ціле число gridSize як вхідні параметри і повертає чотиривимірний масив цілих чисел, що представляє розділену матрицю. Параметр gridSize задає кількість блоків, на які слід розбити матрицю в кожному рядку та стовпчику. Наприклад, якщо gridSize дорівнює 2, то матриця буде розбита на 4 блоки однакового розміру. Спочатку обчислює розмір кожного блоку шляхом ділення розміру матриці на gridSize. Потім він створює чотиривимірний масив цілих чисел з розмірами gridSize x gridSize x blockSize x blockSize.

Потім він ітераційно проходить через кожен блок, де i та j представляють індекси рядка та стовпця поточного блоку, а k - індекс рядка в блоці. Вона копіює елементи з вихідної матриці у відповідний блок 4D масиву за допомогою функції System.arraycopy().

Нарешті, повертає чотиривимірний масив цілих чисел.

Другий метод multiply(), він як і попередній метод є частиною класу FoxAlgorithm і представляє собою реалізацію алгоритму множення матриць Фокса. Він перемножує дві квадратні матриці А і В, використовуючи numOfThreads потоків, де розмір кожного блоку має ділитися на квадратний корінь з numOfThreads.

Сам метод спочатку перевіряє, що вхідні матриці є квадратними і мають однаковий розмір, а також, що розмір блоку є допустимим. Потім він створює порожню матрицю resultMatrix і ділить матрицю A і матрицю B на блоки однакового розміру за допомогою методу divideMatrix. Створюється об'єкт FoxSyncObject для синхронізації потоків і зберігання блоків, а також створюється і запускається потік FoxThread для кожного блоку матриці A і матриці B. Нарешті, він чекає на завершення роботи всіх потоків і повертає A screen shot of a computer program

Description automatically generated with low confidenceматрицю resultMatrix.

## Файл FoxSyncObject:

A picture containing text, screenshot, font

Description automatically generated

Цей клас використовується для зберігання блоків лівої та правої матриць та управління доступом до них екземплярів FoxThread. Клас має наступні властивості та методи:

Властивості:

* listOfBlocksLeftMatrix: 4-вимірний масив, що представляє блоки лівої матриці, де listOfBlocksLeftMatrix[i][j] представляє блок у позиції (i, j).
* listOfBlocksRightMatrix: 4-вимірний масив, що представляє блоки правої матриці, де listOfBlocksRightMatrix[i][j] представляє блок у позиції (i, j).
* blockedBlocksOfRightMatrix: ArrayList з int[], що представляє індекси заблокованих блоків правої матриці.
* blockedBlocksOfLeftMatrix: масив ArrayList з int[], що представляє індекси заблокованих блоків лівої матриці.

**A picture containing text, screenshot, font

Description automatically generated**

Метод getAndBlockBlocksPair - це синхронізований метод, який блокує блоки, доки не з'явиться незаблокована пара блоків для множення. Метод отримує чотири аргументи, які представляють індекси пари блоків для множення, а саме: leftX, leftY, rightX та rightY. Він входить в цикл while, який продовжує чекати, поки поточна пара блоків не буде заблокована. Він перевіряє це за допомогою виклику методу isBlocksBlocked, який буде показаний далі. Якщо поточну пару блоків не заблоковано, метод додає пару блоків до списків заблокованих блоків blockedBlocksOfLeftMatrix та blockedBlocksOfRightMatrix відповідно, і повертає пару блоків у вигляді трьохвимірного масиву. Якщо поточну пару блоків заблоковано, потік очікує на повідомлення, яке вказує на зміну статусу пари блоків.

За допомогою цього метода можна бути впевненим, що кожна пара блоків множиться лише один раз, блокуючи їх, як тільки вона використовується потоком, і розблоковуючи її, коли множення завершено. Цей механізм гарантує, що жодні два потоки не намагатимуться одночасно помножити одну і ту ж пару блоків.

A picture containing text, screenshot, software, font

Description automatically generated

Метод unblockBlocksPair таклж є частиною класу FoxSyncObject і використовується для зняття блокування з пари блоків, отриманих з матриць listOfBlocksLeftMatrix та listOfBlocksRightMatrix. Він отримує індекси блоків у матрицях і видаляє їх з відповідних заблокованих списків індексів блоків.

Спочатку циклічно переглядає заблокований список BlocksOfLeftMatrix і видаляє пару індексів, які відповідають заданим індексам leftX і leftY. Аналогічно, циклічно переглядається список blockedBlocksOfRightMatrix і видаляється пара індексів, яка відповідає заданим індексам rightX і rightY.

Після видалення індексів із заблокованих списків викликається метод notifyAll(), щоб розбудити всі потоки, які могли очікувати блокування для цих блоків.

І останній метод це isBlocksBlocked(), який перевіряє, чи пара блоків наразі заблокована, чи ні. Якщо пара блоків знаходиться в блоках blockedBlocksOfLeftMatrix або blockedBlocksOfRightMatrix, то повертається true. В іншому випадку повертається false, що означає.

Тут блок ідентифікується за його індексами leftX, leftY (для лівої матриці) та rightX, rightY (для правої матриці), які однозначно визначають позицію блоку в матриці, перевіряють чи будь-який з блоків у blockedBlocksOfLeftMatrix або blockedBlocksOfRightMatrix має ті самі індекси, що й задана пара блоків, і якщо так, то повертає true. В іншому випадку повертається false.

A screen shot of a computer program

Description automatically generated with low confidence

## Файл FoxThread:

A screen shot of a computer program

Description automatically generated with low confidence

Клас FoxThread представляє собою потік, який виконує частину множення алгоритму Фокса. Він отримує на вхід індекси сітки блоків для обчислення (i та j), кількість блоків у рядку/стовпчику сітки (blocksGridSize), об'єкт синхронізації (syncObject), який використовується для координації доступу потоку до блоків вхідних матриць, матрицю результату (result), яку потрібно заповнити обчисленим блоком, та розмір блоків (blockSize) вхідних матриць.

A picture containing text, screenshot, font

Description automatically generated

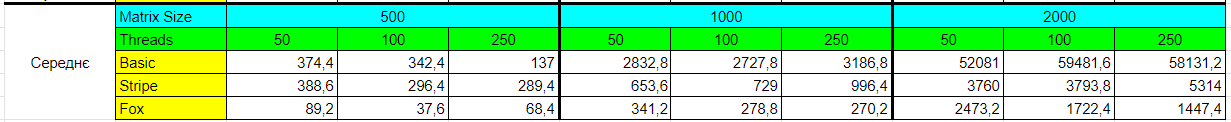
Метод run() класу FoxThread перевизначає метод run() класу Thread, який виконується при запуску потоку і ітераційно перебирає блоки blocksGridSize. У кожній ітерації він викликає метод getAndBlockBlocksPair() об'єкта syncObject, передаючи чотири параметри: (i + k) % blocksGridSize, i, j і (i + k) % blocksGridSize. Метод getAndBlockBlocksPair() повертає двовимірний цілочисельний масив blocks. Потім викликається метод multiplyMatrices() з двома елементами масиву blocks як аргументами.

Метод multiplyMatrices() перемножує дві матриці matrixA та matrixB і зберігає результат у двовимірному цілочисельному масиві result, використовуючи три вкладені цикли для перебору елементів матриць. Нарешті, викликається метод unblockBlocksPair() об'єкта syncObject з тими самими параметрами, що й раніше, для розблокування заблокованих блоків.

## Приклад виконання і роботи програми:

Запустимо програму і заміряємо данні для матриць розмірністю 500, 1000, 2000 та кількістю потоків для кожної в 50, 100, 250.

Знизу приклад 5 проведених експериментів та середнього серед цих 5. Усі заміри проводилися в мілісекундах.



A picture containing text, screenshot, parallel, number

Description automatically generated

А це загальна статистика для всіх матриць:

A screenshot of a computer

Description automatically generated with medium confidence

**ВИСНОВОК**

У ході лабораторної роботи були розроблені реалізації стрічкового та алгоритму Фокса для множення матриць. Експерименти проходили на Восьмиядерному AMD Ryzen 9 4000 з тактовою частотою 3.3 – 4.4 Гц та на Windows 11. Як ви можете помітити на таблицях зверху основні параметри були 50, 100, 250 потоків та матриці розмірністю 500, 1000, 2000. Варіюючи кількість виділених потоків виконання, бачимо, що найбільшої ефективності досягаємо з алгоритмом Фокса при близькій до ідеальної кількості потоків (такої, що співпадає з кількістю віртуальних ядер). Варіюючи розмірність матриці, стає зрозуміло, що для відносно великих матриць більше ефективним завжди буде алгоритм Фокса. Однак, через складність конфігурації, для малих матриць швидше виконання саме стрічкового.