6 ЛАБОРАТОРНАЯ РАБОТА №6

«Исследование способов профилирования программного обеспечения»

6.1 Цель работы

Исследовать критические по времени выполнения участки программного кода и возможности их устранения. Приобрести практические навыки анализа программ с помощью профайлера dotTrace.

6.2 Вариант задания

Разработать программу на основе библиотеки классов, реализованной и протестированной в предыдущей работе. Программа должна как можно более полно использовать функциональность класса. При необходимости для наглядности профилирования в методы класса следует искусственно внести задержку выполнения. Выполнить профилирование разработанной программы, выявить функции, на выполнение которых тратится наибольшее время. Модифицировать программу с целью оптимизации времени выполнения. Выполнить повторное профилирование программы, сравнить новые результаты и полученные ранее, сделать выводы.

6.3 Ход выполнения работы

6.3.1 В начале выполнения работы для одного из классов, разработанных в ходе первой лабораторной работы, была добавлена операция для задержки выполнения, чтобы затем сравнить результаты изменения в скорости выполнения программы. Код класса для профилирования представлен в листинге 6.1.

Листинг 6.1 – Код класса для профилирования

```
public class MatrixCounter
{
    private MatrixCounter() {}
```

```
    public int CountNegativeNumbersInRowsWithZeros(List<List<int>> data)
    {
        Thread.Sleep(5000);
        int result = 0;
        foreach (var row in data)
        {
            if (row.Contains(0))
            {
                var amount = row.Count(x => x < 0);
                result += amount;
            }
        }

        return result;
    }

    public class MatrixCounterBuilder()
    {
        public MatrixCounter Build()
        {
            Thread.Sleep(1000);
            return new MatrixCounter();
        }
    }
}
```

Далее было выполнено профилирование в режиме tracing, его результаты отражены на рисунке 6.1.

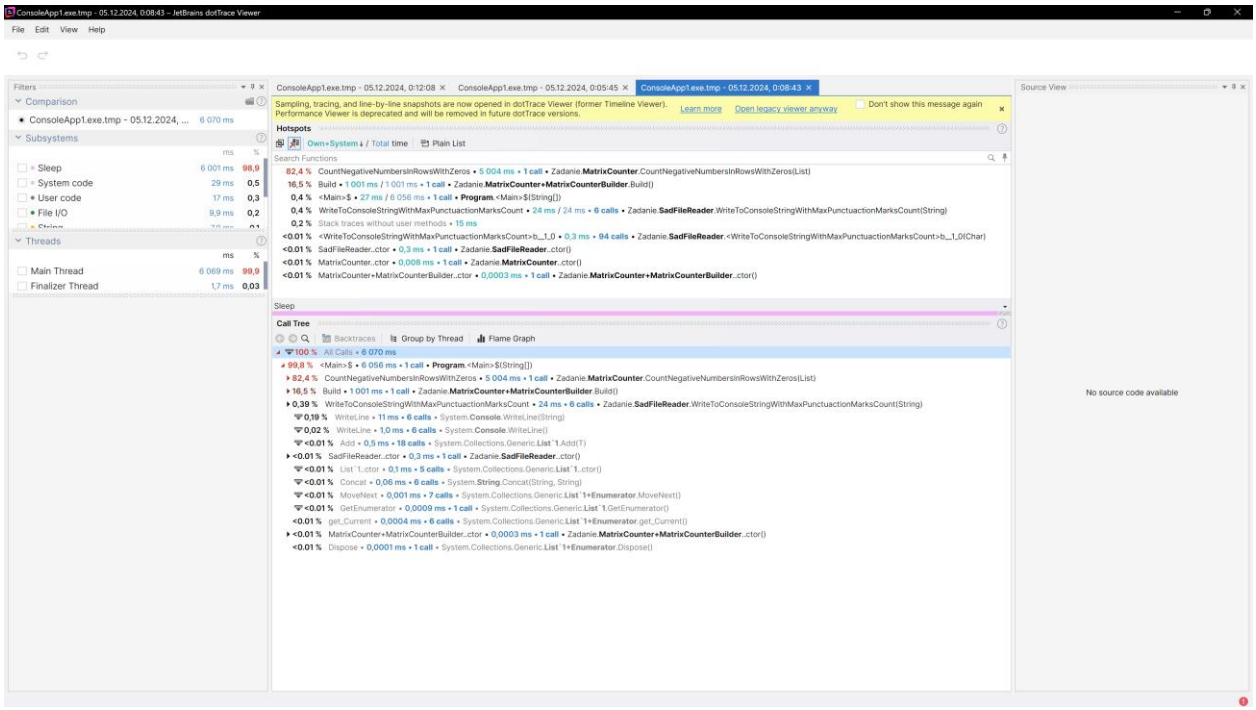

Рисунок 6.1 – Результаты профилирования первого варианта программы

В листинге 6.2 показан класс без внесенных изменений, который покажет реальное время выполнения программы без задержек.

Листинг 6.2 – Код класса для профилирования

```
namespace Zadanie;

public class MatrixCounter
{
    private MatrixCounter() {}

    public int CountNegativeNumbersInRowsWithZeros(List<List<int>> data)
    {
        int result = 0;
        foreach (var row in data)
        {
            if (row.Contains(0))
            {
                var amount = row.Count(x => x < 0);
                result += amount;
            }
```

```
            }

            return result;
        }

        public class MatrixCounterBuilder()
        {
            public MatrixCounter Build()
            {
                return new MatrixCounter();
            }
        }
    }
```

На рисунке 6.2 показан результат профилирования для показанной выше программы.

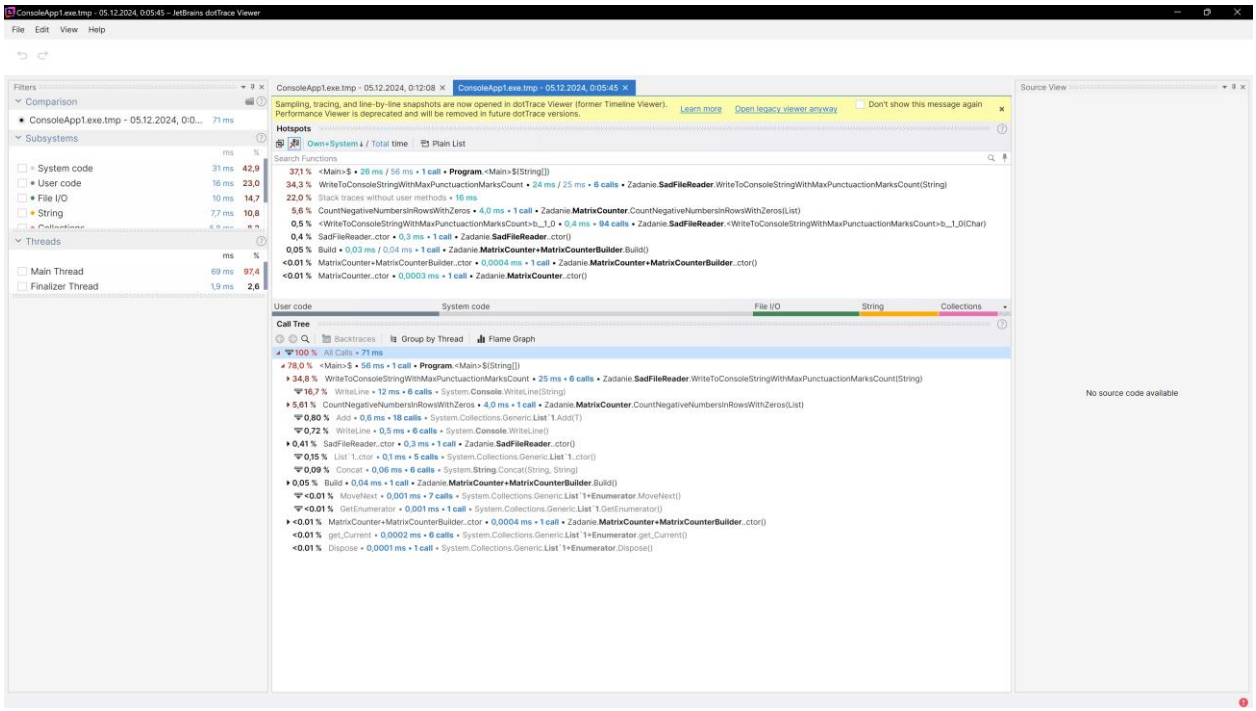

Рисунок 6.2 – Результаты профилирования второго варианта программы

Далее два полученных снимка были подвергнуты сравнению, полученные результат (рисунок 6.3) демонстрирует то, что первый вариант

программы выполняется на 6 секунд дольше, чем второй из-за использования метода остановки потока.

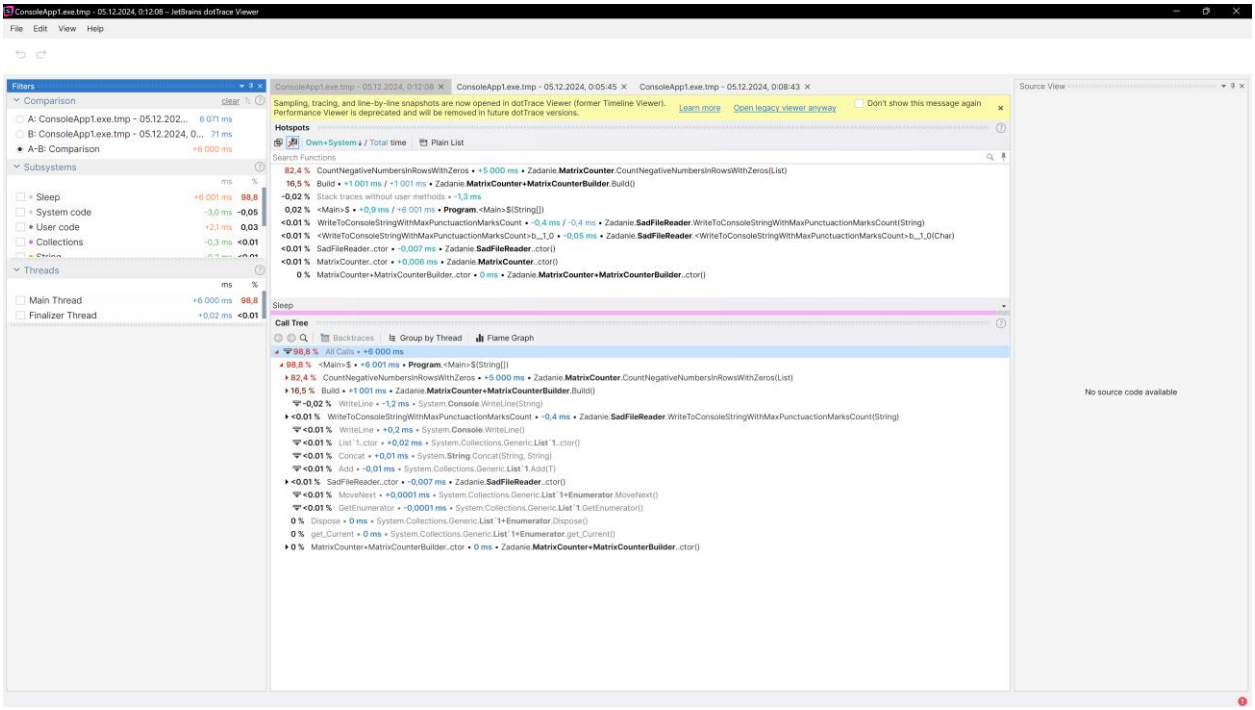

Рисунок 6.3 – Результат сравнения двух снимков

Выводы

В ходе лабораторной работы были исследованы критические по времени выполнения участки программного кода и возможности их устранения. Также были приобретены практические навыки анализа программ с помощью профайлера dotTrace. В конце выполнения лабораторной работы был написан отчет.