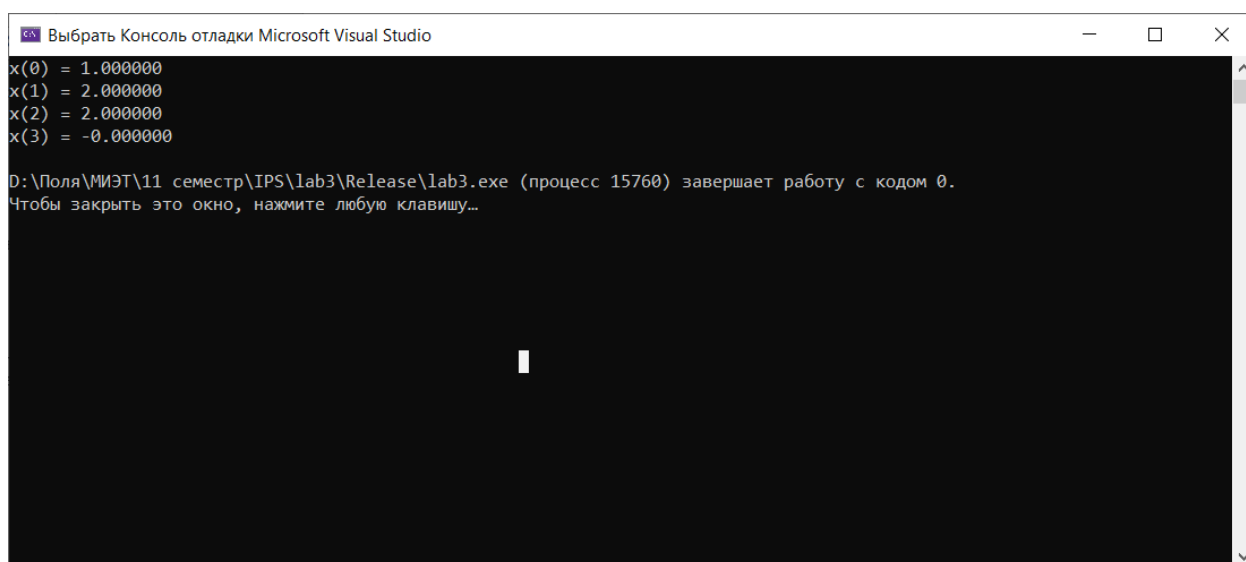


Отчет к занятию 3.

1. В файле **task_for_lecture3.cpp** приведен код, реализующий последовательную версию метода Гаусса для решения СЛАУ. Проанализируйте представленную программу.
2. Запустите первоначальную версию программы и получите решение для тестовой матрицы **test_matrix**, убедитесь в правильности приведенного алгоритма. Добавьте строки кода для измерения времени (см. задание к занятию 2) выполнения прямого хода метода Гаусса в функцию **SerialGaussMethod()**. Заполните матрицу с количеством строк **MATRIX_SIZE** случайными значениями, используя функцию **InitMatrix()**. Найдите решение СЛАУ для этой матрицы (закомментируйте строки кода, где используется тестовая матрица **test_matrix**).

Запуск первоначальной версии программы



```
Выбрать Консоль отладки Microsoft Visual Studio
x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000

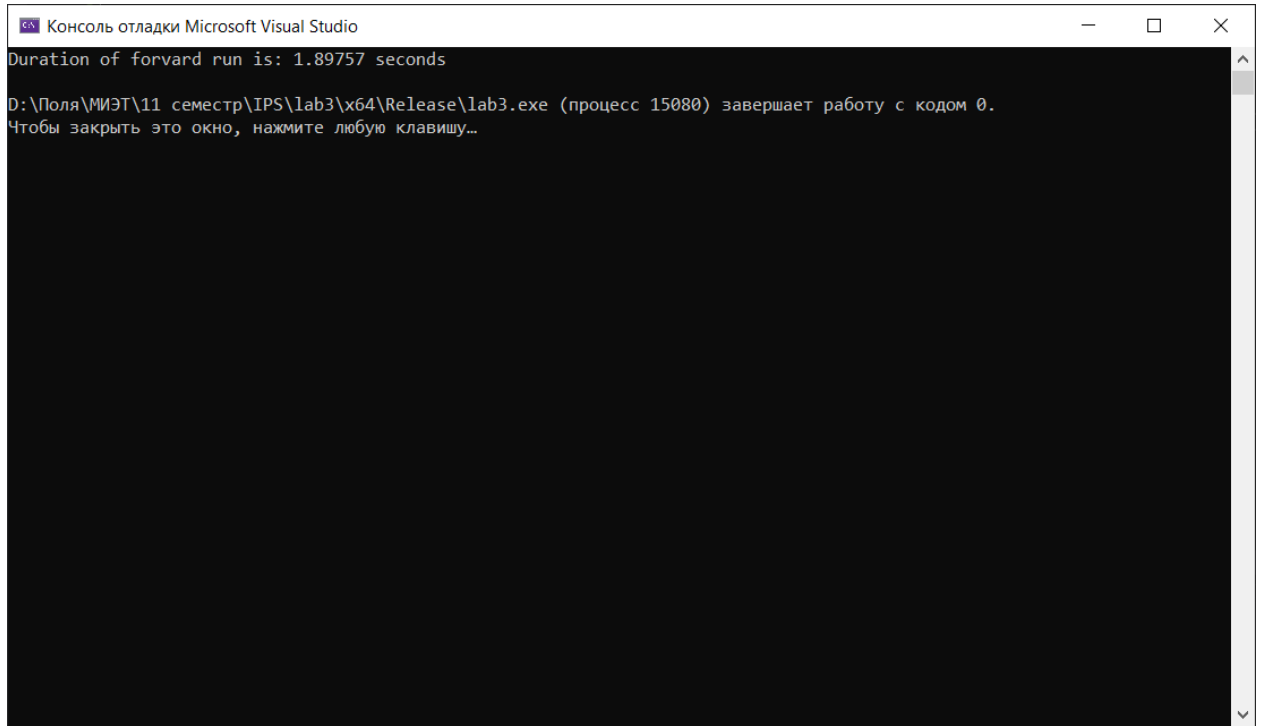
D:\Поля\МИЭТ\11 семестр\IPS\lab3\Release\lab3.exe (процесс 15760) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

Проверка правильности решения с помощью Matlab

```
>> M = [2 5 4 1 20;
1 3 2 1 11;
2 10 9 7 40;
3 8 9 2 37];
>> x = M(:,1:end-1)^(-1)*M(:,end)
x =
1.0000
2.0000
2.0000
-0.0000
```

Решение найдено верно.

Время работы последовательного метода Гаусса для матрицы с количеством строк MATRIX_SIZE:



```
Консоль отладки Microsoft Visual Studio
Duration of forward run is: 1.89757 seconds
D:\Поля\МИЭТ\11 семестр\IPS\lab3\х64\Release\lab3.exe (процесс 15080) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

(без вывода решения)

3. С помощью инструмента **Amplifier XE** определите наиболее часто используемые участки кода новой версии программы.

Сохраните скриншот результатов анализа **Amplifier XE**. Создайте, на основе последовательной функции ***SerialGaussMethod()***, новую функцию, реализующую параллельный метод Гаусса. Введите параллелизм в новую функцию, используя ***cilk_for***.

Примечание: произвести параллелизацию одного внутреннего цикла прямого хода метода Гаусса (определить какого именно), и внутреннего цикла обратного хода. Время выполнения по-прежнему измерять только для прямого хода.

Результаты анализа Amplifier XE

Hotspots Hotspots by CPU Utilization ?

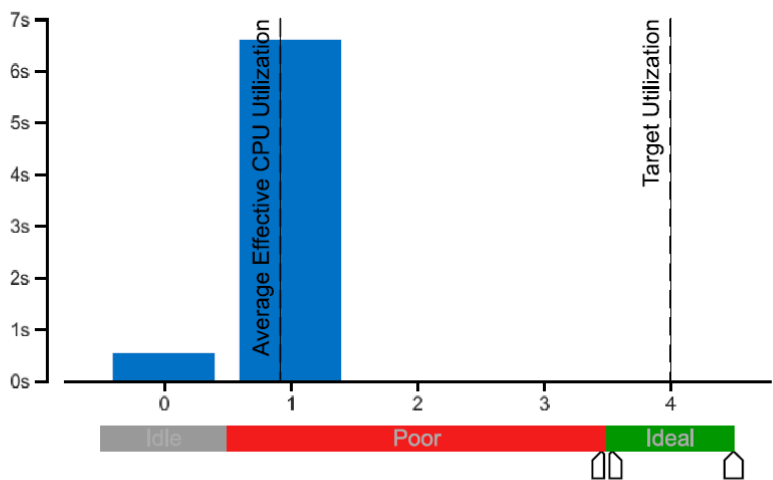
Analysis Configuration	Collection Log	Summary	Bottom-up	Caller/Callee	Top-down Tree	Platform
SerialGaussMethod	lab3.exe	6.374s				
rand	ucrtbased.dll	0.109s				
InitMatrix	lab3.exe	0.040s				
_stdio_common_vfprintf	ucrtbased.dll	0.032s				

*N/A is applied to non-summable metrics.

opportunities to increase parallelism in your application.

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.



Collection and Platform Info

This section provides information about this collection, including result set size and collection platform data.

Application Command Line: D:\Поля\МИЭТ\11 семестр\IPS\lab3\x64\Debug\lab3.exe

Environment Variables:

Operating System: Microsoft Windows 10

Computer Name: DESKTOP-CSEUGT1

Result Size: 2 MB

Collection start time: 22:35:46 31/10/2019 UTC

Collection stop time: 22:35:54 31/10/2019 UTC

Collector Type: User-mode sampling and tracing

Finalization mode: Fast. If the number of collected samples exceeds the threshold, this mode limits the number of processed samples to speed up post-processing.

CPU

Name: Unknown

Frequency: 2.5 GHz

Logical CPU Count: 4

Новая функция, реализующая параллельный метод Гаусса.

```
void ParallelGaussMethod(double** matrix, const int rows, double* result)
{
    int k;
    double koef;

    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    // прямой ход метода Гаусса
    for (k = 0; k < rows; ++k)
    {
        for(int i = k + 1; i < rows; ++i)
        {
            koef = -matrix[i][k] / matrix[k][k];

            cilk_for (int j = k; j <= rows; ++j)
            {
                matrix[i][j] += koef * matrix[k][j];
            }
        }

        high_resolution_clock::time_point t2 = high_resolution_clock::now();

        duration<double> duration = (t2 - t1);
        cout << "Duration of parallel forward run is: " << duration.count() << "
seconds" << endl;

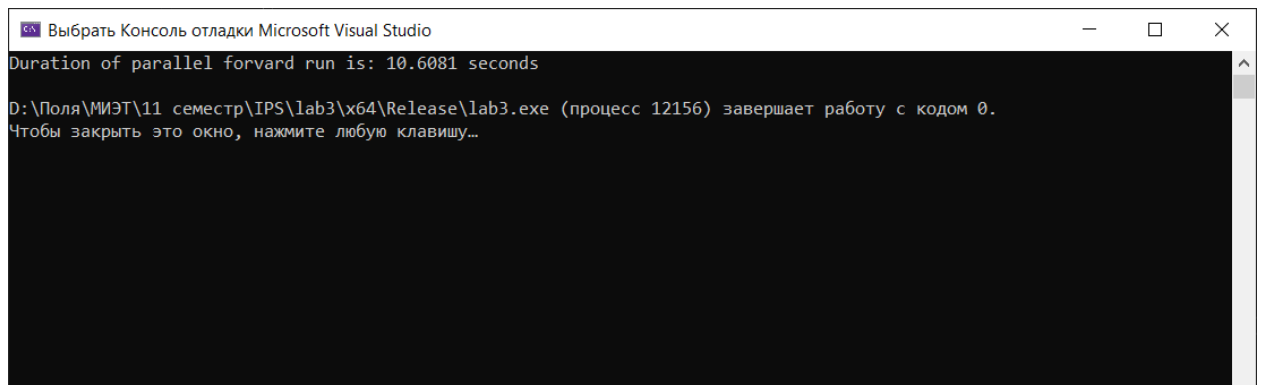
        // обратный ход метода Гаусса
        result[rows - 1] = matrix[rows - 1][rows] / matrix[rows - 1][rows - 1];

        for (k = rows - 2; k >= 0; --k)
        {
            result[k] = matrix[k][rows];

            cilk_for(int j = k + 1; j < rows; ++j)
            {
                result[k] -= matrix[k][j] * result[j];
            }

            result[k] /= matrix[k][k];
        }
    }
}
```

Время работы параллельного метода Гаусса



The screenshot shows a console window titled "Выбрать Консоль отладки Microsoft Visual Studio". The output text is as follows:

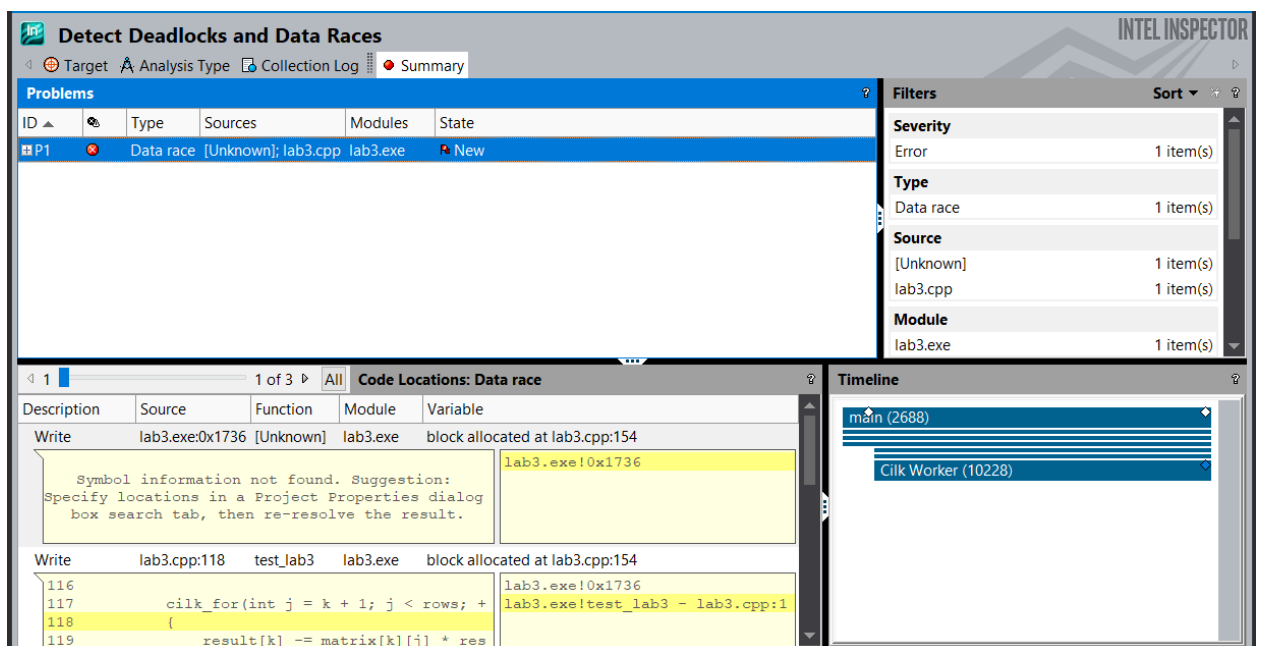
```
Duration of parallel forward run is: 10.6081 seconds

D:\Поля\МИЭТ\11 семестр\IPS\lab3\х64\Release\lab3.exe (процесс 12156) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

Метод стал работать медленнее. Далее используем Inspector XE для поиска ошибок.

4. Далее, используя **Inspector XE**, определите те данные (если таковые имеются), которые принимают участие в гонке данных или в других основных ошибках, возникающих при разработке параллельных программ, и устраните эти ошибки. Сохраните скриншоты анализов, проведенных инструментом **Inspector XE** в случае обнаружения ошибок и после их устранения.

Проверим наличие возможных ошибок



Обнаружена гонка данных.

Изменим функцию для устранения ошибок.

```
void ParallelGaussMethod(double** matrix, const int rows, double* result)
{
    int k;
    double koef;

    high_resolution_clock::time_point t1 = high_resolution_clock::now();

    // прямой ход метода Гаусса
    for (k = 0; k < rows; ++k)
    {
        cilk_for (int i = k + 1; i < rows; ++i)
        {
            koef = -matrix[i][k] / matrix[k][k];

            for (int j = k; j <= rows; ++j)
            {
                matrix[i][j] += koef * matrix[k][j];
            }
        }
    }
}
```

```

    }

    high_resolution_clock::time_point t2 = high_resolution_clock::now();

    duration<double> duration = (t2 - t1);
    cout << "Duration of parallel forward run is: " << duration.count() << " seconds"
    << endl;

    // обратный ход метода Гаусса
    result[rows - 1] = matrix[rows - 1][rows] / matrix[rows - 1][rows - 1];

    for (k = rows - 2; k >= 0; --k)
    {
        cilk::reducer_opadd<double> res(matrix[k][rows]);

        cilk_for(int j = k + 1; j < rows; ++j)
        {
            res -= matrix[k][j] * result[j];
        }

        result[k] = res->get_value() / matrix[k][k];
    }
}

```

Анализ после устранения ошибок

Intel Inspector: Detect Deadlocks and Data Races

Target: Analysis Type: Collection Log: Summary

ID	Type	Sources	Modules	State
P1	Data race [Unknown]; reducer.h; reducer_opadd.h	lab3.exe	New	
P2	Data race [Unknown]	lab3.exe	New	

Filters: Severity: Error (2 item(s)); Type: Data race (2 item(s)); Source: [Unknown] (2 item(s)), reducer.h (1 item(s)), reducer_opadd.h (1 item(s)); Module:

Code Locations: Data race

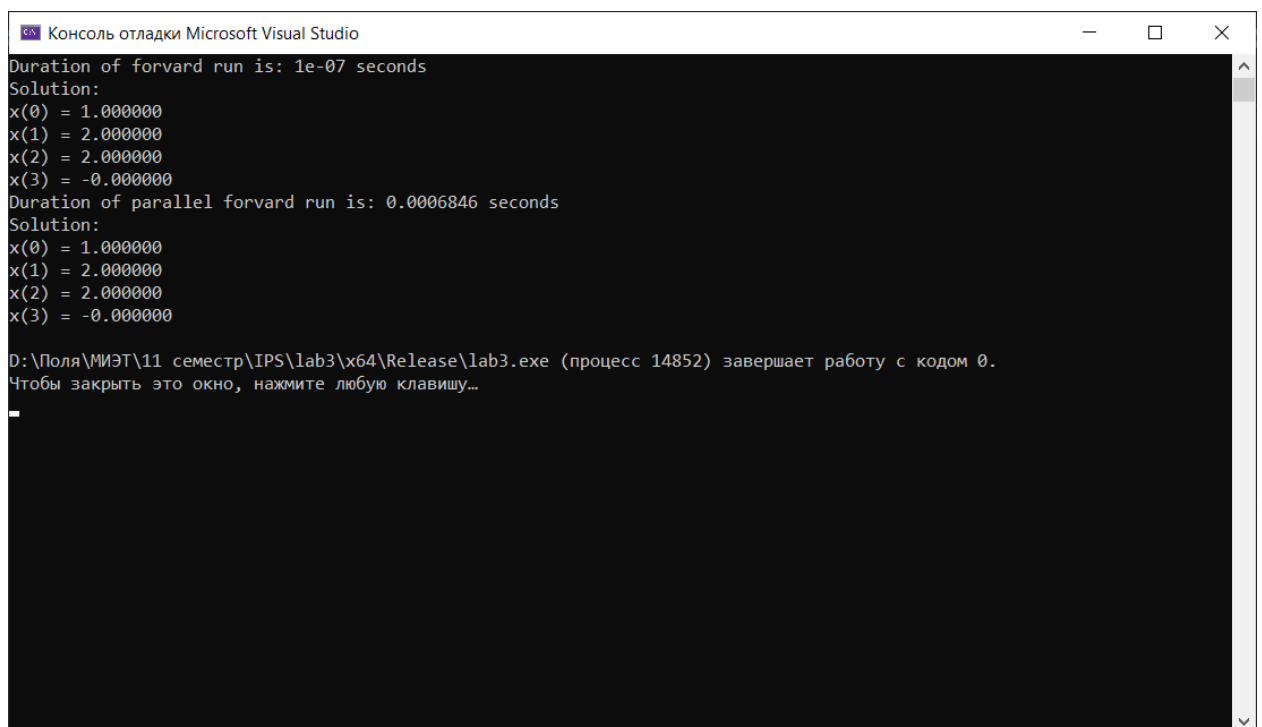
Description	Source	Function	Module	Variable
Read	reducer_opadd.h:265	reduce_wrapper	lab3.exe	block
<pre> 263 * reduce operation. 264 */ 265 void reduce(op_add_view* right) { this- 266 267 /** @name Accumulator variable updates. </pre>				
Write	lab3.exe:0x210a	[Unknown]	lab3.exe	block
<pre> lab3.exe!0x210a </pre> <p>Symbol information not found. Suggestion: Specify locations in a Project Properties dialog box search tab, then re-resolve the result.</p>				

Timeline: main (15192), Cilk Worker (14900)

Найденная в начале ошибка была устранена.

5. Убедитесь на примере тестовой матрицы **test_matrix** в том, что функция, реализующая параллельный метод Гаусса работает правильно. Сравните время выполнения прямого хода метода Гаусса для последовательной и параллельной реализации при решении матрицы, имеющей количество строк **MATRIX_SIZE**, заполняющейся случайными числами. Запускайте проект в режиме **Release**, предварительно убедившись, что включена оптимизация (**Optimization->Optimization=/O2**). Подсчитайте ускорение параллельной версии в сравнении с последовательной. Выводите значения ускорения на консоль.

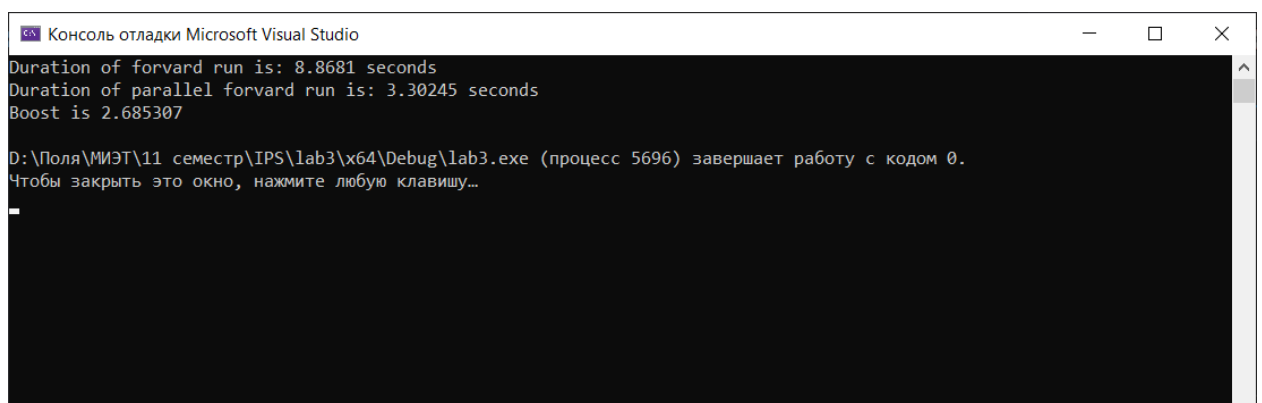
Решение параллельным методом Гаусса для тестовой матрицы test_matrix



```
Консоль отладки Microsoft Visual Studio
Duration of forward run is: 1e-07 seconds
Solution:
x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000
Duration of parallel forward run is: 0.0006846 seconds
Solution:
x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000
D:\Поля\МИЭТ\11 семестр\IPS\lab3\х64\Release\lab3.exe (процесс 14852) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

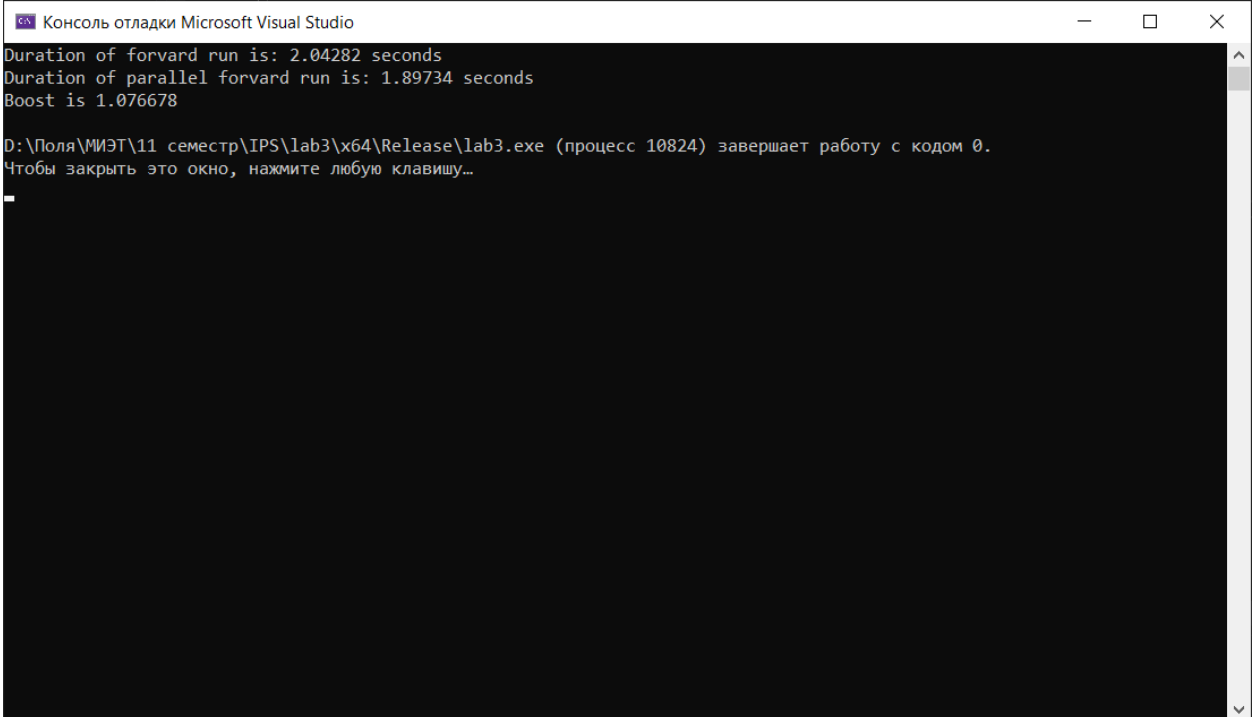
Метод работает верно.

Сравнение времени для последовательного и параллельного методов в режиме Debug



```
Консоль отладки Microsoft Visual Studio
Duration of forward run is: 8.8681 seconds
Duration of parallel forward run is: 3.30245 seconds
Boost is 2.685307
D:\Поля\МИЭТ\11 семестр\IPS\lab3\х64\Debug\lab3.exe (процесс 5696) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

Сравнение времени для последовательного и параллельного методов в режиме Release



```
Консоль отладки Microsoft Visual Studio
Duration of forward run is: 2.04282 seconds
Duration of parallel forward run is: 1.89734 seconds
Boost is 1.076678

D:\Поля\МИЭТ\11 семестр\IPS\lab3\x64\Release\lab3.exe (процесс 10824) завершает работу с кодом 0.
Чтобы закрыть это окно, нажмите любую клавишу...
```

Параллелизация дает ускорение в режиме Debug примерно в 2.685 раз, в режиме Release ускорение практически не наблюдается.

Метод Гаусса параллелится плохо.