

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

САМАРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Лабораторная работа №1

Параллельное и последовательное умножение матриц

Цель работы: ознакомиться с основами многопоточного программирования в C# с помощью библиотеки System. Threading. Tasks.

Задачи работы:

- 1. Изучить базовые принципы многопоточного программирования.
- 2. Реализовать последовательный алгоритм умножения матриц.
- 3. Реализовать многопоточное умножение с помощью Task
- 4. Провести измерение времени выполнения обоих методов.
- 5. Вывести результаты в графическом интерфейсе.
- 6. Провести сравнительный анализ производительности.

Задание на лабораторную работу:

Написать программу, реализующую произведение матриц для однопоточного и многопоточного режима. Представить таблицу, зависимости времени выполнения для данных режимов от размерности матриц А и В. Генерация элементом матриц А и В выполняется случайным образом для натуральных чисел от 0 до 9. Параллельные процессы выполняются либо построчно, либо поэлементно (это два разных метода, аргументировать выбор метода, показать на данных, почему выбрали такой). Количество задач = количество потоков.

Требования к графическому интерфейсу:

- 1. Запрет ввода букв в числовые поля
- 2. Проверка на пустые поля
- 3. Ограничение диапазона чисел
- 4. Обработка исключений при умножении

Содержание отчета:

- 1. Титульный лист
- 2. Экранные формы работы программы и листинг кода
- 3. Таблица с результатами исполнения.
- 4. График зависимости времени исполнения для одно и двухзадачного режима в зависимости от размера матрицы.

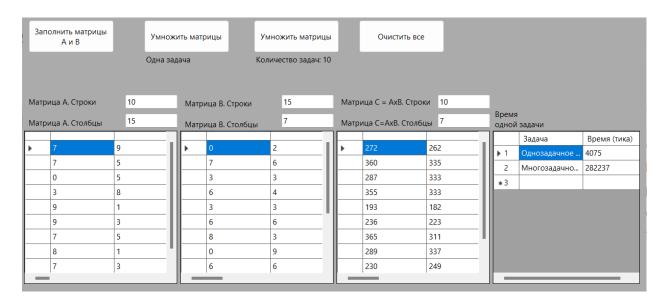


Рисунок 1 – Пример реализации экранной формы

Теоретические сведения:

Формула умножения двух матриц A и B:

Лабораторный практикум «Методы параллельных вычислений на С#»

$$A \times B = C$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{pmatrix}$$

$$c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \qquad \cdots$$

$$c_{23} = a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33}$$

В однопоточном программировании реализуется вложенными циклами.

<u>Ссылочный тип массива - C# reference | Microsoft Learn</u> - Описание многомерных массивов и их обработки.

Для выполнения параллельных вычислений используется класс Task, который позволяет запускать задачи асинхронно.

<u>Task.Run Meтод (System.Threading.Tasks) | Microsoft Learn</u> - Позволяет выполнять операции в отдельных потоках без явного управления потоками.

Для каждой задачи создается отдельный Task, выполняющий вычисления. Task.Run запускает поток, который выполняет какое-то действие. Task.WaitAll гарантирует, что выполнение продолжится только после завершения всех задач.

<u>Task.WaitAll Метод (System.Threading.Tasks) | Microsoft Learn</u> - Позволяет дождаться завершения всех задач перед продолжением выполнения кода.

Замер времени рекомендуется сделать с помощью <u>Stopwatch Класс</u> (System.Diagnostics) | Microsoft Learn

Для ввода матриц использовать DataGridView.