Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

**Паралельні та розподілені обчислення**

**Лабораторна робота №3**

Виконав:

Студент групи ПМі-33

Тимчишин Ярема

Львів 2023

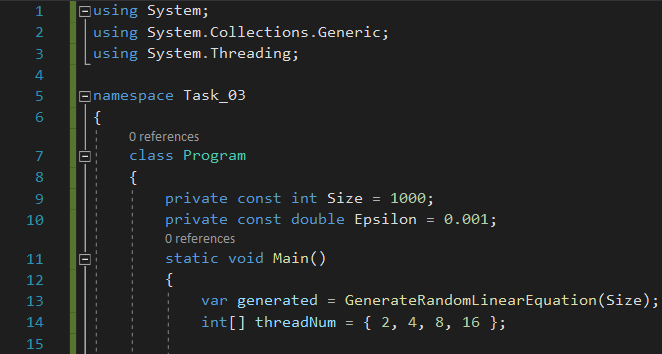
**Тема:** Розв’язування системи лінійних алгебраїчних рівнянь.

**Завдання:** Написати програму для розв’язування системи лінійних алгебраїчних рівнянь за допомогою розпаралелення довільного з існуючих методів розв’язування СЛАР.

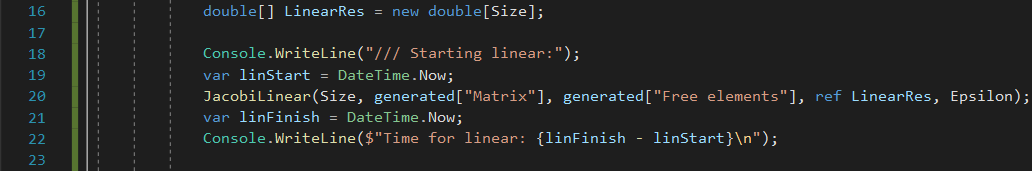
**Хід роботи**

Використовуючи мову програмування C# написав програму для розв’язування системи лінійних алгебраїчних рівнянь за допомогою розпаралелення методу Якобі.

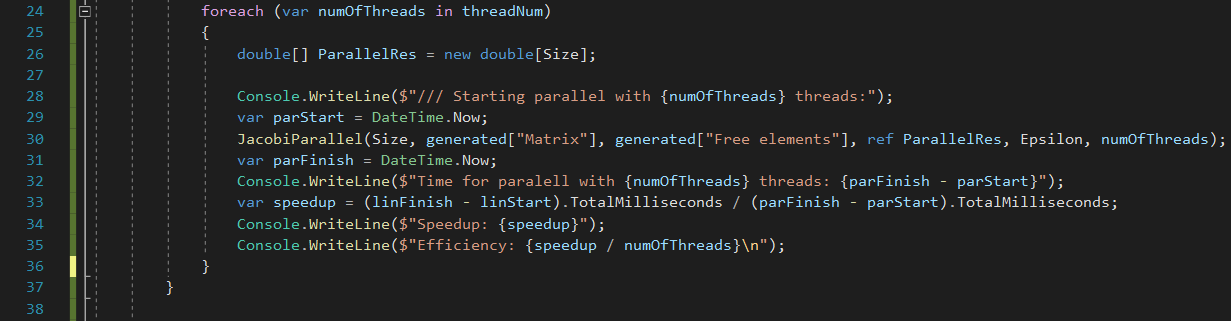
Почав із оголошення константи ‘Size’, яка вказує на кількість рівнянь в лінійній системі, та константи ‘Epsilon’, яка використовуватиметься для визначення точності обчислень в методі Якобі. В головному методі ‘Main’ почав із виклику методу ‘GenerateRandomLinearEquation’, який генерує випадкову лінійну систему рівнянь з кількістю рівнянь, визначеною вищезгаданою константою ‘Size’, і про який я розкажу пізніше. Потім створив масив ‘threadNum’ із заданими кількостями потоків для тестування паралельної версії алгоритму.



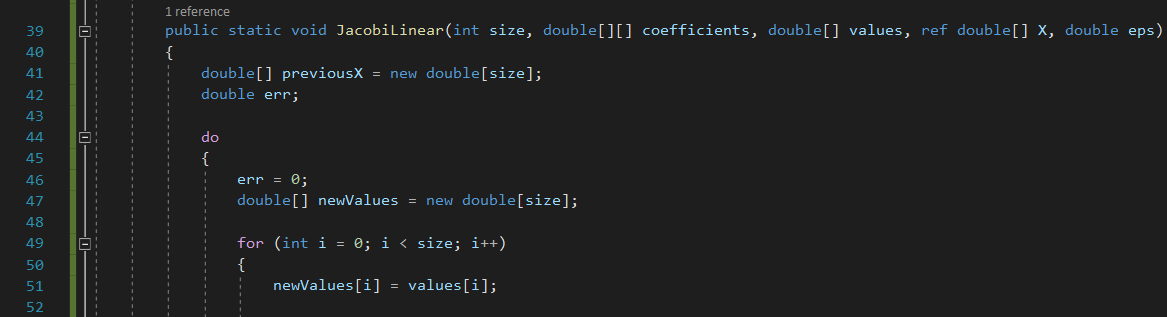
Далі в головному методі ‘Main’ розпочав тестування послідовного алгоритму для розв’язування лінійної системи рівнянь. Почав із оголошення масиву ‘LinearRes’, в якому зберігатиметься результат роботи послідовного алгоритму. Записав початок виконання послідовного алгоритму у змінну ‘linStart’. Після чого, викликав метод ‘JacobiLinear’, який вирішує лінійну систему рівнянь методом Якобі, використовуючи лише один потік (детальніше про цей метод я розповім пізніше). Записав кінець виконання послідовного алгоритму в змінну ‘linFinish’. Після чого, вивів на консоль час, який знадобився для виконання послідовного алгоритму, розрахований як різниця між ‘linStart’ та ‘linFinish’.

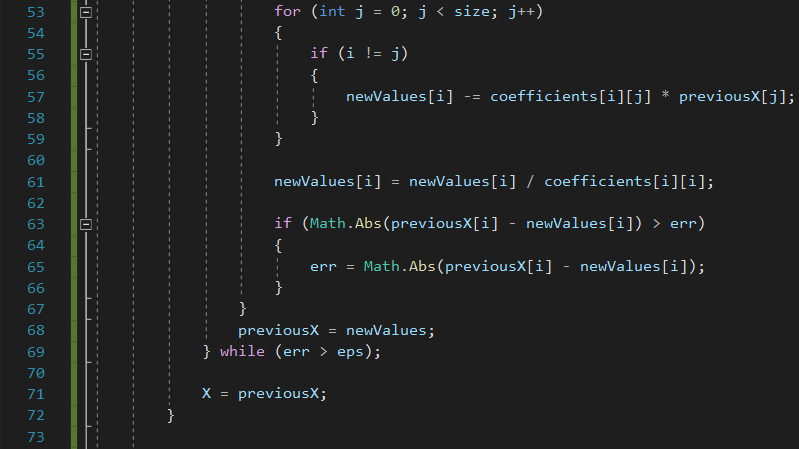


Опісля, почав тестування паралельного алгоритму для розв’язування лінійної системи рівнянь з різною кількістю потоків. Почав із циклу ‘foreach’, який проходить по кількості потоків(‘numOfThreads’), які ми хочемо використовувати для паралельного алгоритму. Нагадаю, що ‘threadNum’ це масив із заданими кількостями потоків які ми хочемо протестувати (2, 4, 8, 16). Створив масив ‘ParallelRes’ для збереження результатів роботи паралельного алгоритму з певною кількістю потоків. Записав початок виконання паралельного алгоритму у змінну ‘parStart’. Викликав метод ‘JacobiParallel’, який вирішує лінійну систему рівнянь з певною кількістю потоків, та який я детальніше опишу пізніше. Записав кінець виконання паралельного алгоритму в змінну ‘parFinish’. Після чого, вивів час, який знадобився для виконання паралельного алгоритму з певною кількістю потоків. Потім оголосив змінну ‘speedup’, в яку записав результат обчислення коефіцієнту прискорення, який визначає, в скільки разів паралельний алгоритм є швидший за лінійний. Порахував я цей коефіцієнт прискорення поділивши час затрачений послідовним алгоритмом на час затрачений паралельним алгоритмом. Вивів коефіцієнт прискорення на консоль. А також вивів на консоль коефіцієнт ефективності, який вказує на ефективність використання потоків. Цей коефіцієнт я обчислив поділивши коефіцієнт прискорення на кількість потоків.

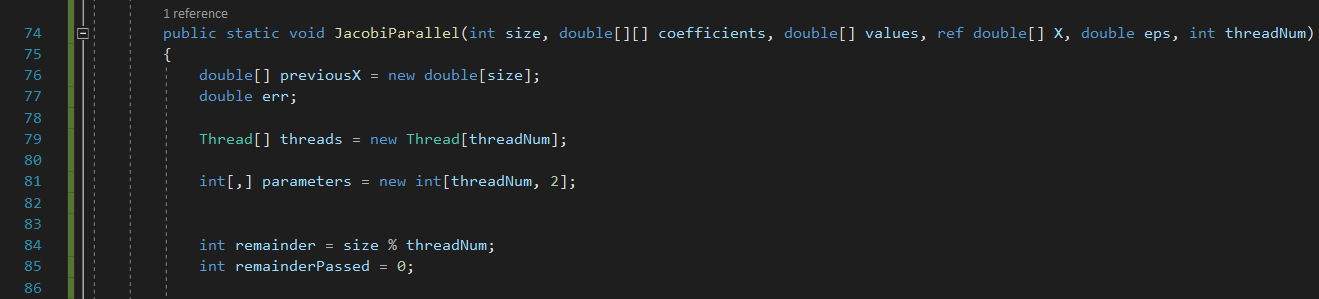


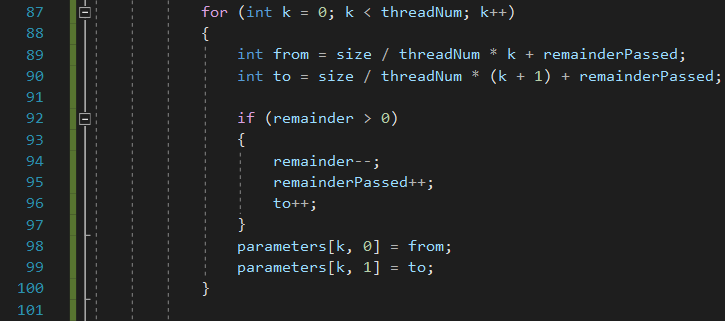
Метод розв’язування лінійних рівнянь за допомогою алгоритму Якобі, з використанням послідовного обчислення. Оголосив публічний статичний метод ‘JacobiLinear’, який приймає наступні аргументи: ‘size’ (кількість рівнянь у системі); ‘coefficients’ (двовимірний масив, що представляє матрицю коефіцієнтів системи рівнянь); ‘values’ (одновимірний масив, який містить вільні члени системи рівнянь); ‘X’ (посилання на одновимірний масив, в який будуть записані розв’язки системи); ‘eps’ (точність, яка визначає умову завершення ітерацій). Тепер, в самому методі, починаю із оголошення масиву ‘previousX’, який міститиме попереднє наближене рішення системи, та змінної ‘err’ для відстеження похибки (різниця між попереднім та поточним наближеним рішенням). Цикл ітерацій ‘do…while (err > eps)’ – це цикл, який виконується принаймні один раз і продовжується, допоки помилка ‘err’ більша за задану точність ‘eps’. Тобто ітерації тривають допоки не досягнута необхідна точність. Тепер, в самому циклі, починаю із присвоєння змінній ‘err’ нуля та створення масиву ‘newValues’, який зберігатиме нове наближення розв’язку системи лінійних рівнянь. Кожен елемент цього масиву представляє нове наближення до значення змінної, яку ми шукаємо в системі рівнянь. Вкладений цикл ‘for’ обчислює нові значення розв’язку системи на основі попереднього наближення та коефіцієнтів системи. Детальніше розглянемо як він працює: ‘newValues[i] = values[i];’ (присвоюємо нове значення з правої частини системи; ‘for (int j = 0; j < size; j++)’ (вкладений цикл для обчислення нового наближення розв’язку на основі попереднього наближення та коефіцієнта; ‘newValues[i] = newValues[i] / coefficients[i][i];’ (обчислюємо нове наближення розв’язку для поточного рівняння); ‘if (Math.Abs(previousX[i] – newValues[i]) > err) {…}’ (порівнюємо поточну похибку з попередньою та оновлюємо значення похибки). По закінченні вкладеного циклу ‘for’, оновлюю попереднє наближення новими значеннями (‘previousX = newValues;’). Потім, після закінчення циклу ‘do…while (err > eps)’, присвоюю оновлені значення розв’язку вихідному масиву ‘X’ (‘X = previousX;’). На цьому закінчується послідовний метод ‘JacobiLinear’.

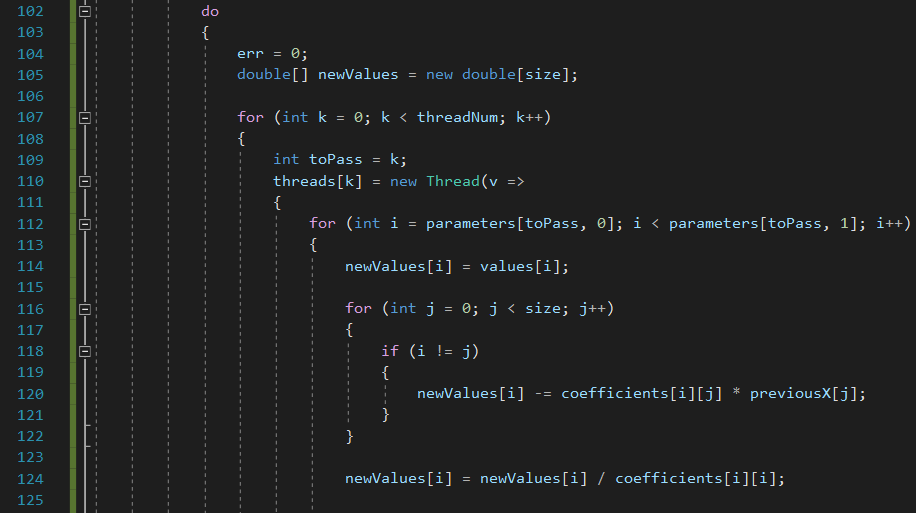


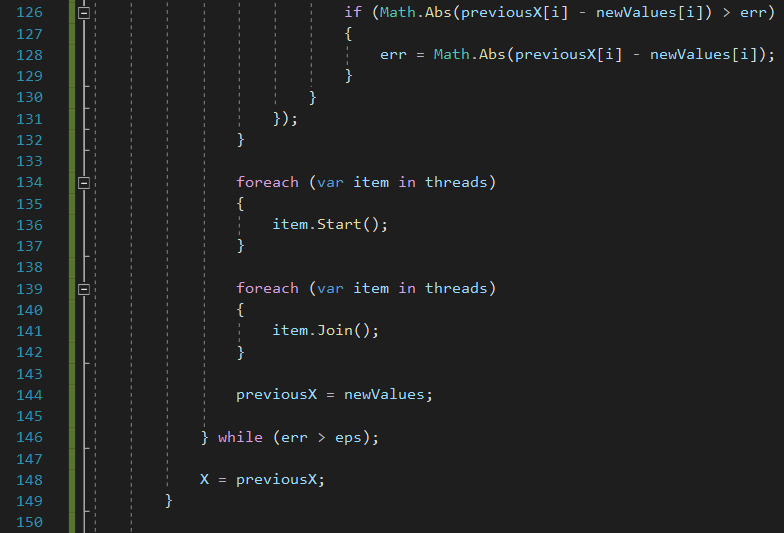


Метод розв’язування системи лінійних рівнянь за допомогою алгоритму Якобі для паралельного виконання з використанням потоків. Оголосив публічний статичний метод ‘JacobiParallel’, який приймає наступні аргументи: ‘size’ (кількість рівнянь у системі); ‘coefficients’ (двовимірний масив, що представляє матрицю коефіцієнтів системи рівнянь); ‘values’ (одновимірний масив, який містить вільні члени системи рівнянь); ‘X’ (посилання на одновимірний масив, в який будуть записані розв’язки системи); ‘eps’ (точність, яка визначає умову завершення ітерацій); ‘threadNum’ (кількість потоків). Тепер, в самому методі, починаю із оголошення масиву ‘previousX’, який міститиме попереднє наближене рішення системи, змінної ‘err’ для відстеження похибки (різниця між попереднім та поточним наближеним рішенням), масиву потоків ‘threads’, який буде містити об’єкти потоків, двовимірного масиву ‘parameters’ для збереження параметрів, які будуть використовуватися для розподілу роботи між потоками. Також оголошую змінну ‘remainder’ в якій обчислюю і зберігаю залишок від ділення розміру системи на кількість потоків, що допоможе розподілити роботу між потоками рівномірно. І ще ініціалізую змінну ‘remainderPassed’ зі значенням 0, яка використовуватиметься для відстеження кількості потоків, які отримали додаткові рядки обчислень через залишок. Після чого, розпочинаю цикл ‘for (int k = 0; k < threadNum; k++)’: змінна ‘k’ служить лічильником для кількості потоків, які будуть створені для паралельних обчислень; змінна ‘from’ обчислює та зберігає початковий індекс для потоку з номером ‘k’, розрахунок відбувається так, щоб кожний потік обробляв приблизно однакову кількість обчислень; змінна ‘to’ в свою чергу обчислює та зберігає кінцевий індекс для потоку з номером ‘k’. Вкладений цикл ‘if (remainder > 0)’ перевіряє чи є ще залишок від ділення кількості обчислень на кількість потоків. Якщо є залишок, то відбувається коригування кінцевого індексу та зменшення лічильника залишків. Після циклу ‘if (remainder > 0), зберігаю початковий та кінцевий індекси обчислень для заданого потоку ‘k’ у відповідний рядок масиву ‘parameters’. Після циклу ‘for (int k = 0; k < threadNum; k++), починається головний цикл ‘do…while (err > eps)’, який виконується принаймні один раз і продовжується, допоки помилка ‘err’ більша за задану точність ‘eps’. Тобто ітерації тривають допоки не досягнута необхідна точність. Тепер, в самому циклі, починаю із присвоєння змінній ‘err’ нуля та створення масиву ‘newValues’, який зберігатиме нове наближення розв’язку системи лінійних рівнянь. Кожен елемент цього масиву представляє нове наближення до значення змінної, яку ми шукаємо в системі рівнянь. Вкладений цикл ‘for (int k = 0; k < threadNum; k++)’ – це цикл для створення та запуску потоків, який пройдеться через кількість потоків, вказану в ‘threadNum’. Змінна ‘toPass’ приймає значення ‘k’, тобто індекс поточного потоку. Це значення потім використовується в лямбда-виразі для визначення діапазону індексів, який цей конкретний потік повинен обчислити. Опісля, створюю новий потік з лямбда-виразом, який виконується в цьому потоці (‘threads[k] = new Thread(v => {…});’). Цикл ‘for (int i = parameters[toPass, 0]; i < parameters[toPass, 1]; i++)’ ітерується через певний діапазон індексів для кожного потоку у масиві ‘parameters’. Тут ‘parameters[toPass, 0]’ – це нижня межа діапазону індексів, яку ми хочемо обробити в потоці з індексом toPass, а ‘parameters[toPass, 1]’ – відповідно верхня межа діапазону індексів. Цей вкладений цикл ‘for (int i = parameters[toPass, 0]; i < parameters[toPass, 1]; i++)’, так само як це було і в послідовному методі, обчислює нові значення розв’язку системи на основі попереднього наближення та коефіцієнтів системи. Детальніше розглянемо як він працює: ‘newValues[i] = values[i];’ (присвоюємо нове значення з правої частини системи; ‘for (int j = 0; j < size; j++)’ (вкладений цикл для обчислення нового наближення розв’язку на основі попереднього наближення та коефіцієнта; ‘newValues[i] = newValues[i] / coefficients[i][i];’ (обчислюємо нове наближення розв’язку для поточного рівняння); ‘if (Math.Abs(previousX[i] – newValues[i]) > err) {…}’ (порівнюємо поточну похибку з попередньою та оновлюємо значення похибки). По закінченні вкладеного циклу ‘for’, використовую два цикла ‘foreach’. Перший проходить через масив потоків ‘threads’ і викликає метод ‘Start()’ для кожного потоку. Це запускає кожний потік та починає виконання паралельних обчислень. Другий цикл ‘foreach’ пройдеться також по масиву потоків ‘threads’, але викликає метод ‘Join()’ для кожного потоку. Метод ‘Join()’ призводить до блокування виконання основного потоку (у нашому випадку, це головна програма ‘Main’) до тих пір, поки кожен потік не завершить своє виконання. Таким чином, головна програма чекає, поки всі потоки закінчать обчислення. Після чого я оновлюю попереднє наближення новими значеннями (‘previousX = newValues;’). І потім, після закінчення циклу ‘do…while (err > eps)’, присвоюю оновлені значення розв’язку вихідному масиву ‘X’ (‘X = previousX;’). На цьому закінчується паралельний метод ‘JacobiParallel’.

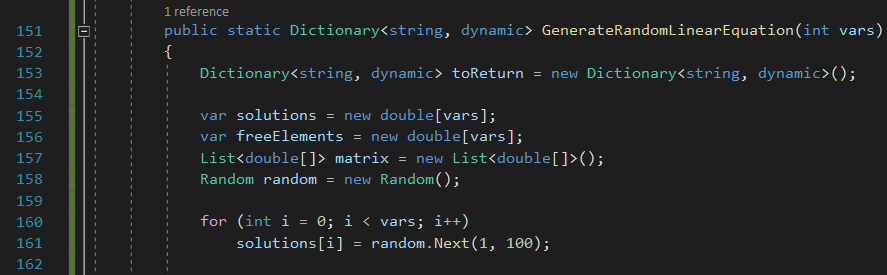


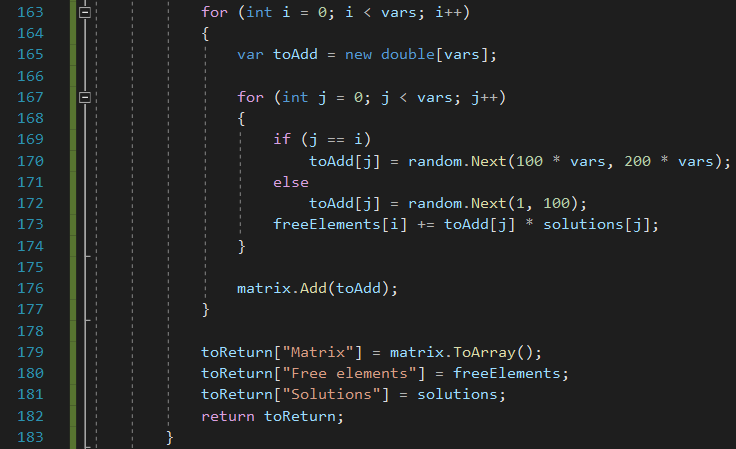






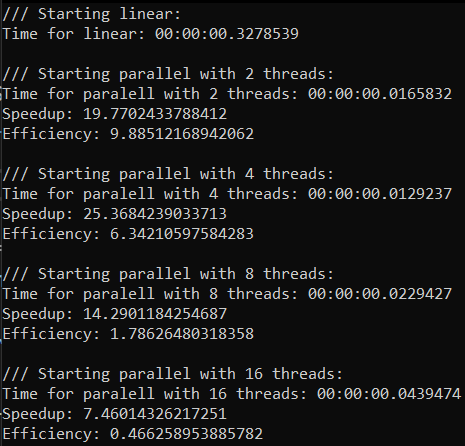
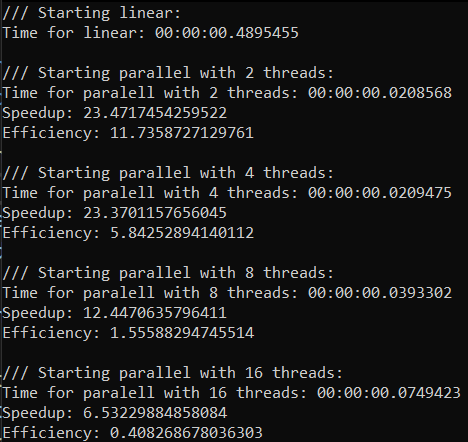
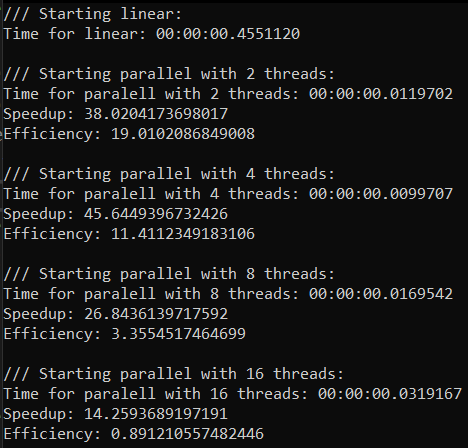
Допоміжний метод для створення випадкової системи лінійних рівнянь. Спочатку створюю словник ‘toReturn’, який міститиме дані щодо випадкової системи лінійних рівнянь. Також створюю масиви ‘solutions’, ‘freeElements’ та ‘matrix’, які будуть містити коефіцієнти, вільні члени та матрицю системи лінійних рівнянь відповідно. Маленький цикл ‘for’ для генерації випадкових значень для рішень рівнянь ‘solutions’. Для кожного рішення береться випадкове значення у діапазоні від 1 до 100. Ще один, але вже більший, цикл ‘for’, який генеруює випадкові коефіцієнти для кожного рівняння, які утворюють матрицю коефіцієнтів. Для діагональних елементів (де i == j), коефіцієнти вибираються у великому інтервалі (100 \* vars до 200 \* vars), інакше – у діапазоні від 1 до 100. Випадкові коефіцієнти множаться на відповідні рішення, і результати додаються до вільних членів ‘freeElements’. В кінці, отримані матриця коефіцієнтів, вільні члени та рішення додаються до словника ‘toReturn’ з відповідними ключами, після чого, сформований словник ‘toReturn’ повертається як результат роботи методу.



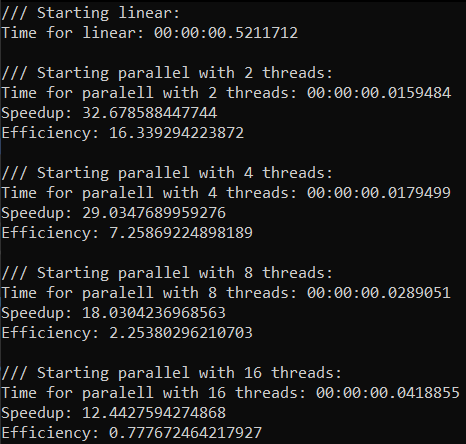
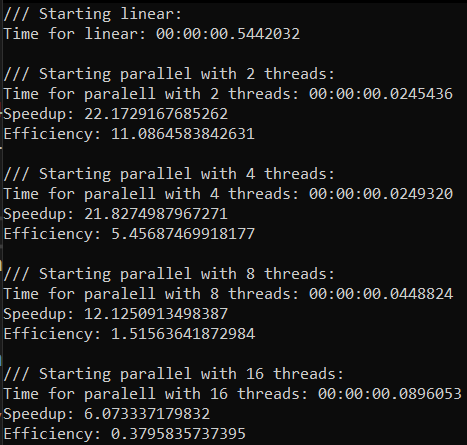
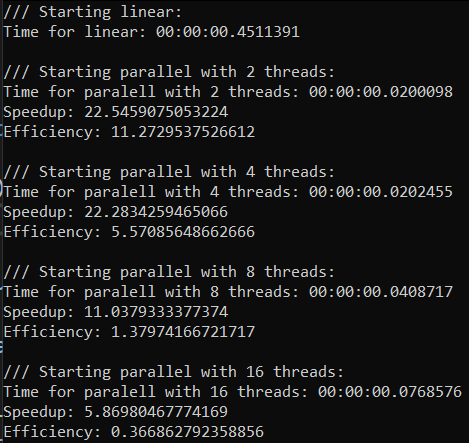


Результати роботи програми для різних вхідних значень:

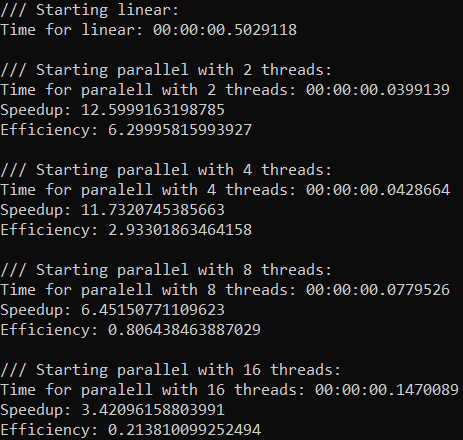
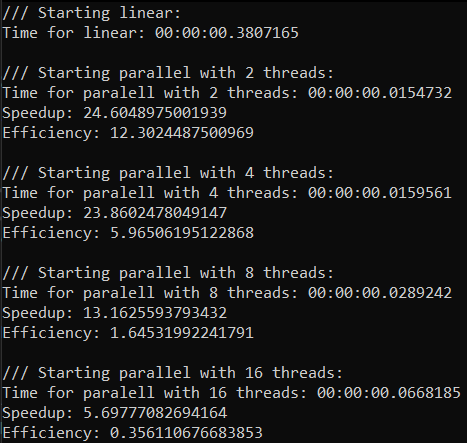
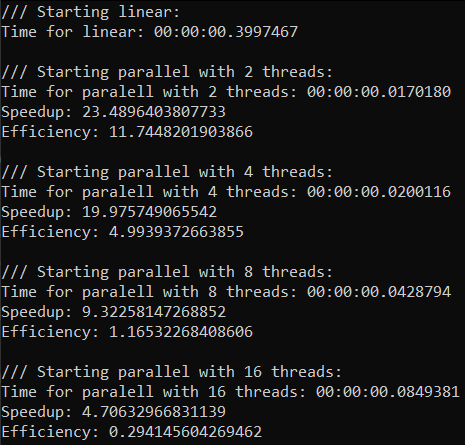
Const int Size = 2; Const int Eps = 0.001

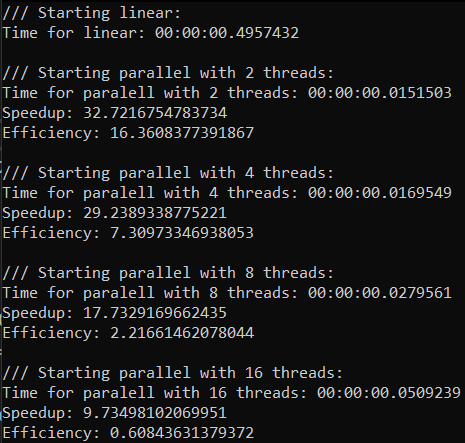
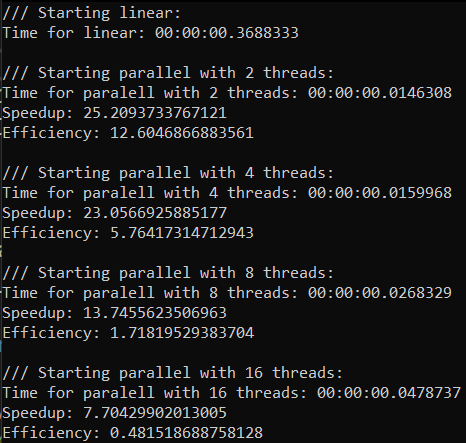
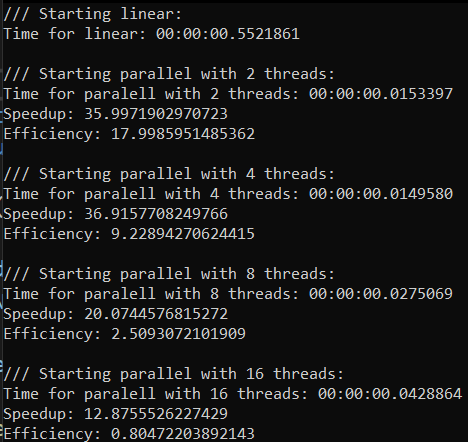
Const int Size = 4; Const int Eps = 0.001

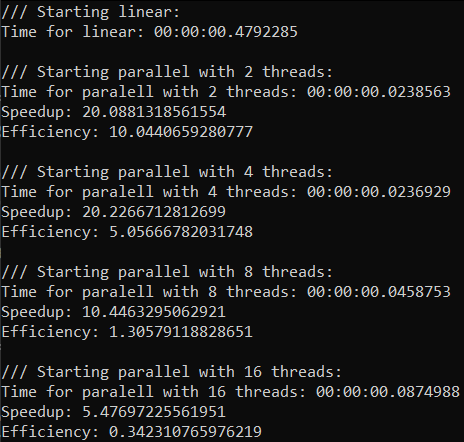
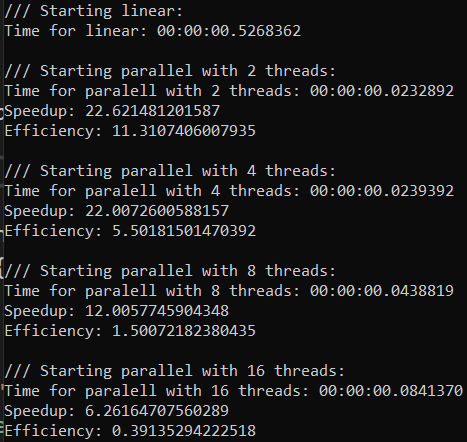
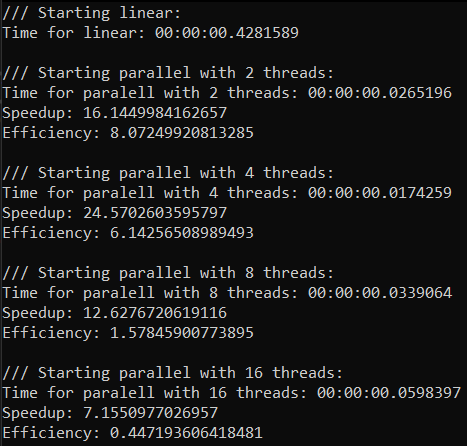
Const int Size = 8; Const int Eps = 0.001

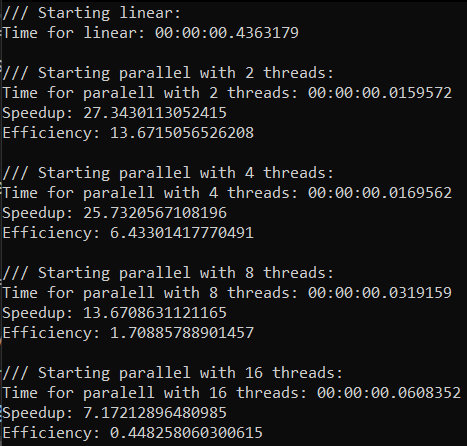
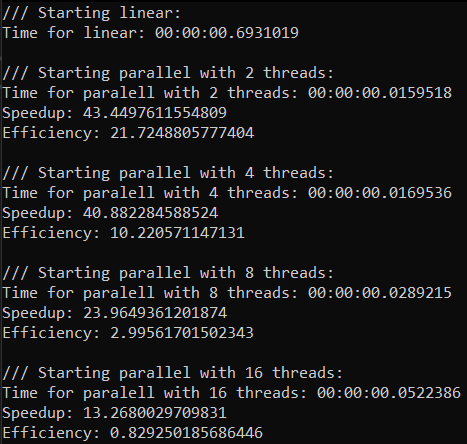
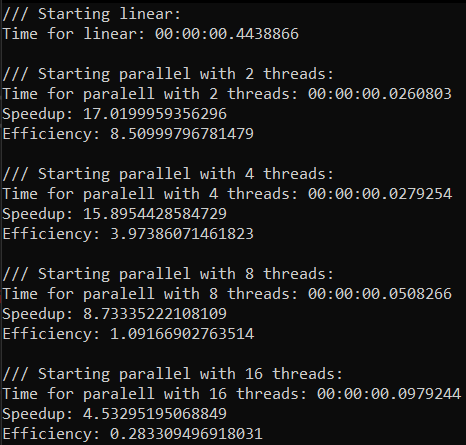
Const int Size = 16; Const int Eps = 0.001

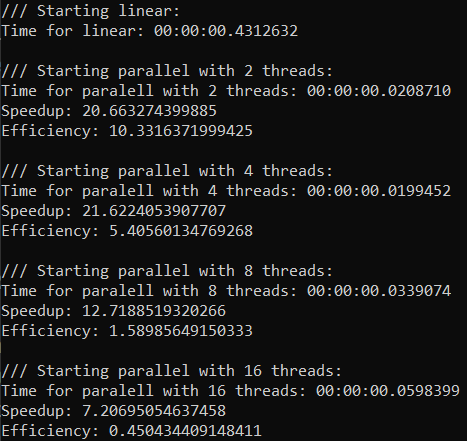
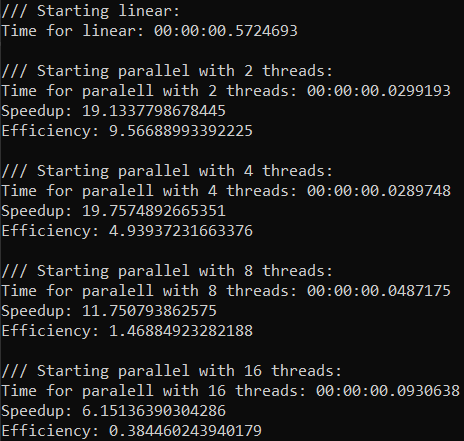
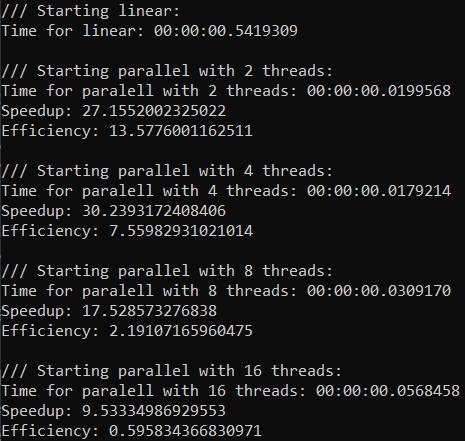
Const int Size = 50; Const int Eps = 0.001

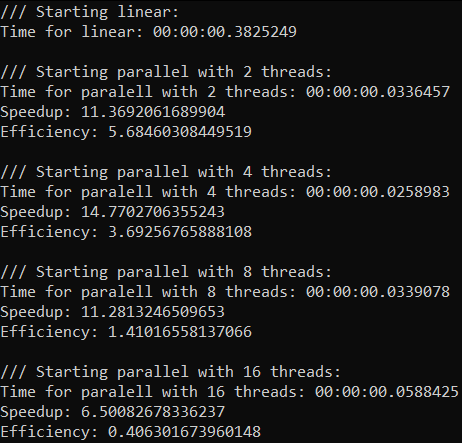
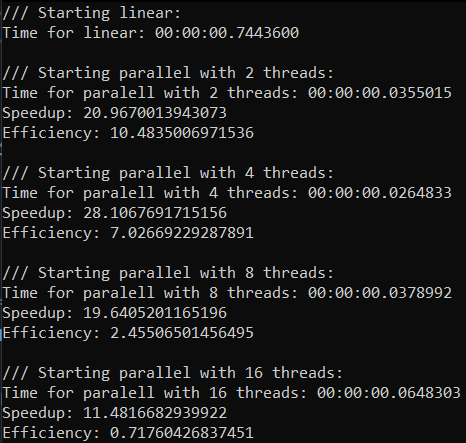
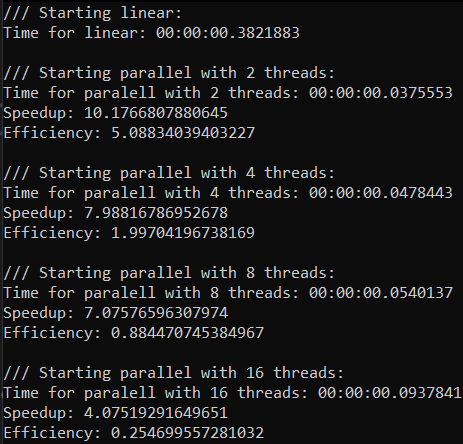
Const int Size = 100; Const int Eps = 0.001

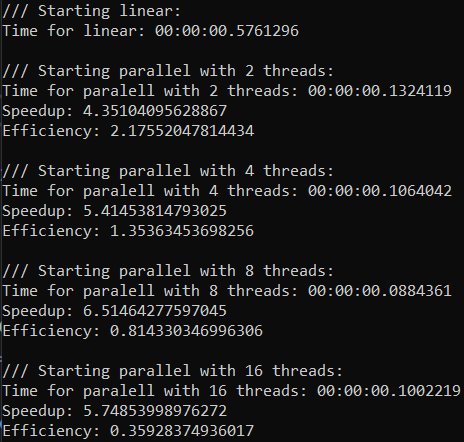
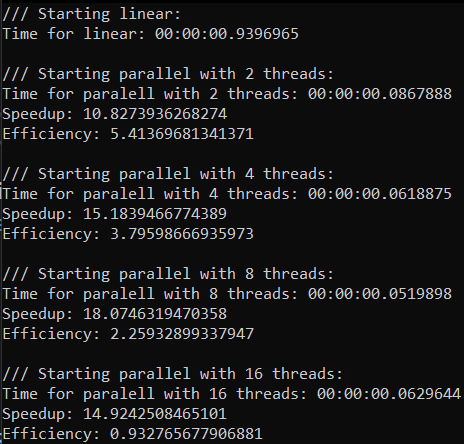
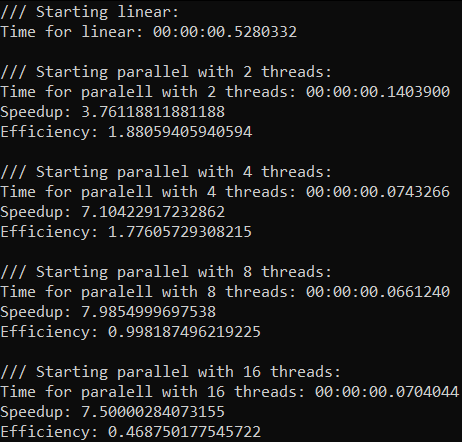
Const int Size = 256; Const int Eps = 0.001

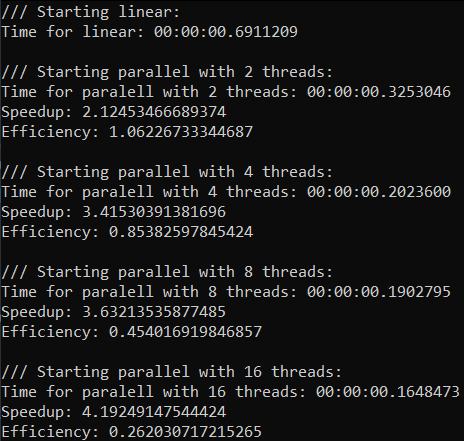
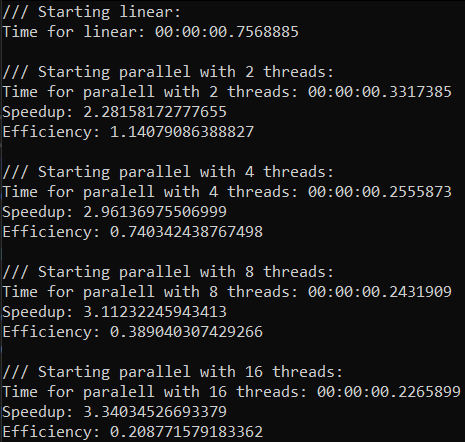
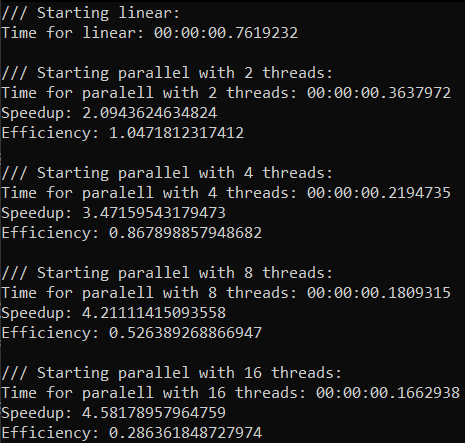
Const int Size = 500; Const int Eps = 0.001

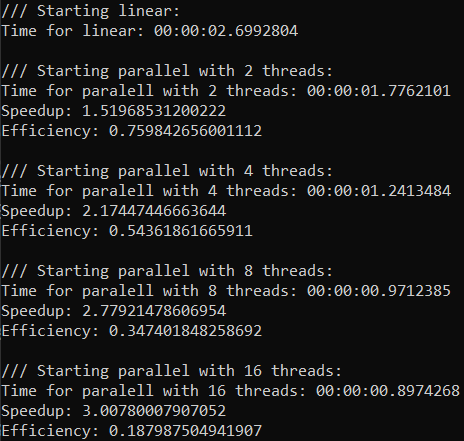
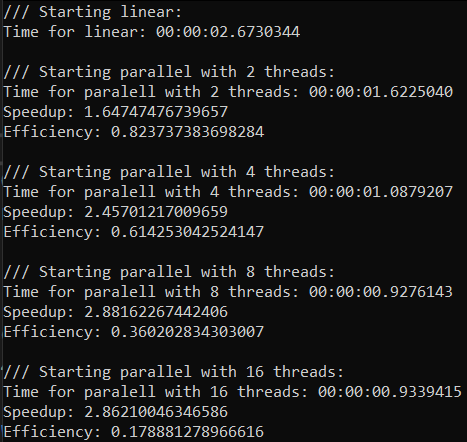
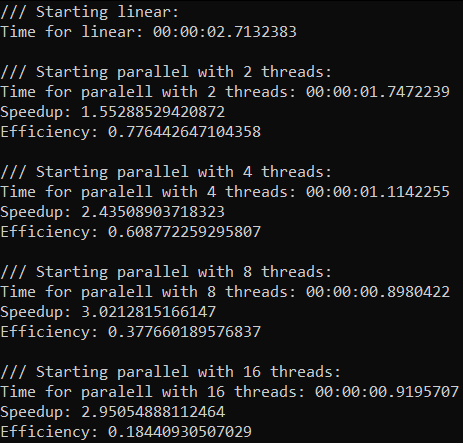
Const int Size = 1000; Const int Eps = 0.001

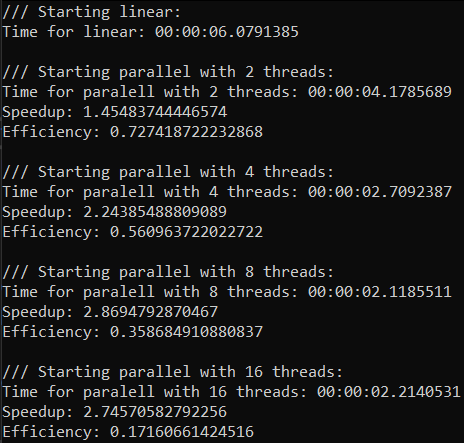
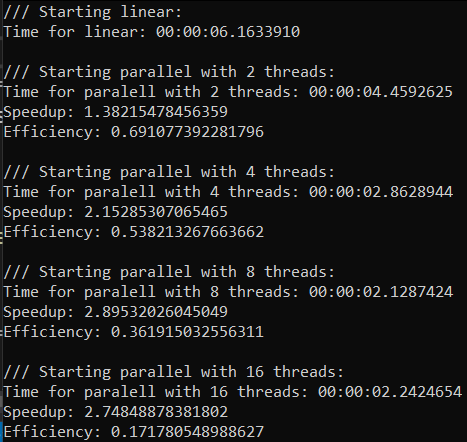
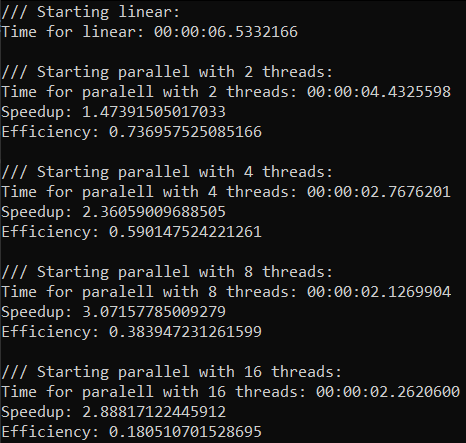
Const int Size = 2048; Const int Eps = 0.001

Const int Size = 5000; Const int Eps = 0.001

Const int Size = 8192; Const int Eps = 0.001

Const int Size = 10000; Const int Eps = 0.001

