

## №2

Вывод первоначальной версии программы:

Solution:

```
x(0) = 1.0000000  
x(1) = 2.0000000  
x(2) = 2.0000000  
x(3) = -0.0000000
```

Проверим решение с помощью numpy :

```
import numpy as np
```

```
A = np.array([  
    [2,5,4,1],  
    [1,3,2,1],  
    [2,10,9,7],  
    [3,8,9,2]  
)  
b = np.array([20,11,40,37])  
print(np.linalg.solve(A, b))
```

```
[ 1.00000000e+00  2.00000000e+00  2.00000000e+00 -3.70074342e-15]
```

Решения совпадают.

Время работы программы для матрицы размером MATRIX\_SIZE :

```
Serial Gauss duration: 753154 microseconds
```

## №3

Hotspots
Hotspots by CPU Utilization
?
📖

Analysis Configuration
Collection Log
Summary
Bottom-up
Caller/Callee
Top-down Tree
Platform

Elapsed Time<sup>?</sup>: 6.217s

CPU Time<sup>?</sup>: 5.295s
Total Thread Count: 1
Paused Time<sup>?</sup>: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	Module	CPU Time <sup>?</sup>
<a href="#">SerialGaussMethod</a>	IPS1.exe	5.076s
<a href="#">rand</a>	ucrtbased.dll	0.200s
<a href="#">free_dbg</a>	ucrtbased.dll	0.010s
<a href="#">malloc</a>	ucrtbased.dll	0.009s

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

Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the [Bottom-up](#) view for in-depth analysis per function. Otherwise, use the [Caller/Callee](#) view to track critical paths for these hotspots.

Explore Additional Insights
Parallelism<sup>?</sup>: 21.3% 📈

Use [Threading](#) to explore more opportunities to increase parallelism in your application.

INSIGHTS

Напишем заведомо неправильную параллелизацию метода Гаусса:

```
int ParallelGaussMethod(double** matrix, const int rows, double* result)
{
    int k;
    auto begin = std::chrono::high_resolution_clock::now();

    for (k = 0; k < rows; ++k)
        for(int i = k + 1; i < rows; ++i) {
            double koef = -matrix[i][k] / matrix[k][k];
            cilk_for (int j = k; j <= rows; ++j)
                matrix[i][j] += koef * matrix[k][j];
        }

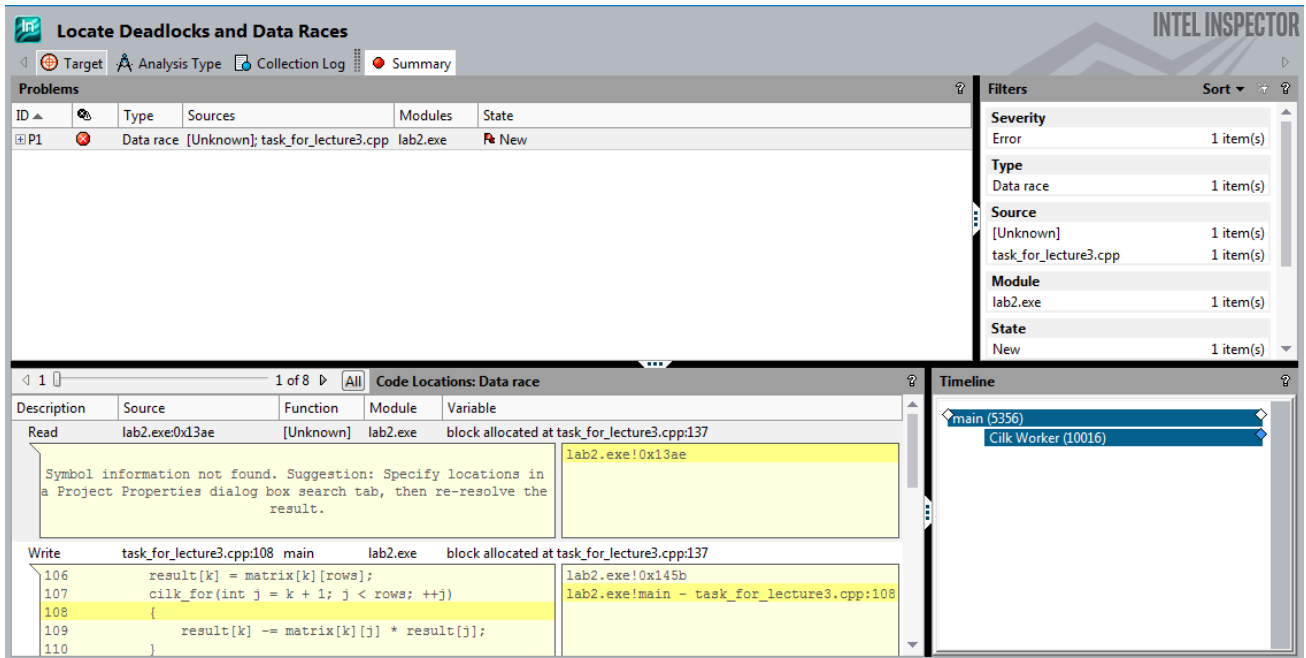
    auto end = std::chrono::high_resolution_clock::now();

    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];
    }
    return (int)std::chrono::duration_cast<std::chrono::microseconds>(end - be
}
```

Serial Gauss duration: 807899 microseconds  
Parallel Gauss duration: 3799353 microseconds  
Boost ratio: 0.212641

Скорость выполнения ожидаемо снизилась.

## №4



Инспектор обнаружил гонку данных. Исправляем:

```
int ParallelGaussMethod(double** matrix, const int rows, double* result)
{
    int k;
    auto begin = std::chrono::high_resolution_clock::now();

    for (k = 0; k < rows; ++k)
        cilk_for(int i = k + 1; i < rows; ++i) {
            double koef = -matrix[i][k] / matrix[k][k];
            for (int j = k; j <= rows; ++j)
                matrix[i][j] += koef * matrix[k][j];
        }

    auto end = std::chrono::high_resolution_clock::now();

    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];
    }
    return (int)std::chrono::duration_cast<std::chrono::microseconds>(end - be
}

```

Повторный анализ:

**Intel Inspector: Locate Deadlocks and Data Races**

**Problems**

ID	Type	Sources	Modules	State
P1	Data race	reducer_opadd.h	lab2.exe	New

**Filters**

- Severity: Error (1 item(s))
- Type: Data race (1 item(s))
- Source: reducer\_opadd.h (1 item(s))
- Module: lab2.exe (1 item(s))
- State: New (1 item(s))
- Suppressed

**Code Locations: Data race**

Description	Source	Function	Module	Variable
Write	reducer_opadd.h:265	reduce_wrapper	lab2.exe	0x2bf940
<pre> 263 *           reduce operation. 264 */ 265 void reduce(op_add_view* right) { this-&gt;m_value += rig 266 267 /** @name Accumulator variable updates. </pre>				
Write	reducer_opadd.h:265	reduce_wrapper	lab2.exe	0x2bf940
<pre> 263 *           reduce operation. 264 */ 265 void reduce(op_add_view* right) { this-&gt;m_value += rig 266 267 /** @name Accumulator variable updates. </pre>				

**Timeline**

- main (4044)
- Cilk Worker (9748)

## №5

Решение для тестовой матрицы параллельным методом:

Solution:

```

x(0) = 1.000000
x(1) = 2.000000
x(2) = 2.000000
x(3) = -0.000000

```

Сравнение решений последовательного и параллельного методов будем проводить при помощи следующей функции:

```

bool equals(double* a, double* b, size_t len)
{
    for (int i = 0; i < len; ++i) {

```

```
        if (std::fabs(a[i]-b[i]) > fabs(a[i] * .0001))  
            return false;  
    }  
    return true;  
}
```

Сравнение времени выполнения последовательного и параллельного методов:

```
Serial Gauss duration: 936415 microseconds  
Parallel Gauss duration: 746388 microseconds  
Do results match? ✓  
Boost ratio: 1.254596
```