№2

Вывод первоначальной версии программы:

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

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))
```

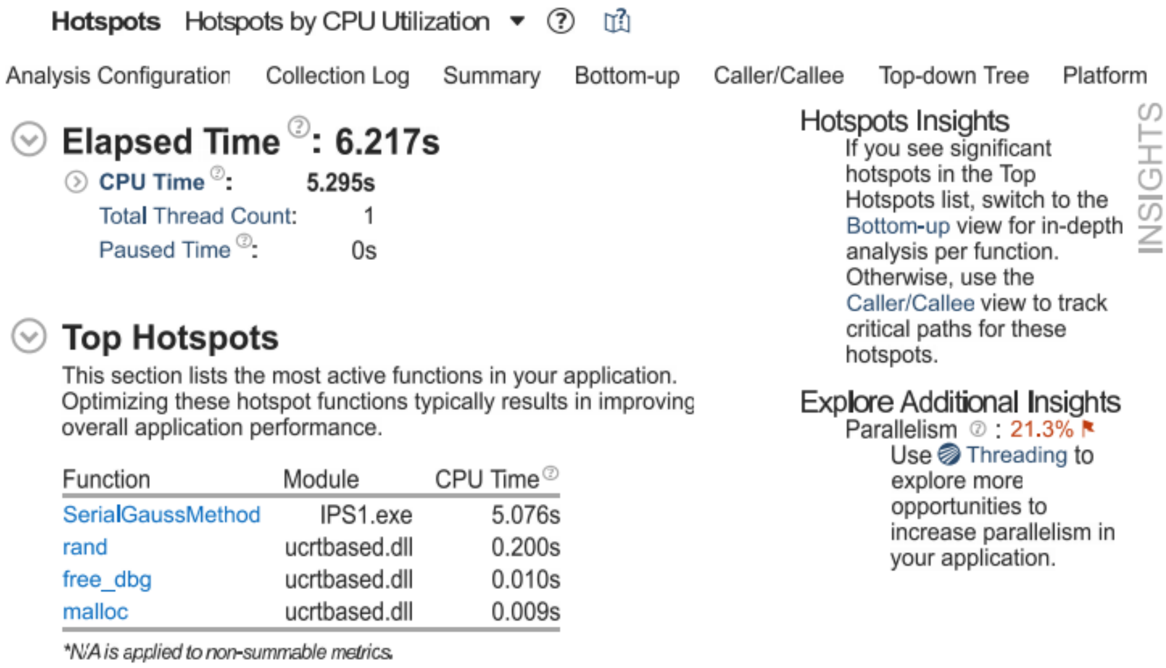
```
[ 1.00000000e+00  2.00000000e+00  2.00000000e+00 -3.70074342e-15]
```

Решения совпадают.

Время работы программы для матрицы размером `MATRIX_SIZE`:

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

№3



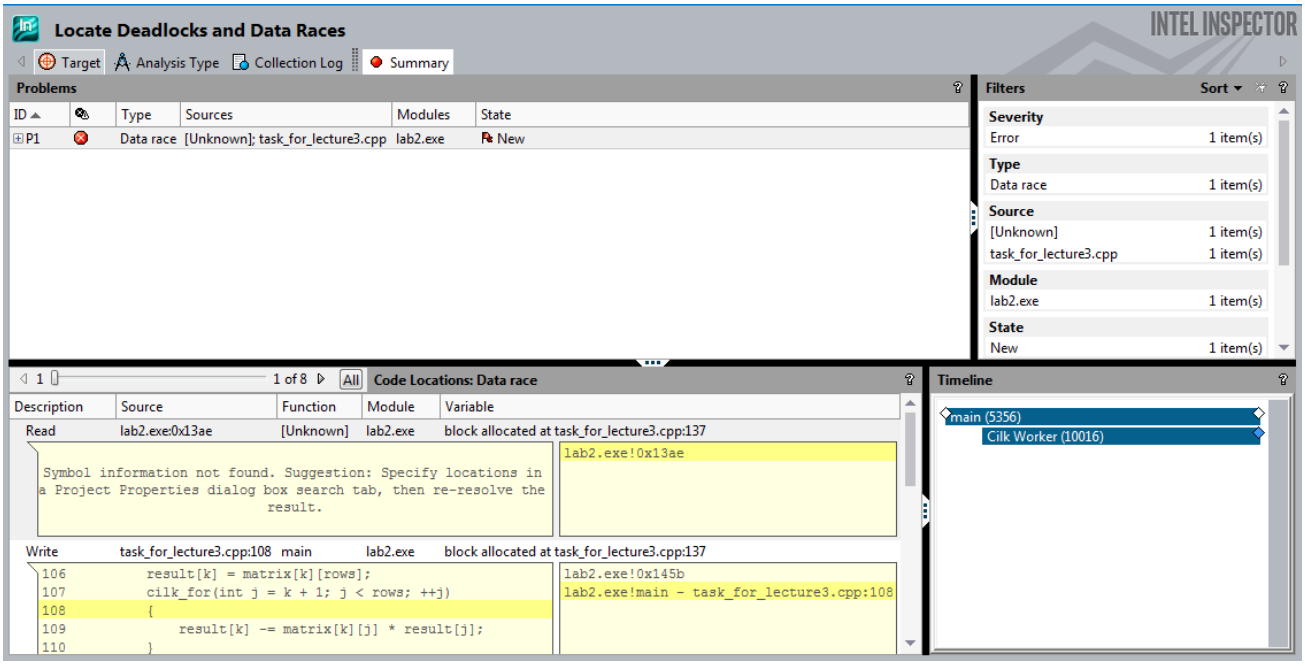
Напишем заведомо неправильную параллелизацию метода Гаусса:

```
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

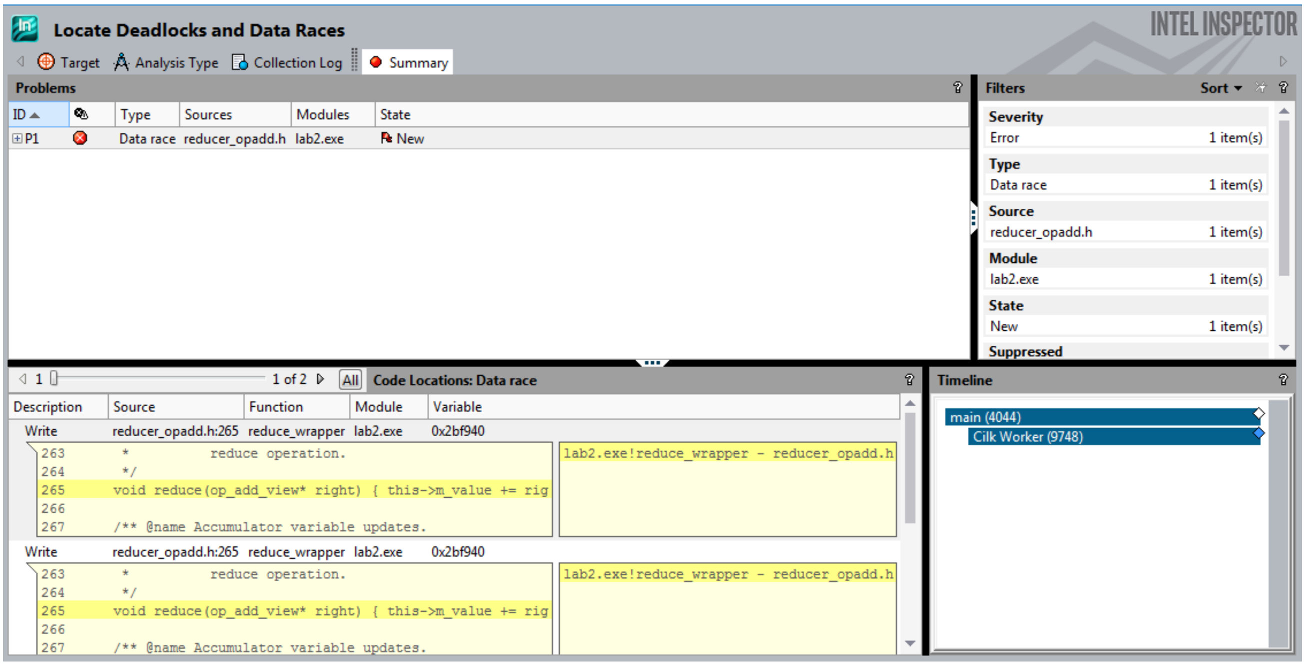


Инспектор обнаружил гонку данных. Исправляем:

```
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:



№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 времени выполнения последовательного и параллельного методов:

```
Serial Gauss duration: 766317 microseconds
Parallel Gauss duration: 616903 microseconds
Boost ratio: 1.242200
```